# Excel 2020

# SBORNÍK

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Studentská konference (letos online) s těmi nejzajímavějšími IT nápady, inovacemi a technickými řešeními

# Excel@FIT 2020

Studentská konference inovací, technologií a vědy v IT

6. ročník

Brno, 27.5.2020

Excel@FIT 2020

Konference Excel@FIT již po šesté nabízí studentům příležitost prezentovat své nápady a výsledky svých prací. Letošní rok je ovšem jiný. Rok 2020 žijeme všichni ve znamení boje s novým světovým fenoménem jménem koronavirus. Situace zasáhla snad do všech částí našich životů, tedy i do výuky, práce na studentských projektech či organizace naší konference. Osobní spolupráce a komunikace se studenty byla nahrazena komunikací přes video-hovory, přístup do laboratoří byl nejdříve znemožněn a následně velmi omezen atd. I přes různá omezení většina studentů tvrdě pracovala na svých projektech a my máme velkou radost z rozsahu a kvality studentských výsledků. V tomto sborníku najdete výběr těch nejlepších prací studentů FIT, kteří měli zájem své projekty zde prezentovat.

Opatření, která mimořádná situace přinesla, však bohužel znemožnila připravit konferenci Excel@FIT tak, jak jsme na ni byli zvyklí. Některé části programu jsme nuceni zcela zrušit, jako třeba přehlídku studentských prací, stánky technologických firem či obědový raut. Jiné části pak přenést do digitálního světa. Vybrané prezentace studentů, diskuzní panel se zástupci studentů i firem a slavnostní předávání cen odborných komisí, Partnerů či veřejnosti budou v tomto roce realizovány formou webináře. Je to výzva pro nás všechny a věříme, že vzniklé potíže zvládneme vyřešit. Snad budou negativa z tohoto způsobu setkání vyvážena kvalitním obsahem a zkušenosti s těmito technologiemi přinesou nové možnosti do dalších let.

Konference je věnována památce významného pedagoga Jiřího Kunovského, který byl velkou osobností Fakulty informačních technologií a příkladným pedagogem ve vztahu ke studentům. Děkuji za tuto tichou vzpomínku a přeji nám organizátorům Vaši trpělivost během konference a Vám i nám inspirativní zážitky z digitálního setkání s našimi šikovnými studenty a zástupci našich partnerů z průmyslu.

Partnerům tímto velice děkujeme za podporu konference i v tomto režimu.

**Víťa Beran**
Výkonný předseda rady Excel@FIT

## Organizátoři / Organizers

| | |
|---|---|
| Předseda rady Excel@FIT: | Prof. Ing. Tomáš Hruška, CSc. |
| Výkonný předseda rady Excel@FIT: | Ing. Vítězslav Beran, Ph.D.<br>Prof. Ing. Adam Herout, Ph.D. |
| Zástupce pro sponzory konference, organizace konference: | Mgr. Sylva Sadovská |
| Zástupce pro spolupráci s FIT: | Ing. Vítězslav Beran, Ph.D. |
| Webmaster: | Ing. Josef Žižka |
| Organizace konference, propagační předměty: | Ing. Petr Veigend |
| Správa příspěvků, organizace odb. panelů, recenze: | Ing. Igor Szőke, Ph.D. |
| Programová rada, organizace odborných panelů: | Ing. Martin Hrubý, Ph.D. |
| Styk s veřejností, propagační materiály: | Mgr. Hana Nečasová |
| Sběr materiálů: | Ing. Martin Fajčík |

## Programová rada / Program board

Ing. Martin Hrubý, Ph.D.
Ing. Igor Szőke, Ph.D.
Prof. Ing. Adam Herout, Ph.D.

## Odborné panely / Expert panels

Doc. Dr. Ing. Jan Černocký
Prof. Ing. Adam Herout, Ph.D.
Doc. Ing. Martin Čadík, Ph.D.
Ing. Igor Szőke, Ph.D.
Ing. Martin Kruták, KINALI
Ing. Lukáš Hübner, ThermoFisher Scientific
Ing. Miroslav Vomela, camea

Doc. Mgr. Adam Rogalewicz, Ph.D.
Prof. Ing. Tomáš Vojnar, Ph.D.
Dr. Ing. Petr Peringer
Doc. Ing. Vladimír Janoušek, Ph.D.

Ing. Michal Bidlo, Ph.D.
Doc. Ing. Vladimír Drábek, CSc.
Doc. Ing. Jan Kořenek, Ph.D.
Ing. Josef Strnadel, Ph.D.
Ing. Filip Kadlček, Ph.D., camea
Bc. Mykola Kaplenko, ThermoFisher Scientific

RNDr. Marek Rychlý, Ph.D.
Ing. Vladimír Bartík, Ph.D.
Ing. Vladimír Veselý, Ph.D.
Doc. Ing. Ondřej Ryšavý, Ph.D.

## Recenzenti / Reviewers

Ing. Vladimír Bartík, Ph.D.
Mgr. Lukáš Holík, Ph.D.
Ing. Michal Hradiš, Ph.D.
Ing. Radek Kočí, Ph.D.
Ing. Ondřej Lengál, Ph.D.
Ing. Tomáš Milet
Ing. Jaroslav Rozman, Ph.D.
RNDr. Marek Rychlý, Ph.D.
Ing. Aleš Smrčka, Ph.D.
Ing. Martin Veľas
Ing. Karel Beneš

RNDr. Milan Češka, Ph.D.
Doc. Dr. Ing. Otto Fučík
Ing. Matěj Grégr, Ph.D.
Ing. Martin Hrubý, Ph.D.
Doc. Ing. Jiří Jaroš, Ph.D.
Santosh Kesiraju
Doc. Ing. Jan Kořenek, Ph.D.
Ing. Zbyněk Křivka, Ph.D.
Mgr. Kamil Malinka, Ph.D.
Ing. Tomáš Martínek, Ph.D.
Ing. Petr Matoušek, Ph.D., M.A.

Ing. Ladislav Mošner
Santosh Kesiraju
Doc. Ing. Jan Kořenek, Ph.D.
Ing. Zbyněk Křivka, Ph.D.
Mgr. Kamil Malinka, Ph.D.
Ing. Tomáš Martínek, Ph.D.
Ing. Petr Matoušek, Ph.D., M.A.
Ing. Ladislav Mošner
Ing. Oldřich Plchot, Ph.D.
Ing. Libor Polčák, Ph.D.
Doc. Mgr. Adam Rogalewicz, Ph.D.
Doc. Ing. Ondřej Ryšavý, Ph.D.
Ing. Václav Šimek
Ing. Josef Strnadel, Ph.D.
Ing. Michal Španěl, Ph.D.
Doc. Ing. František Zbořil, Ph.D.

## Seznam partnerů / List of partners

### Generální partneři / General partners

### Hlavní partneři / Main patrners

**Konference je pořádána pod záštitou / Patrons of the conference**

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

## Oceněné práce / Awarded papers

### Práce oceněné odbornými panely / Papers selected by expert panels

Juraj Holub: Spätný preklad špecializovaných a pokročilých instrukčných sád nástrojom RetDec
Tomáš Dacík: Static Deadlock Detection in Frama-C
Monika Mužikovská: Detekce paralelních chyb ve víceprocesových programech
David Mikšaník: Inclusion of Regular Expressions with Counting
Simeon Borko: EnzymeMiner: Web Server for Automated Mining of Soluble Enzymes
Michal Ormoš: Infrastructure for Testing and Deployment of the Real-Time Localization Platform
Roman Bártl: Hardwarový indikátor oznámení pro telefony se systémem Android
Lukáš Dobiš: Automated human recognition from image data
Michal Piňos: Evoluční návrh konvolučních neuronových sítí
Dominik Vagala: Náhľad farby na stene pomocou rozšírenej reality v mobile
Pavol Karlík: Speech Enhancement with Cycle-Consistent Neural Networks
Tomáš Sýkora: Best possible speech recognizer on your own data
Dominik Gabzdyl: Counting Vehicles in Image and Video
Kateřina Trávníčková, Oldřich Kodym: Benchmarking medical segmentation models with limited training sets
Ondřej Zemánek: Counting Vehicles in Static Images
Patrik Goldschmidt: Adaptive SYN Flood Mitigation Based on Attack Vector Detection and Mitigation Process Monitoring
David Kozák: Enticing – Semantic Search Engine
Vladislav Bambuch: Platform for Cryptocurrency Address Collection
Tomáš Vlk: Využití Nix/NixOps pro průběžnou integraci a nasazení software při vývoji
Michael Polák: Behavior-Based Network Device Tracking

### Práce oceněné partnery konference / Papers selected by conference partners

Ondřej Zemánek: Counting Vehicles in Static Images
Monika Mužikovská: Detekce paralelních chyb ve víceprocesových programech
Tomáš Dacík: Static Deadlock Detection in Frama-C
Dominik Vagala: Náhľad farby na stene pomocou rozšírenej reality v mobile
Monika Mužikovská: Detekce paralelních chyb ve víceprocesových programech
Adam Ferencz: Počítání lidí v davu pomocí neuronových sítí s integrací do mapy
Dominik Vagala: Náhľad farby na stene pomocou rozšírenej reality v mobile
Adam Ferencz: Počítání lidí v davu pomocí neuronových sítí s integrací do mapy

### Práce oceněné odbornou veřejností / Papers selected by the expert public

Martin Gaňo: Improving robustness of neural networks against adversarial examples
Dominik Vagala: Náhľad farby na stene pomocou rozšírenej reality v mobile
Filip Januš: Pokročilá evaluace privátnosti na sociálních sítích
Monika Mužikovská: Detekce paralelních chyb ve víceprocesových programech
Vladislav Bambuch: Platform for Cryptocurrency Address Collection
Adam Ferencz: Počítání lidí v davu pomocí neuronových sítí s integrací do mapy

## Práce přijaté do sborníku konference / Papers accepted into the proceedings

*Příspěvky na Excel@FIT procházejí recenzním řízením, ale nejsou považovány za finální publikaci výsledku. Příspěvek na Excel@FIT se nevylučuje s následnou publikací na vědecké či odborné konferenci nebo v časopise. Autorům kvalitních příspěvků na Excel@FIT a jejich vedoucím se naopak taková publikace doporučuje.*
*Submissions at Excel@FIT go through a reviewing proces, but they are not considered to be final publications of the result. Contributions presented at Excel@FIT do not exclude further publication at a scientific or professional conference or in a journals. Contrary to that, authors of high-quality contributions at Excel@FIT and their supervisors are encouraged to prepare such a contribution.*

## Seznam prací / List of papers


**Platform for Cryptocurrency Address Collection**
**Vladislav Bambuch** . . . . . 3

**Spätný preklad špecializovaných a pokročilých instrukčných sád nástrojom RetDec**
**Juraj Holub** . . . . . 11

**Využití Nix/NixOps pro průběžnou integraci a nasazení software při vývoji**
**Tomáš Vlk** . . . . . 18

**Monitorování IoT sítě pomocí SNMP**
**Kateryna Polishchuk** . . . . . 27

**A Library for Tree Structures Analysis and Similar Structures Generation**
**Sergey Panov** . . . . . 34

**Hardwarový indikátor oznámení pro telefony se systémem Android**
**Roman Bártl** . . . . . 42

**EnzymeMiner: Web Server for Automated Mining of Soluble Enzymes**
**Simeon Borko** . . . . . 49

**Behavior-Based Network Device Tracking**
**Michael Polák** . . . . . 55

**Náhľad farby na stene pomocou rozšírenej reality v mobile**
**Dominik Vagala** . . . . . 61

**Simulace sněhových a ledových struktur**
**Ondřej Čech** . . . . . 67

**Pokročilá evaluace privátnosti na sociálních sítích**
**Filip Januš** . . . . . 72

**Static Deadlock Detection in Frama-C**
**Tomáš Dacík** . . . . . 80

**Virtual Tour of FIT for Oculus Quest**
**Michal Janů** . . . . . 87

Speech Enhancement with Cycle-Consistent Neural Networks
Pavol Karlík . . . . . 91

Best possible speech recognizer on your own data
Tomáš Sýkora . . . . . 100

Estimation of Emotions from a Text and Analyzing Tweets
Aneta Dufková . . . . . 107

Counting Vehicles in Image and Video
Dominik Gabzdyl . . . . . 112

Enticing – Semantic Search Engine
David Kozák . . . . . 119

Detekce paralelních chyb ve víceprocesových programech
Monika Mužikovská . . . . . 127

Rekonštrukcia poškodených častí odtlačkov prstov s využitím neurónových sietí
Michael Halinár . . . . . 135

Benchmarking medical segmentation models with limited training sets
Kateřina Trávníčková Oldřich Kodym . . . . . 139

Infrastructure for Testing and Deployment of the Real-Time Localization Platform
Michal Ormoš . . . . . 146

Framework pro tvorbu informačních systémů nad Universal Windows Platform
Jan Rajnoha . . . . . 152

Evoluční návrh konvolučních neuronových sítí
Michal Piňos . . . . . 159

Ultimate Arcade Suitcase
Lukas Erlich . . . . . 168

Improving robustness of neural networks against adversarial examples
Martin Gaňo . . . . . 173

Počítání lidí v davu pomocí neuronových sítí s integrací do mapy
Adam Ferencz . . . . . 181

Counting Vehicles in Static Images
Ondřej Zemánek . . . . . 188

Inclusion of Regular Expressions with Counting
David Mikšaník . . . . . 195

Adaptive SYN Flood Mitigation Based on Attack Vector Detection and Mitigation Process Monitoring
Patrik Goldschmidt . . . . . 203

Automated human recognition from image data
Lukáš Dobiš . . . . . 211

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

# Platform for Cryptocurrency Address Collection

Vladislav Bambuch*

**Abstract**
The goal of this work is to build a platform for collecting and displaying metadata about cryptocurrency addresses from public and also dark web. To achieve this goal, the author uses web parsing technologies written in PHP. Challenges accompanying a website parsing are solved by scaling capabilities of Apache Kafka streaming platform. The modularity of the platform is accomplished by microservice architecture and Docker containerization.
The work creates a unique way how to search for potential crypto criminal activities, that appeared outside of the blockchain world, by building a web page application on top of this platform (that serves for managing the platform and exploring the extracted data). The platform architecture allows adding loosely coupled modules smoothly where the Apache Kafka mediates communication of the modules.
The result of this article is meant to be used for cybercrime detection and prevention. Its users can be law enforcement authorities or other agencies interested in reputations of cryptocurrency addresses.
**Keywords:** web scraping — cryptocurrencies — crypto crime detection — microservices — apache kafka — data streaming

**Supplementary Material:** Demonstration Video — Downloadable Code

*xbambu03@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

## 1. Introduction

The goal of this work is to build a web-scraping platform for a collection of criminal related cryptocurrency addresses. The platform introduces high-throughput web mining features using data-streaming techniques and stores the extracted metadata into persistent storage for further analysis. On top of the system is a web-based application that is meant to be used by a law enforcement authorities searching for proof of crypto criminal activities.

Using cryptocurrency transactions, a criminal can exchange money for illegal services [1] extremely quickly due to its pseudonymization features. We can only obtain the information about who is an owner of a cryptocurrency address when the person chooses to expose it to the public. Due to the natural behaviour of Internet websites and their dynamic content changes, we need to look up for this type of information regularly. This topic is worthy of exploration as we do not have sufficient tools freely available that would address the mentioned difficulties.

Currently, there are many websites focusing on displaying cryptocurrency addresses[1], some of them also contain owners for particular addresses subset[2] or general discussions about cryptocurrencies[3]. The

[1] https://bitinfocharts.com
[2] https://www.walletexplorer.com
[3] https://bitcointalk.org

data on those websites can help to clarify what happened outside of blockchain and to map pseudonymous crypto-addresses to real users. There are also websites[4] focusing on collecting addresses seen in fraud emails and other forms of extortion activities. On the other hand, we lack a web application that would unify and link all the data from mentioned websites and provide possibility to search in them. The result of this paper aims to fill this blind spot.

Given the fact that there are many websites with interesting data from Open-source intelligence (OSINT) point of view and do not expose any Application Programming Interface (API), this platform enables smooth integration of parsers for particular websites. Those parsers can be implemented as results of school projects or theses and without dependency on a programming language.

The potential amount of data this platform is going to process is enormous. This requires designing a robust architecture that provides parallel data processing using scalability features and fault-tolerant properties. All modules need to be excessively supervised by a monitoring tool, and its metrics displayed in a coherent dashboard. The monitoring enables alerting when some soft-dependency is broken, but without the need of stopping all modules. This approach follows the microservices architecture design [2].

At the output of this platform lies scraped cryptocurrency metadata that can be searched with a simple web-based application on top of the system. The application also allows maintenance of the underling backend layer and scheduling of repetitive jobs. Law enforcement authorities and, in general, all agencies interested in reputations of specific cryptocurrency addresses can use this system.

This work is a part of the project *Integrated platform for analysis of digital data from security incidents*[5], developed at Brno University of Technology, and it is planned to integrate it with its other components.

## 2. Challenges The Platform Faces

In this chapter, there are explained several challenges the platform is dealing with and how they are solved.

This paper tackles many obstacles from a data collection point of view, as well as using reliable processing and transformation until the results can be monitored, stored and kept in persistent storage. Each of these obstacles has to align with the intention to implement modules loosely coupled, language-independent, and as general as possible.

The following challenges are discussed in this chapter:

- Scraping unstructured data;
- Rate-limiting;
- Browser fingerprinting;
- Processing vast amount of data;
- Sharing parsed data across processes;
- Parallel task execution;
- Data quality of scraped results.

### 2.1 Cryptocurrency Websites Parsing

The reason why we want to scrape data from public sources is quite simple. We do not have enough information to successfully find relations between data in the blockchain and real-world entities. It is necessary to mention that the extraction of the data is aligned with OSINT initiative, and therefore, all of the information is publicly available. That being said, privacy issues are not related to this topic.

The idea is to collect as much data as possible and to perform it fast and frequently. The disadvantages to this idea are application security measures, such as rate-limiting, browser fingerprinting or traffic throttling. There are also issues with dynamic structural changes of the websites and currently highly popular isomorphic web applications [3].

The limitation of requests a website is receiving can be overcome by IP address pool and performing requests to a single website on behalf of different IP addresses. This behaviour can be accomplished by the help of proxy servers and rotation of addresses they expose.

Browser fingerprinting is a technique employing HTTP headers like `User-Agent`, `Connection`, `Encoding` or `Language` and other connection parameters to find a network entity even if it swaps IP addresses. The `User-Agent` is the essential header in this scenario and therefore changing its value is an effective way how to deal with this challenge [4].

Both of the mentioned principles are handled by a proxy module. Implementation of the module is delivered by project Lemmit that was created as a result of thesis *Automated Web Analysis and Archivation* [5] which is integrated into the platform. The project also consists of archival module executing JavaScript code and extracting fully-loaded Document Object Model (DOM). Integration of this feature into the work suppresses the issue with isomorphic websites.

Every scraper in this work is built using Goutte [6]. Goutte is a library for web crawling and scraping that is written in PHP. In the PHP community, Goutte is some-

[4] https://www.bitcoinabuse.com
[5] https://www.fit.vut.cz/research/project/1063/.en

what of standard for scraping techniques. It is truly easy to load a web page, filter some HTML elements, iterate over them and extract all necessary data. This library cannot handle isomorphic web applications, but it is solved by a combination of the mentioned smart proxy server.

Finally, the structural changes of scraped websites are handed by extensive monitoring capabilities of all requests and expecting responses. Therefore, every detected DOM change triggers an alert.

### 2.2 The Platform Data Processing

As it was mentioned before, the idea is to collect publicly available metadata as fast as possible and without being blocked by application firewalls. To conquer the problem, the author of this platform introduces massive parallel processing using the Apache Kafka streaming platform. An illustration of how particular Kafka APIs communicate together is present in Figure 1.

The system consists of several loosely coupled modules, performing a single task. Every web-scraping module is subscribed to a data stream and listens for incoming messages – URLs that need to be parsed. When a message is processed, its result is sent to an output data stream by which are other modules notified. By this principle, modules can interact asynchronously and share the same information between multiple simultaneously running processes. The core of the described communication is the Apache Kafka, that is described in this section.

Kafka is a distributed streaming platform, allowing to publish and subscribe to particular data flows. It is used for building reactive event-driven applications and also for building data pipelines for reliable communication between systems. Kafka provides high-speed and fault-tolerant data processing for which has been chosen in many enterprise solutions[6].

In comparison to a traditional message-queuing system, Kafka is capable of storing messages persistently and does not delete them when they are read. That enables reproducing the whole sequence of events if needed and reading one message multiple times using different logic of different data consumers [7]. This technology combines two concepts, *queuing* and *publish-subscribe*, and solves their issues using *Kafka Consumer Group*. That means advantages from both models – *in-order delivery* and *parallel processing* – are merged together [8].

This data streaming platform can be operated as a cluster on multiple servers which proves this technology is an excellent choice for fault tolerance and data replication. One cluster stores stream of records aggregated into categories called *topics*. Each record has a *key*, a *value* and a *timestamp* [9].

[6]https://kafka.apache.org/powered-by

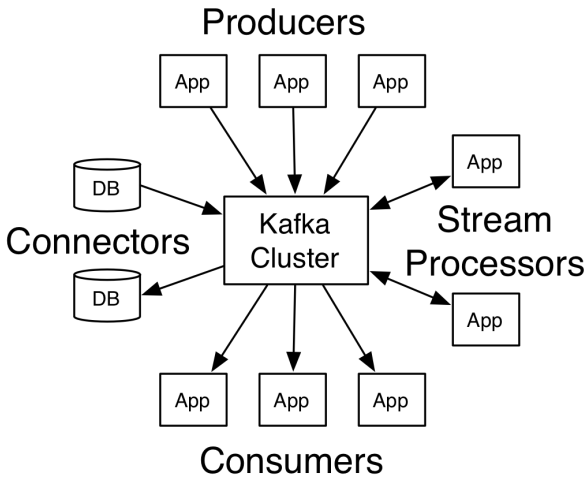



**Figure 1.** The Kafka architecture diagram is showing how many types of modules can be connected together through this data-streaming platform.

One Kafka topic can be distributed across multiple partitions. The number of partitions goes hand in hand with the level of parallelism.

### 2.3 Storing Crawling Progress

Every website that is supposed to be scraped is crawled at first, and all required URLs are extracted. Individual URL addresses are stored with additional metadata. For example, if a page has been parsed already or if a page is the last one from a sequence of pages. This metadata can be used for running scrapers on appropriate pages or for other robots to know where to start a new round of crawling a website.

Storing the crawling process provides better visibility on performing tasks and speeds up the whole data flow.

## 3. The Platform Architecture Overview

This chapter contains detailed information about the architecture of the platform and what technologies are used to build loosely coupled, highly modular, data-streaming system.

The platform (depicted in Figure 2) consists of the following modules:

- Web crawlers – collect URL addresses from websites, store them into a database and shares them with other modules through Kafka topics.
- Web scrapers – parse web pages according to crawled URL addresses and extract interesting

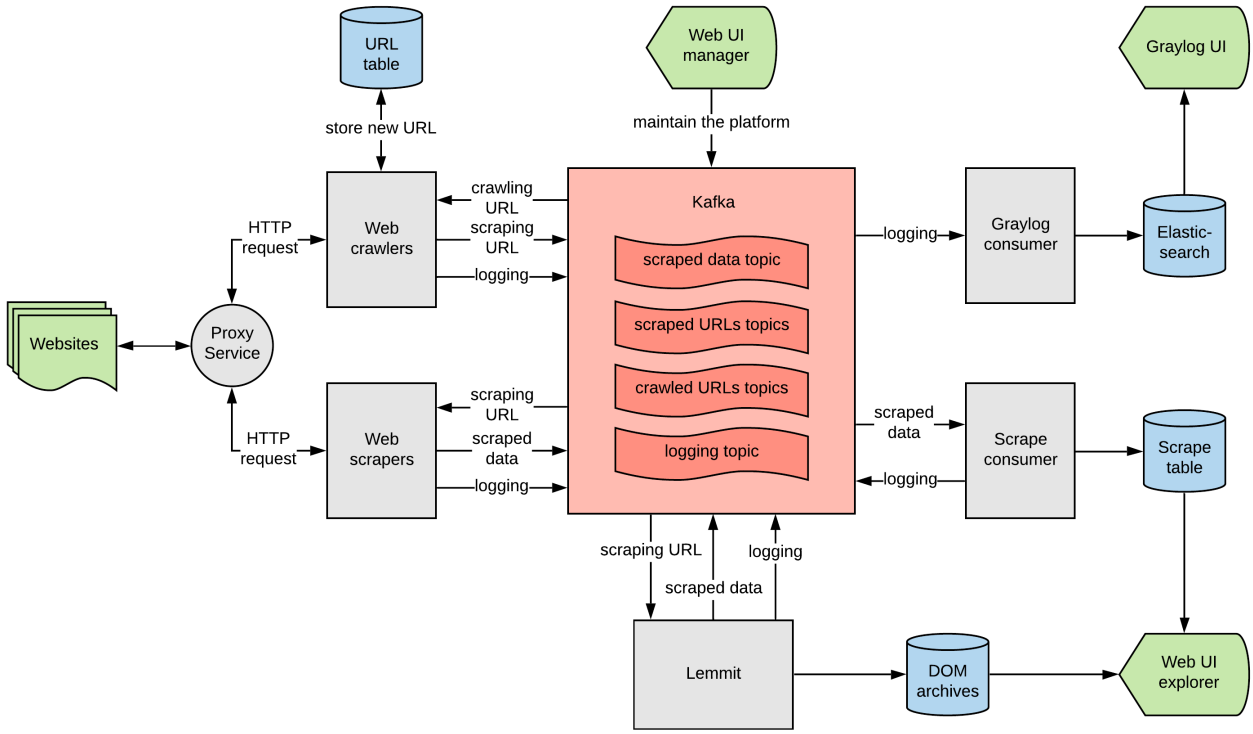



**Figure 2.** The platform architecture diagram. The blue elements are databases, the green are associated with a web browser, the grey are data-processing modules and the red one is the core – Apache Kafka.

metadata. The data are streamed back to Kafka for further processing.

- Scrape consumer – consumes resulting metadata from all scrapers and stores them into the database with a unified schema.
- Proxy service – allows making HTTP requests to a single website from multiple IP addresses at the same time. It ensures application firewalls will not block the scrapers. The module is provided by the project *Automated Web Analysis and Archivation* [5].
- Apache Kafka with Zookeeper[7] – the core module of the entire system. It is a scalable, robust and fault-tolerant streaming platform that assures all the modules can communicate in a simple and unified way.
- PostgreSQL – stores the resulting scrapes and information about processing statuses of all websites in *URL table*, *DOM archives* and *Scrape table*.
- Graylog monitoring tool – web-based monitoring tool for all parts of the platform.
- Lemmit – it gets URL addresses from Kafka topics so that archives whole DOM structure of a webpage. The DOM structure is used as evidence that data were present on a particular webpage at the time of scraping [5].
- Web UI – allows an admin to manage the platform, to run parsing jobs manually and to schedule them. It also empowers a user to inspect the scraped data in order to see cryptocurrency activities that appeared outside of blockchain world. Every scraped information is linked to the proof mentioned above.

### 3.1 Platform Modularity

Platform is written in PHP with use of Laravel framework. In this work, the framework is employed for web scraping, communication with the PostgreSQL database and for building CLI commands that manage separate modules. Scheduling features in the platform are implemented by Laravel Scheduler.

Even though there is used single programming language almost for the whole platform, it does not mean all future extensions has to be written in it. The idea is that all core modules are language independent and communicate through a unified API. It means a specific web scraper module can be implemented by any programming language and reuse common modules if it meets the API requirements. This principle allows fast prototyping of new scraping modules without the need to understand the complexity of the entire system. That being said, the problematic part of website

[7] https://zookeeper.apache.org

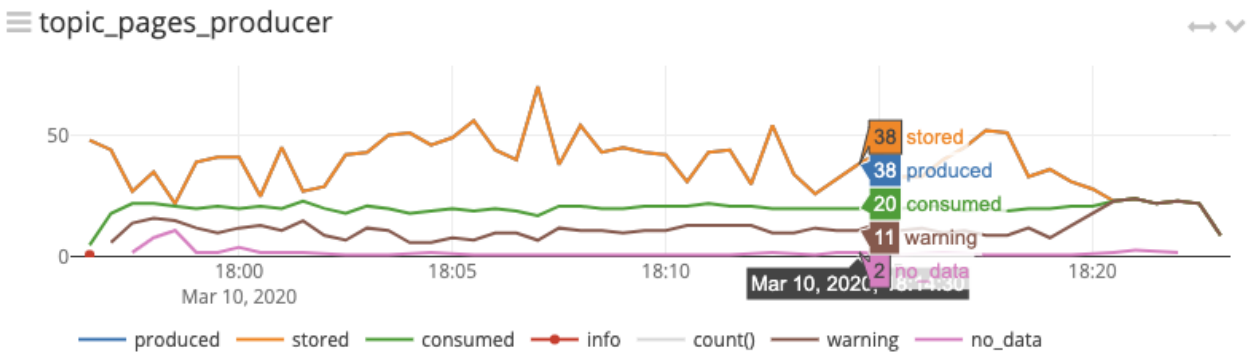



**Figure 3.** Monitoring example of one of bitcointalk.com scrapers. This graph was taken from a dashboard containing eleven, quite similar bitcointalk-monitoring graphs.

scraping is fully covered by this work and the more straightforward parts can be added seamlessly.

The archiving and proxy modules, database or monitoring tool, can also be smoothly changed to different implementation or other technology. This enables the possibility to keep track of technology alternatives and swap them if needed. The architecture with loosely coupled parts allows doing that without unnecessary changes.

Platform modularity is powered by Docker containers. The modules are divided into several categories, that can be seen in the platform diagram 2:

- Web crawlers/scrapers;
- Apache Kafka;
- PostgreSQL database;
- Graylog with supplementary databases;
- Web server powering the web application;
- Proxy server;
- Lemmit.

Every mentioned category runs in a separate Docker container and in the case of Web crawlers/scrapers it is expected to have tens of containers running simultaneously. All the modules are managed by Docker Compose that allows defining container dependencies, internal network communication and many other properties[8].

### 3.2 Data Layer Architecture

In this work, the author uses PostgreSQL database engine for the following purposes:

- saving all extracted data and their metadata in a unified structure,
- for metadata about individual web pages,
- for additional information about cryptocurrencies and websites categorization.

[8] https://docs.docker.com/compose

PostgreSQL is associated with *URL table*, *Scrape table* and *DOM archives* in the diagram 2.

The data layer also contains MongoDB and Elasticsearch technologies. Both of them are related to Graylog monitoring tool where MongoDB is used to store configuration files and Elasticsearch keeps all the logs produces by the platform. The Graylog-related databases correspond to *Elasticsearch* module in 2.

### 3.3 Unified Database Schema for All Parsers

It is crucial to keep data quality of scraped content at the highest possible level. Otherwise, any additional processing can be extremely difficult.

The database schema for storing the scraped data consists of particular tables:

- Owners – contains re-identified owners of crypto wallets;
- Identities – there are stored metadata about a page from where an owner and its address has been extracted;
- Addresses – contains scraped crypto addresses and their metadata.

## 4. The Platform Monitoring

This chapter focuses on monitoring of the entire platform and describes how critical this step is in software development.

Monitoring of the final product is one of the essential steps in software development. Without proper real-time behaviour analysis and alerting system, it is nearly impossible to maintain and operate complex systems [10].

This work uses Graylog tool for overseeing the entire platform. All modules stream logs into this tool. Graylog uses Elasticsearch DB to store the logs and MongoDB for managing configuration files. The tool is capable of displaying metrics generated from logs, creating alerts, dashboards, investigating log streams

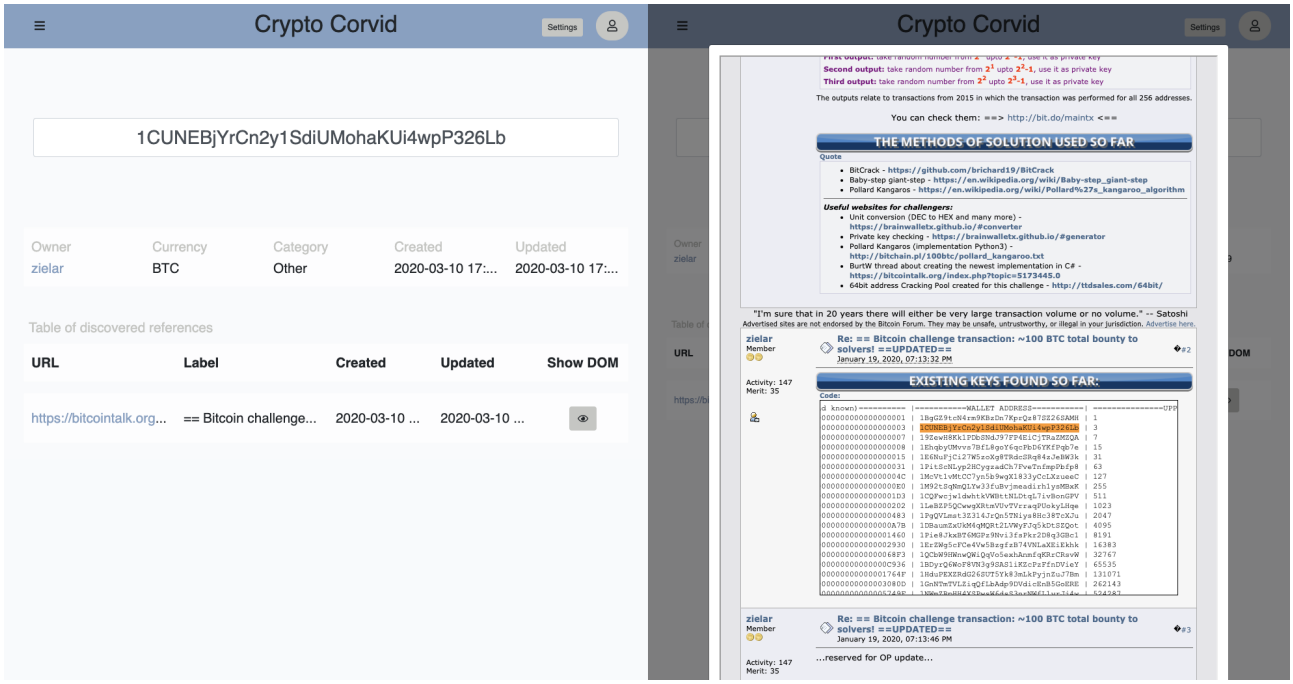



**Figure 4.** On the left screenshot, there are metadata as a result of a cryptocurrency address lookup. The right side shows a copy of a DOM associated with the searching address. The DOM copy is displayed after clicking on the ”Show DOM” button.

and has many other features that are useful for monitoring such complex systems. Figure 3 shows an example of a graph with metrics from scraping bitcointalk.com. The Kafka module produces the following metrics:

- stored – number of records stored into PostgreSQL database;
- produced – number of messages streamed into Kafka output topic;
- consumer – number of messages consumed from Kafka input topic;
- warning – number of connection warnings, in this case, it is when the scraper hits rate-limits of a particular website;
- other less exciting metrics.

## 5. Searching In The Results

This chapter introduces a web-based application that is capable of searching in the scraped metadata and scheduling repetitive scraping jobs.

The described platform is capable of generating a massive amount of cryptocurrency metadata. Only an intuitive search engine with a friendly user interface can maximize the data usage though. To reach the maximal potential of the scraped data, the author also designed and implemented a simple web-based application in PHP and Vue.js as a part of the work.

The web search engine has three major use-cases where a user can search for the following information and receive these properties:

- a cryptocurrency address;
  - category – Exchange, Mining pool, Person, Scam, etc.;
  - currency – BTC, LTC, ETH and others;
  - owner – an identified internet entity;
  - references – what web pages contain the address;
  - timestamps – when was the address scraped for the first time and when was updated at the last time.
- a source of a scraped data;
  - URL – web address of the source;
  - type – Web forum, Social network, Abuse report tool etc.;
  - addresses – which addresses were scraped from the source.
- an owner or wallet.
  - category – Exchange, Person etc.;
  - sources – which websites contain a mention of the owner;
  - addresses – list of addresses assigned to the wallet.

Figure 4 shows screenshots of the implemented web-based search engine. The engine displays results from search by a bitcoin address. The result discovers hypothetical user `mOtex2` that is classified

as an abuser, and also there are displayed several activities, associated with the user. Every activity is associated with a DOM copy from the time of scraping so a user of this application can see the proof from where was the information taken. The DOM copies are provided by archival capabilities of integrated project [5].

## 6. Evaluating The Platform

This chapter describes several ways of how the platform is evaluated and possible testing improvements that might be implemented in the future.

When the infrastructure part of the platform is executed, it takes around one minute to get the data streaming module, with two monitoring tools and databases, prepared for operation. After that, the scraping/crawling modules can be triggered.

Every crawler stores an URL metadata into a database and streams them into Kafka topic simultaneously (note, this can be simplified by using Confluent Kafka Connectors[9]). This is the first touchpoint where we can verify the behaviour. First of all, we know the numbers from the monitoring 3, but we can also perform SQL query in the database as well as in the Kafka, so all the numbers have to match for a specific time–topic–crawler combination. After performing this, we can be sure the data are not lost during the process. The similar evaluation can be done for all the web-scrapers as they also stream/store data into two destinations, and all the scrapers are monitored.

The second touchpoint is verifying whether the scraped data are actually associated with the correct webpage. Currently, this is manual work and can be surely automated. We can use the web application, built on top of the platform, to search for a specific crypto address and click on the link, associated with the search result 4. The address has to be present on the page.

Due to the fact that some websites were scraped by other projects at FIT BUT in previous years, we can also compare the older data with currently scraped results. This touchpoint is just a theory, and no such a cross-project comparison has been performed yet.

The very interesting finding would be to compare how much time it takes when the web scraping is performed sequentially and with the use of Kafka parallelism. Currently, this analysis is also not possible because the platform is missing a connection with the smart proxy that is using an IP address pool. It means, the scrapers work in parallel, but they have to wait in order to not to hit rate-limiting.

This paper does not conduct any results in terms of the platform performance. These results are associated with the performance of the Apache Kafka covered by the study *Kafka versus rabbitmq* [8].

## 7. Conclusions

The goal of this work was to implement a platform for collecting cryptocurrency addresses and web application for managing this platform. The platform meant to be highly modular with monitoring for each module and suppose to utilize scalability features. The core of this platform should parse interesting web pages containing cryptocurrencies metadata and store the data into storage with a unified database scheme. The data should be extracted from publicly available sources according to the definition of OSINT.

The solution had to be platform-independent. The platform independency supposes to be achieved by Docker containerization and using loosely coupled modules communicating through the Apache Kafka streaming platform.

The author successfully designed and implemented the platform with the aim of modularity and easy addition of new webpage parsers. The whole platform composes from multiple Docker containers. The parsing core of this platform consists of several tasks that are scheduled from a web application and all of them are monitored through Graylog service. The platform uses PostgreSQL as a persistent data storage.

The processing pipeline outputs metadata about cryptocurrency addresses that are used to searching for activities that happened outside of blockchain world. For the searching purposes, there was implemented web-based application. The gained information can lead to crypto criminal activities detection. With a combination of integrated project *Automated Web Analysis and Archivation*, the created system is able to provide legal evidence of scraped data validity.

This work solves most of the web parsing issues and enables seamless extensibility of scraping modules that can be implemented during networking courses here at BUT FIT or as part of Bachelor's or Master's thesis. Currently, all the scrapers cannot reach the maximal speed due to rate-limiting because of the missing proxy server. The proxy will be implemented later.

The core functionality is meant to be published as open-source software. The author plans to continue with further development and to connect all parts of project [5] with this platform.

[9] https://www.confluent.io/connectors

## Acknowledgements

I would like to thank my supervisor Mr. Ing. Vladimír Veselý, Ph.D. for his professional guidance and Ing. Tomáš Kocman for help with the integration of his master's thesis.

## References

[1] Dante Disparte. Crypto crime is taking a violent turn. online, Apr 2020. https://www.forbes.com/sites/dantedisparte/2019/01/28/crypto-crime-is-taking-a-violent-turn.

[2] Wilhelm Hasselbring. Microservices for scalability. Keynote Talk Abstract, 3 2016.

[3] Megan Mary Jane. How to bypass anti-scraping techniques in web scraping. online, Apr 2020. https://bigdata-madesimple.com/how-to-bypass-anti-scraping-techniques-in-web-scraping/.

[4] Pierre de Wulf. A guide to web scraping without getting blocked in 2020. online, Apr 2020. https://www.scrapingbee.com/blog/web-scraping-without-getting-blocked.

[5] Tomáš Kocman. Automated web analysis and archivation, 2019.

[6] FriendsOfPHP. Goutte, a simple php web scraper. online, Apr 2020. https://github.com/FriendsOfPHP/Goutte.

[7] Hendrik Swanepoel. A super quick comparison between kafka and message queues. online, Apr 2020. https://hackernoon.com/a-super-quick-comparison-between-kafka-and-message-queues-e69742d855a8.

[8] Kyumars Sheykh Esmaili Philippe Dobbelaere. Kafka versus rabbitmq: A comparative study of two industry reference publish/subscribe implementations: Industry paper. *ACM International Conference on Distributed and Event-based Systems*, (11):227–238, 2017.

[9] Apache.org. Kafka introduction. online, Apr 2020. https://kafka.apache.org/intro.

[10] Gregor Scheithauer Matthias Winkler, Jorge Cardoso. Challenges of business service monitoring in the internet of services. *International Conference on Information Integration and Web-based Applications Services*, (10):613–616, 2008.

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

# Spätný preklad špecializovaných a pokročilých inštrukčných sád nástrojom RetDec

Juraj Holub*

**Abstrakt**
V dnešnej dobe je proces analýzy nebezpečného softvéru dôležitou súčasťou informačných technológií. Jedna z kľúčových techník je spätný preklad škodlivých binárnych programov. Spätný preklad je komplexný proces, ktorým sa zaoberá niekoľko projektov. Projekt RetDec sa zameriava na flexibilný návrh a riešenie spätného prekladača s možnosťou znovupoužiteľnosti. Ide o open-source projekt vedený firmou Avast. Tento článok sa zaoberá návrhom nového rozšírenia pre spätný prekladač RetDec v oblasti podpory špecializovanej inštrukčnej sady pre jednotku FPU, ktorá je súčasťou procesorovej architektúry x86.

**Kľúčové slová:** RetDec, decompiler, Avast, reverse engineering, x86, FPU

**Priložené materiály:** RetDec Demonstration Video — Downloadable Code

*xholub40@stdu.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

## 1. Úvod

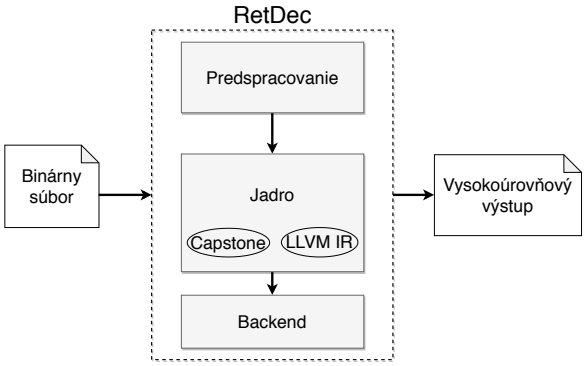
Táto práca sa zaoberá využitím reverzného inžinierstva v oblasti softvérových technológií. Reverzné inžinierstvo je všeobecne metóda získavania informácií alebo plánov o akýchkoľvek objektoch vytvorených človekom. V oblasti informačných technológií je význam tejto disciplíny najmä v rámci kybernetickej bezpečnosti. Táto technika je využívaná tvorcami škodlivého softvéru (tzv. malvér). Malvér využíva reverzné inžinierstvo na získavanie citlivých informácií o operačnom systéme s potenciálnym cieľom získať kontrolu nad zariadením. Ďalšia rozšírená oblasť je softvérové pirátstvo, kedy sa útočník snaží prelomiť ochranu komerčného digitálneho obsahu ako sú knihy, filmy, hudba, hry alebo rôzne platené programy. Na druhej strane môže pomôcť práve pri analýze malvéru za účelom zvýšenia bezpečnosti voči danému softvéru. Jedna z kľúčových techník je analýza malvéru pomocou programu všeobecne nazývaného spätný prekladač. Spätný prekladač je program, ktorý analyzuje spustiteľné binárne súbory a zrekonštruuje vysoko úrovňový výstup, napríklad v podobe grafu alebo kódu v programovacom jazyku. [1]

V dnešnej dobe existuje niekoľko projektov spätných prekladačov. Projekt RetDec sa zameriava na vytvorenie open-source nástroja, ktorý je rozdelený na viacero knižníc. Takýto návrh má za cieľ umožniť znovupoužiteľnosť jednotlivých nástrojov spätného prekladača. Cieľom tejto práce je navrhnúť optimalizačnú komponentu pre tento spätný prekladač. Navrhovaná komponenta optimalizuje preklad inštrukčnej sady jednotky FPU pre architektúru x86.

## 2. Spätný prekladač RetDec

RetDec (Retargetable Decompiler[1]) si dáva za cieľ vykonávať spätný preklad nezávisle od procesorovej

[1] Projekt RetDec: https://retdec.com/




**Obrázok 1.** Schéma architektúry spätného prekladača RetDec a technológie, ktoré používa.

architektúry, operačného systému a formátu spustiteľných binárnych súborov. Obrázok 1 zobrazuje architektúru prekladača a technológie použité v jednotlivých častiach prekladača. Kľúčovú technológiu tvorí LLVM a Capstone[2]. Prekladač je štruktúrovaný do troch hlavných blokov a každý blok sa skladá z množstva menších knižníc. Tieto bloky sú zreťazené v nasledujúcej sekvencii:

1. **Predspacovanie** zjednotí a zanalyzuje binárne súbory a extrahované metadáta sú vstupom pre jadro prekladača.
2. **Jadro** prekladača vytvára vnútornú reprezentáciu kódu v medzi jazyku (anglicky Intermediate Representation, ďalej len IR). Program aplikuje desiatky optimalizácií, ktoré transformujú IR. Optimalizovaná IR je výstupom tohto bloku.
3. **Backend** aplikuje rôzne optimalizácie s cieľom zvýšiť zrozumiteľnosť kódu a na záver vygeneruje výstup v jazyku C.

## Knižnica LLVM

Pre prácu s IR v jadre prekladača sa používa knižnica LLVM. LLVM definuje bežnú, nízkoúrovňovú reprezentáciu podobnú assamblerovskej rodine inštrukčnej sady RISC. Zároveň ale poskytuje informácie užitočné pre efektívne analyzovanie a optimalizovanie IR s cieľom vyššej abstrakcie. Knižnica napríklad umožňuje jazykovo-nezávislý typový systém, graf toku riadenia alebo typovanú množinu registrov. LLVM IR neposkytuje vlastnosti vysokoúrovňových programovacích jazykov ako sú triedy, dedičnosť alebo spracovanie výnimiek. Cieľom LLVM IR nie je byť univerzálny IR ale je naopak doplnkom vysokoúrovňových virtuálnych strojov ako sú Smalltalk VM alebo Self VM. Benefity LLVM IR sa plne využijú pri reprezentácii staticky prekladaných jazykov ako je C a C++. [2]

[2]Capstone projekt. http://www.capstone-engine.org/

Základný koncept syntaktickej reprezentácie LLVM IR sa skladá z nasledujúcich datových štruktúr (viď [3]):

- **Modul** definuje obsah celého LLVM IR súboru.
- **Funkcia** je koncept syntakticky podobný procedúre v programovacom jazyku C. Telo funkcie je definované sekvenciou základných blokov.
- **Základný blok** predstavuje jeden uzol v grafe toku riadenia (anglicky *Control Flow Graph*, ďalej len *CFG*) v rámci jeho funkcie. Blok reprezentuje sekvenciu inštrukcií s jedným vstupným bodom a jedným výstupným bodom.
- **Inštrukcia** je najnižšia úroveň abstrakcie, ktorá tvorí typicky trojadresný kód s maximálne dvoma vstupmi a jedným výstupom. LLVM IR obsahuje inštrukcie na aritmetické operácie, logické operácie, bitové operácie, zápis a čítanie z pamäte, porovnávanie hodnôt, volanie funkcií a ďalšie.

Príklad 1 zobrazuje základné syntaktické prvky v LLVM module. Tento modul obsahuje jednu definíciu funkcie s globálnym názvom `@sum`. Taktiež definuje jednu globálnu premennú `@GLOBAL_VAR`, ktorej dátový typ je 32-bitový integer. Telo funkcie obsahuje jediný základný blok s označením `entry`. Vyjadrovacia schopnosť inštrukcií v základnom bloku je veľmi podobná assemblerovským inštrukciám, avšak navyše explicitne definuje dátové typy.

```
1 @GLOBAL_VAR = external global i32
2
3 define i32 @sum(i32 %a, i32 %b) {
4 entry:
5     %add = add i32 %b, %a
6     ret i32 %add
7 }
```

**Príklad 1.** Príklad syntaktických prvkov LLVM IR.

## Jadro RetDec

Navrhované rozšírenie je súčasťou jadra spätného prekladača, preto nasledujúca podsekcia priblíži princíp jeho fungovania. Jadro spätného prekladača v prvej fáze mapuje assemblerovské inštrukcie získané spätným prekladom na LLVM IR. Cieľom mapovania nie je preložiť tieto inštrukcie s plným sémantickým významom. Zámerom prekladača je vygenerovať jednoducho pochopiteľný výstup v C/C++. Takýto výstup je následne efektívne analyzovateľný ľuďmi. Preto sémantické prekladanie inštrukcií do LLVM IR prebieha pre konkrétne inštrukcie v štyroch módoch:

1. **Plný preklad**: Inštrukcie, ktorých plný sémantický význam sa dá vyjadriť jednoduchou

sekvenciou LLVM IR.Ide tipycky o aritmetické inštrukcie a inštrukcie na transformáciu dát).

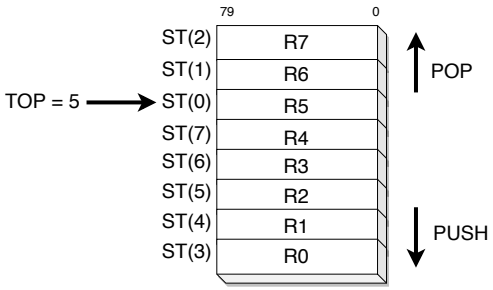
2. **Pseudo assembler funkcie**: Niektoré nízkoúrovňové inštrukcie nie je možné reprezentovať pomocou LLVM IR. Napríklad inštrukcia `FWAIT` sleduje prebiehajúce floating-point výnimky. Tento typ inštrukcií je reprezentovaný samovysvetľujúcou pseudo funkciou (napríklad `@__asm_fwait()`).
3. **Čiastočný preklad**: Inštrukcia je opäť reprezentovaná volaním pseudo funkcie. Navyše je ale explicitne poskytnutá informácia o riadení toku dát. Napríklad inštrukciu `FXSAVE [addr]`, ktorá uloží stav FPU, MMX, XMM jednotiek a ich registrov do 512 bajtov v pamäti na adresu `addr`. Preklad bude vyzerať ako volanie pseudo funkcie a následné uloženie 512 bajtov.
4. **Ignorovanie inštrukcie**: Inštrukcia `NOP` je typ inštrukcie, ktorej výskyt vo vysokoúrovňovom výstupe jazyka C je už nepotrebný a prekladač takéto inštrukcie ignoruje.

Po preklade inštrukcií je táto reprezentácia transformovaná množstvom optimalizačných priechodov, ktorých cieľom je zvýšiť abstrakciu a priblížiť IR výstupnému vysokoúrovňovému jazyku. V tejto časti spätného prekladu prebieha navrhované optimalizačné rozšírenie.

## 3. Preklad inštrukčnej sady FPU

Táto sekcia popisuje inštrukčnú sadu hardvérovej jednotky FPU (anglicky Floating Point Unit, viď [4]), ktorá slúži na výpočty pre čísla s plávajúcou desatinnou čiarkou (anglicky floating point, ďalej len FP). Sekcia ďalej vysvetľuje problémy vznikajúce pri mapovaní FPU inštrukcií do LLVM IR v spätnom prekladači RetDec.

FPU má vlastnú sadu inštrukcií a registrov. Obsahuje vlastné registre pre špeciálne účely ako sú riadiaci register alebo stavový register. Ďalej má osem datových registrov, ktoré sa nazývajú `R0` až `R7`. Tieto registre tvoria dátovú štruktúru zásobník a prístup k ním je podriadený princípom tejto dátovej štruktúry. Zásobník registrov zobrazuje Obrázok 2. Hardvérovým registrom je vždy relatívne priradené označenie `ST0` až `ST7`, kde `ST0` odkazuje na register, ktorý je aktuálne na vrchole zásobníka. Hodnota `TOP` určuje, ktorý hardvérový register je aktuálne vrcholom zásobníka. Zmena vrcholu zásobníka prebieha pomocou operácií *push* a *pop*. Inštrukcia *push* (alternatívne *load*) dekrementuje `TOP` a presunie obsah operandu inštrukcie do registru `ST0`. Inštrukcia *pop* (alternatívne *store*) presunie obsah registra `ST0`




**Obrázok 2.** Abstrakcia hardvérových dátových registrov FPU pomocou dátovej štruktúry zásobník.

do operandu inštruckie a inkrementuje `TOP`. Hodnota `TOP` je reprezentovaná 3-bitovou hodnotou v riadiacom registri a pri pretečení alebo podtečení vždy ukazuje na validný register. [4]

### Mapovanie FP inštrukcií do LLVM IR

Relatívne indexovanie dátových registrov jednotky FPU vedie k problémom pri mapovaní FP inštrukcií na LLVM IR. Jednoduché mapovanie pomocou šablón pre jednotlivé inštrukcie nie je možné. Mapovanie vyžaduje pokročilejšiu analýzu. Uvažujme Príklad 2, na ktorom je časť možného assemblerovského kódu, ktorý manipuluje s FPU zásobníkom.

```
1 FADD ST0, ST1
2 FLD1
3 FADD ST0, ST1
```

**Príklad 2.** Príklad manipulácie s FPU zásobníkom v jazyku assembler.

Prvá inštrukcia manipuluje s operandami `ST0` a `ST1`, ktoré odkazujú na konkrétne hardvérové registre. Druhá inštrukcia `FLD1` dekrementuje vrchol zásobníka a uloží konštantu na nový vrchol zásobníka. V dôsledku dekrementácie už nebudú označenia `ST0` a `ST1` odkazovať na rovnaké hardvérové registre ako v prvej inštrukcii. Tretia inštrukcia je identická s prvou, ale jej operandy sú v skutočnosti už iné hardvérové registre. Toto chovanie vedie k problému, pretože bez kontextu získaného z hodnoty `TOP` je mapovanie chybné. Z tohto dôvodu nemôže LLVM IR reprezentovať Príklad 2 sekvenciou kódu zobrazenou v Príklade 3 (uvažujem, že `@st0` a `@st1` sú globálne premenné reprezentujúce dátové registre `ST0` a `ST1`).

```
1 %op0 = load x86_fp80, x86_fp80* @st0
2 %op1 = load x86_fp80, x86_fp80* @st1
3 %res = fadd x86_fp80 %op0, %op1
4 store x86_fp80 %res, x86_fp80* @st0
```

**Príklad 3.** Chybná reprezentácia FPU registrov pomocou LLVM IR.

Analýza preto mapuje inštrukcie `load` a `store` na volanie pseudo inštrukcií, ktoré majú ako parameter aktuálnu hodnotu vrcholu zásobníka. Takéto

mapovanie predpokladá, že tieto pseudo funkcie budú v rámci ďalších optimalizácií nahradené inštrukciami zo správnymi operandami. Príklad 4 zobrazuje výsledok mapovania assembler kódu z Príkladu 2. Premenná `@fpu_stat_TOP` je globálna premenná, ktorá obsahuje aktuálny stav vrcholu zásobníka.

```
1 %0 = load i3, i3* @fpu_stat_TOP
2 %1 = sub i3 %0, 1
3 %op0 = call x86_fp80 @_loadFP(i3 %0)
4 %op1 = call x86_fp80 @_loadFP(i3 %1)
5 %res = fadd x86_fp80 %op0, %op1
6 call void @_storeFP(i3 %0, x86_fp80 %res)
```

**Príklad 4.** Správna reprezentácia FPU registrov pomocou LLVM IR.

## 4. Navrhované rozšírenie

Táto sekcia popisuje navrhovanú optimalizáciu, ktorá mapuje pseudo funkcie pre prístup k FPU zásobníku na inštrukcie s korektnými operandami.

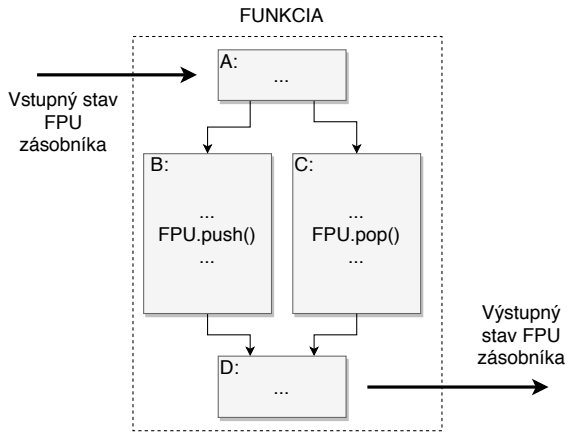
Rekonštrukcia vrcholu zásobníka prebieha vždy v rámci jednej funkcie. Na začiatku funkcie je zásobník vždy prázdny. Tento predpoklad vychádza z analýzy štandardov volania funkcií pre jednotlivé konvencie, architektúry a operačné systémy (viď [5]). Naprieč všetkými štandardami platí, že pri volaní funkcie je vždy zásobník prázdny, a to aj v prípade, že má funkcia parametre dátového typu FP. Takéto parametre sa predávajú ako ukazatele do pamäte, kde je FP hodnota uložená. Na konci volania funkcie je zásobník buď prázdny, alebo obsahuje jednu hodnotu, a to v prípade, že ide o funkciu s návratovou hodnotou typu FP. Avšak návratová hodnota typu FP sa ukladá na zásobník len v prípade, že ide o 32-bitovú architektúru. V prípade 16-bitovej architektúry sa návratová hodnota predáva ako referencia do pamäte a v prípade 64-bitovej architektúry sa návratová hodnota predáva cez registre jednotky SSE. Tabuľka 1 sumarizuje stav FPU zásobníka pre jednotlivé konvencie volania funkcií architektúry x86.

So znalosťou hodnoty vrcholu zásobníku na začiatku funkcie je možné sekvenčne prejsť celú funkciu a sledovať zmeny na zásobníku. Aktuálne volanie pseudo funkcií `load` a `store` je možné nahradiť za inštrukcie s konkrétnymi registrami. Avšak takýto postup ignoruje závislosť na CFG medzi jednotlivými základnými blokmi v rámci funkcie. Obrázok 3 reprezentuje závislosť vrcholu zásobníku na CFG. V bloku *B* sa dekrementuje zásobník a naopak v bloku *C* je zásobník inkrementovaný. Avšak tieto dva bloky tvoria alternatívne vetvy CFG, a preto bude vždy vykonaná len jedna z nich. Analýza neprihliadajúca

| Architektúra | Konvencia volania | Register |
|---|---|---|
| 16 bit | cdecl<br>pascal<br>fastcall | AX |
| | watcom | Nejednoznačné. |
| 32 bit | cdecl<br>stdcall<br>pascal<br>fastcall<br>thiscall | ST(0) |
| | watcom | Nejednoznačné. |
| 64 bit | Windows | SSE registre |
| | Linux, BSD, Mac OS | |

**Tabuľka 1.** Konvencie volania funkcií pre rodinu architektúr x86 a sumarizácia využitia FPU registrov pre predávanie návratovej hodnoty s dátovým typom FP.

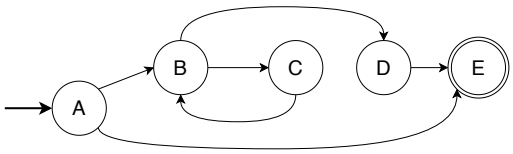
na tento fakt by teda nemohla vyhodnocovať vrchol zásobníka správne.




**Obrázok 3.** Ilustrácia nesprávnej manipulácie s FPU zásobníkom.

### Analýza CFG funkcie

Navrhované riešenie analýzy CFG využíva základné pojmy a vedomosti numerickej lineárnej algebry a maticového počtu (bližšie informácie viď [6]). Navrhovaný algoritmus bude ilustrovaný na príklade CFG z Obrázka 4.




**Obrázok 4.** Príklad CFG pre funkciu.

Rozšírená analýza sa pokúsi získať stav vrcholu zásobníka na začiatku každého základného bloku. S touto znalosťou je ďalej možné prejsť jednotlivo každý základný blok a nahradiť pseudo funkcie za inštrukcie s konkrétnymi registrami. Na získanie hodnoty vrcholu zásobníka na začiatku každého

základného bloku sa podľa CFG zostaví sústava Rovníc (1), kde každý blok definuje dve neznáme: hodnota na začiatku (napríklad $A_{in}$) a konci bloku (paríklad $A_{out}$).

Každý základný blok je samostatne zanalyzovaný. Nezávisle na stave zásobníka pri vstupe do bloku je možné vyhodnotiť relatívny rozdiel vstupného a výstupného stavu vrcholu zásobníka. Napríklad pre blok $A$ je táto hodnota označená ako $A_\Delta$ (obdobná konvencia označenia je použitá aj pre ostatné bloky). Výsledkom tejto analýzy je sústava Rovníc (2).

$$\begin{aligned} B_{in} - A_{out} &= 0 \\ C_{in} - B_{out} &= 0 \\ B_{in} - C_{out} &= 0 \\ D_{in} - B_{out} &= 0 \\ E_{in} - D_{out} &= 0 \\ E_{in} - A_{out} &= 0 \end{aligned} \qquad (1)$$

$$\begin{aligned} A_{out} - A_{in} &= A_\Delta \\ B_{out} - B_{in} &= B_\Delta \\ C_{out} - C_{in} &= C_\Delta \\ D_{out} - D_{in} &= D_\Delta \\ E_{out} - E_{in} &= E_\Delta \end{aligned} \qquad (2)$$

Algoritmus získa Rovnice (3) na základe znalosti konvencie volania funkcie, ktorú dokáže spätný prekladač detekovať. V našom vzorovom príklade je $A_{in}$ hodnota zásobníka na začiatku funkcie a $C_{out}$ hodnota na konci funkcie. Ukončovacích blokov funkcie môže byť viacero, ale všetky musia mať rovnakú hodnotu. Zásobník je buď prázdny alebo obsahuje práve jednu hodnotu (v prípade, že návratová hodnota funkcie je typu FP).

$$A_{in} = 0 \qquad C_{out} = 0 \qquad (3)$$

CFG môže obsahovať veľmi odlišné množstvo hrán. Výsledný systém bude preto mať typicky viac rovníc ako neznámych. Takýto systém sústavy lineárnych rovníc sa nazýva *preurčený* (anglicky *overdetermined*). Preurčený systém nemá vo väčšine prípadov riešenie. Avšak, v tomto prípade je veľká pravdepodobnosť, že existuje práve jedno riešenie. Celý spätný prekladač predpokladá, že analyzovaný binárny súbor je vytvorený štandardným prekladačom.

Takto získanú sústava lineárnych rovníc je možné reprezentovať pomocou matíc ako Rovnicu (4), kde $\mathbf{A}$ je matica koeficientov systému a $\bar{\mathbf{x}}$ je vektor neznámych.

$$\mathbf{A}\,\bar{\mathbf{x}} = \bar{\mathbf{b}} \qquad (4)$$

Matica $(\mathbf{A}|\bar{\mathbf{b}})$ sa nazýva *rozšírená matica sústavy*. Analýza vyhodnotí *hodnosť* matice $\mathbf{A}$ a hodnosť rozšírenej matice $(\mathbf{A}|\bar{\mathbf{b}})$. Systém má práve jedno riešenie, ak sú tieto hodnosti rovné. Na riešenie preurčených lineárnych systémov sa najčastejšie používa aproximačná *metóda najmenších štvorcov* (anglicky *least squares*). Aproximácia vnáša do riešenia chybu avšak naša aplikácia nevyžaduje veľkú presnosť. Výsledky systému predstavujú indexy na registre a teda budú zaokrúhlené na celé čísla. Riešenie sústavy metódou najmenších štvorcov umožňujú rôzne numerické metódy. Tieto metódy *dekomponujú* maticu $\mathbf{A}$ na viacero matíc koeficientov, ktoré sa dajú riešiť efektívnejšie. Nepresnosť vnesená strojovým zaokruhľovaním sa znižuje lebo koeficienty sú substituované za skutočné hodnoty až spätne. Táto práca zhodnotila z hľadiska presnosti a efektívnosti nasledujúce tri metódy (zdroje porovnania [7, 8]):

- **Cholesky** dekompozícia je všeobecne najrýchlejšia ale aj najmenej presná. Malé odchýlky na vstupe vnášajú veľkú nepresnosť do výsledku.
- **QR** dekompozícia je numericky stabilná a teda aj presnejšia ale je náročnejšia na výpočet.
- **SVD** dekompozícia poskytuje všeobecne najpresnejšie riešenie a to aj pre väčšie nepresnosti na vstupe. Na druhej strane je značne náročnejšia na výpočet (približne 10 násobne).

Vypočítaný vektor $\bar{\mathbf{x}}$ obsahuje hodnoty FPU zásobníka na začiatku a konci každého bloku v rámci aktuálne analyzovanej funkcie. S touto vedomosťou optimalizácie prejde zvlášť každý základný blok, pričom už vie hodnotu zásobníku na jeho začiatku. Optimalizácia sleduje (teraz už skutočnú) hodnotu zásobníka sekvenčným prechodom cez blok a nahradí volania pseudo funkcií FPU registrami na ktoré aktuálne odkazuje vrchol zásobníka.

Navrhované rozšírenie bolo implementované v C++ a pre prácu s maticami bola použitá knižnica Eigen[3].

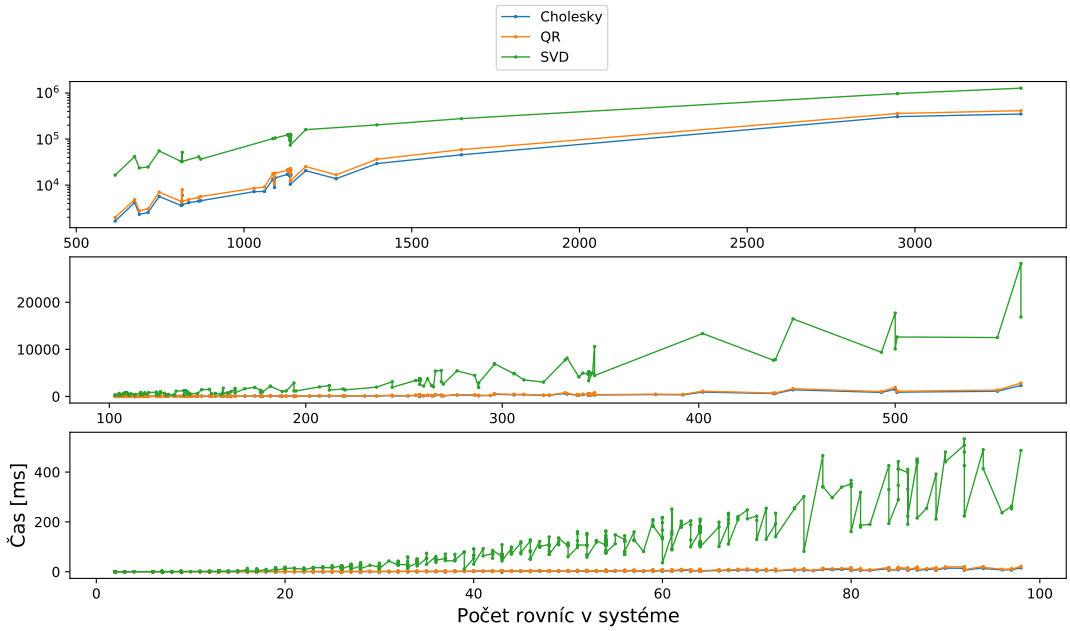
## 5. Výsledky experimentovania

Naimplementovaná optimalizácia bola otestovaná sadou integračných testov. Testovacie prostredie tvoril nástroj[4] vytvorený projektom RetDec. Tento nástroj umožňuje vyhodnocovať návratovú hodnotu spätného prekladu, analýzu vygenerovaného výstupu (mená funkcií, typov, zhoda na reťazce a podobne) a opätovný preklad a spustenie zrekonštruovaného programu (porovnávanie štandardného výstupu pôvodného a zrekonštruovaného programu).

Testovaciu sadu tvorilo 822 spustiteľných binárnych súborov, ktoré boli vytvorené rôznymi prekladačmi pre architektúru x86 (z toho 86%

[3] Free Software projekt Eigen: http://eigen.tuxfamily.org

[4] RetDec testovací nástroj: https://retdec-regression-tests-framework.readthedocs.io/en/latest/




**Obrázok 5.** Porovnanie výpočetnej náročnosti jednotlivých metód dekompozície lineárneho systému vzhľadom na jeho veľkosť.

| Počet rovníc | Cholesky | QR | SVD |
|---|---|---|---|
| $< 100$ | 1.29 ms | 1.88 ms | 43.89 ms |
| $\langle 100\,;\,600 \rangle$ | 264.34 ms | 345.11 ms | 3 948.95 ms |
| $600 >$ | 18 741.14 ms | 23 332.06 ms | 125 831.64 ms |

**Tabuľka 2.** Priemerný čas vyriešenia systému pomocou rôznych dekompozičných metód.

binárnych súborov je určených pre 64-bitovú architektúru, 12% pre 32-bitovú architektúru a zvyšok pre 16-bitovú architektúru). Tieto spustiteľné súbory sú súčasťou databázy testovacieho nástroja projektu RetDec.

### Optimalizačné experimentovanie

V Sekcii 4 boli navrhnuté tri metódy riešenia preurčeného systému (Cholesky, QR, SVD). Algebraická knižnica Eigen podporuje všetky tri. Každá s týchto metód bola experimentálne naimplementovaná v novom rozšírení. V rámci troch experimentov boli jednotlivé metódy aplikované na testovaciu sadu a proces dekompozície bol meraný.

Testovacia sada obsahovala viac ako 4 000 funkcií, ktoré manipulujú s FPU. Výsledky merania pre jednotlivé metódy zobrazuje Obrázok 5. Cholesky a QR dekompozícia trvali zanedbateľne podobne. Na druhej strane, SVD dekompozícia trvala (a to najmä pre veľké systémy) neprijateľne dlho oproti ostatným metódam. Tabuľka 2 zobrazuje priemerné doby výpočtu pre jednotlivé metódy. Pre lepšiu analýzu sú výsledky merania rozdelené do troch skupín (podľa množstva rovníc v systéme).

### Vyhodnotenie úspešnosti optimalizácie

Druhá časť experimentovania zhodnocuje úspešnosť optimalizácie. V rámci danej testovacej sady bolo celkovo analyzovaných 4 158 funkcií, ktoré pracovali s inštrukčnou sadou FPU. Išlo o funkcie vytvorené reálnymi prekladačmi. Preto sa očakávalo, že všetky systémy lineárnych rovníc z nich vytvorené budú mať riešenie (matica sústavy a rozšírená matica sústavy majú rovnakú hodnosť). Optimalizácia našla riešenie sústavy rovníc pre všetky testované funkcie. Pre tieto funkcie nahradila pseudo funkcie za konkrétne FPU registre zo 100 % úspešnosťou. Úspešnosť nahradenia vyhodnocoval testovací nástroj projektu RetDec. Tento nástroj detekuje deklaráciu volaných funkcií a v prípade neúspechu optimalizácie by teda detekoval nesubstituované pseudo funkcie.

## 6. Záver

Na základe výsledkov experimentovania vidím možnú potrebu zamerať sa na zefektívnenie optimalizácie pre funkcie s veľmi veľkým množstvom základných blokov. Pre funkcie, ktoré tvoria lineárne systémy s viac ako 1000 rovnicami je optimalizácia nezanedbateľne zaťažujúca na celú dekompiláciu. Pre moju testovaciu sadu u takto veľkých systémov trvala optimalizácia v priemere viac ako 23 sekúnd. Údaj je samozrejme len relatívny vzhľadom na konkrétne CPU, na ktorom experiment bežal.

Ďalšie potenciálne možnosti rozšírenia spätného prekladača RetDec vidím v oblasti jednotky SSE, ktorá v 64-bitových architektúrach nahradila jednotku FPU.

V rámci tejto jednotky by som chcel preštudovať a zistiť možné optimalizácie pri spätnom preklade tejto pokročilej inštrukčnej sady.

## Poďakovanie

Chcel by som poďakovať vedúcemu mojej práce Zbyňkovi Křivkovi, a taktiež celému Avast tímu, ktorý mi poskytol profesionálnu pomoc. Menovite by som chcel poďakovať Petrovi Matulovi, Jakubovi Křoustkovi, a ostatným za ich konzultácie aa odnotné komentáre.

## Literatúra

[1] Eldad Eilam. *Reversing secrets of reverse engineering*. Wiley, Indianapolis, 2005.

[2] Chris Lattner and Vikram Adve. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO'04)*, Palo Alto, California, Mar 2004.

[3] Bruno Cardoso Lopes and Rafael Auler. *Getting Started with LLVM Core Libraries*. Packt Publishing, 2014.

[4] Kip R Irvine. *Assembly language for x86 processors*. Prentice Hall, Upper Saddle River, 6th ed. edition, 2010.

[5] Agner Fog. *Calling conventions for different C++ compilers and operating systems*. Technical University of Denmark, 2019.

[6] H. Anton. *Elementary Linear Algebra: Applications Version*. Wiley Plus Products. Wiley, 2010.

[7] Pavel Cizek and Lenka Cizkova. Numerical linear algebra. 01 2004.

[8] Do Q Lee. *Numerically efficient methods for solving least squares problems*. The University of Chicago, 2012.

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

# Využití Nix/NixOps pro průběžnou integraci a nasazení software při vývoji

Tomáš Vlk*

```
/nix/store
├─ 5mq2jcn36l...-git-2.20.1/bin/git
├─ 6yaj6n8l92...-glibc-2.27/lib/libc.so.6
└─ z8bn38sz1z...-openssl-1.0.2t/lib/libssl.so.1.0.2
```

**Abstrakt**
Tato práce se zabývá uplatněním funkcionálního balíčkovacího systému Nix a jeho ekosystému (NixOS, NixOps) pro CI/CD při agilním vývoji. Při použití těchto technologií jsou problémy způsobené odlišným prostředím prakticky eliminovány bez nutnosti kontejnerizace. Práce obsahuje popis možností a nedostatků Nix/NixOps a navrhuje obecný postup použití těchto technologií pro jednotlivé fáze agilního vývoje a CI/CD. Díky Nix/NixOps je implementace CI/CD velmi jednoduchá a celý proces je navíc reprodukovatelný. Výstupem práce je sada příkladů demonstrující použití Nix/NixOps v různých projektech, a která je dostupná jako open-source. Díky této sadě mohou vývojáři použít Nix rychle a jednoduše v jakémkoliv projektu, bez nutnosti studia velkého množství materiálů.

**Klíčová slova:** Continuous Integration — Continuous Deployment — Configuration Management — Infrastructure as Code — Nix — NixOS — NixOps

**Přiložené materiály:** nix-examples

*xvlkto00@stud.fit.vutbr.cz, *Fakulta informačních technologií, Vysokého učení technického v Brně*

## 1. Úvod

S agilním vývojem je úzce spjatý proces průběžné integrace a nasazení software (CI/CD). Pro spolehlivé otestování a nasazení aplikace je potřeba eliminovat problémy způsobené odlišným prostředím. V ideálním případě by prostředí, které používá vývojář, mělo být totožné s prostředím CI/CD serveru, na kterém je software testován a s produkčním serverem, na kterém je následně nasazen. Drobné odchylky mezi prostředími mohou způsobit, v lepším případě chybu při zpracovávání CI/CD serverem, ale v horším případě nefunkční nasazenou aplikaci na produkčním serveru.

Problémy spojené s nasazením se často řeší pomocí kontejnerizace aplikací. Software je na CI/CD serveru testován v kontejneru a na produkčním serveru je nasazen ve stejném kontejneru. Díky tomu jsou problémy s prostředím takřka odstraněny. Používání kontejnerů ale není ideální. Ve spoustě případů není potřeba aplikace v systému kompletně izolovat do samostatných kontejnerů. Musí se pak řešit orchestrace, monitorování, sdílení dat mezi kontejnery a v neposlední řadě může být i problém s rychlostí [1].

Ekosystém balíčkovacího systému Nix nabízí deklarativní zápis balíčku, celého systému nebo dokonce celé infrastruktury. To vše s jistotou reprodukovatelnosti. Díky tomu nenastávají problémy s prostředím a proces průběžné integrace a nasazení software je velmi spolehlivý a agilní vývoj rychlejší. Dokonce je i možné použít Nix velmi efektivně na testování nasazení aplikace, ještě před samotným nasazením.

Nevýhodou Nix je, že neexistuje kompletní a jednotný přehled použití pro jednotlivé aplikace různého

druhu. Například webové, mobilní, distribuované atd. Dále, kvůli nutnosti definování každého balíčku pomocí Nix, je nutné mít odlišné příklady pro jednotlivé programovací jazyky a nástroje pro sestavení, případně obecný návod, jak jakýkoliv projekt, používající oficiálně nepodporované technologie, zprovoznit pomocí Nix.

Je možné nalézt několik málo projektů, které demonstrují použití Nix. Například projekt nixtodo[1] nebo TodoMVC[2]. Oba tyto příklady jsou ale pro začátečníka velmi komplikované, protože obsahují pro jeden projekt více než 15 `.nix` souborů. Navíc autoři těchto příkladů mají s Nix velké zkušenosti a mají tendenci používat Nix i pro jiné účely, než pro které byl původně určen.

Cílem této práce je vytvoření obsáhlé sady příkladů demonstrující komplexní použití Nix. Přitom je ale kladen důraz na jednoduchost a jednotnost příkladů. Každý příklad obsahuje stejné `.nix` soubory se stejným významem a je snaha o minimalizaci rozdílů mezi různými technologiemi. Je ale potřeba přistupovat odlišně ke správě závislostí. U některých sestavovacích nástrojů mohou být závislosti projektu dodány jako samostatné balíčky, ale u jiných musí být dodány všechny jako jeden balíček. Díky jednotnému rozhraní je pak možné spustit proces průběžné integrace a nasazení jedním příkazem a pro integraci s klasickými CI/CD řešeními stačí už jenom Nix nainstalovat.

Čtenář se v této práci dozví, jak Nix funguje, jaké má vlastnosti a díky příkladům, které jsou výstupem této práce, bude moci Nix ihned použít ve svých projektech. Příklady pokrývají valnou většinu typických projektů od desktopových aplikací až po aplikace pro mobilní či vestavěné platformy. Díky funkcionálnímu jazyku a nástrojům vytvořeným okolo Nix, není pak problém tyto aplikace škálovat, či komponovat do větších celků.

# 2. Nix a jeho ekosystém

Povědomí o Nix je zatím spíše raritou, a proto jej a technologie, které s ním bezprostředně souvisí, tato kapitola popisuje. Protože je ale vývoj Nix velmi rychlý a jeho možnosti jsou obrovské, obsahuje tato kapitola hlavně zásadní principy a vlastnosti jednotlivých technologií. Bez tohoto úvodu do Nix by nemusely být následující kapitoly plně srozumitelné, pokud ale čtenář Nix zná, může tuto kapitolu přeskočit.

[1] https://github.com/basvandijk/nixtodo
[2] https://github.com/nix-community/todomvc-nix

## 2.1 Standardní balíčkovací systémy

Standardní správce balíčků, jako je APT pro distribuce založené na Debianu, nebo DNF pro distribuce používající RPM formát balíčků, jsou *stavové*. Pokud se balíčky aktualizují, tak se přepisují soubory nainstalovaných balíčků a mění se stav systému. Změna se nemusí povést a špatně se pak řeší rollback, jestliže nebyly staré soubory zazálohovány. Společně s tím nastává i problém atomičnosti, některé balíčky mohou být v určitý čas už aktualizované a některé ještě ne. [2]

Dalším problémem standardních balíčkovacích systémů je nemožnost mít několik verzí stejného programu/knihovny. Může nastat situace, kdy budou dva balíčky závislé na jiné verzi knihovny, přičemž jenom jedna verze této knihovny může být v systému aktivní. Takový problém je ve standardních balíčkovacích systémech neřešitelný a nazývá se jako *dependency hell*. Dokonce samotná verze nemusí stačit a může být vyžadována přímo nějaká varianta (například přeložená s jinou konfigurací). Varianty ale standardní správci balíčků vůbec neidentifikují, rozlišují pouze mezi verzemi. Navíc není jednoduché zreprodukovat sestavení již sestaveného balíčku[3], takže každé sestavení může mít stejné důsledky jako jiná varianta balíčku.

Problém několika verzí a s ním spjatý dependecy hell se snaží řešit správci balíčků Snappy a Flatpak. Fungují tak, že v instalovaném balíčku jsou zabaleny současně i všechny jeho závislosti a instalovaný balíček pak používá jenom svoje závislosti, ne ty nainstalované globálně v systému. Problémem tohoto přístupu je ale velká velikost balíčku. Balíčky nainstalované pomocí Snappy a Flatpak mezi sebou nesdílí závislosti nebo jenom minimálně. Dalším problémem je bezpečnost, když je nalezena bezpečností díra v nějaké závislosti. Pokud závislost není sdílená mezi balíčky, je obtížné opravit nebo aktualizovat tuto závislost u všech balíčků. V neposlední řadě nemusí vždy fungovat správně vizuální integrace se systémem, jako jsou vlastní témata nebo fonty.

Velká výhoda výše zmíněných balíčkovacích systémů je jejich rychlost. Balíčky jsou distribuované už zkompilované a takové správce balíčku lze pak označit jako *binary based*. Naproti tomu existují *source code based* balíčkovací systémy. V takových balíčkovacích systémech probíhá sestavení balíčku u uživatele a až teprve se instaluje do systému. Někdy se balíčky mohou distribuovat primárně v binárním formátu, ale uživatel může zvolit i lokální kompilaci a následnou instalaci. Takový balíčkovací systém lze pak označit jako *hybridní*.

[3] https://reproducible-builds.org

## 2.2 Nix: čistě funkcionální správce balíčků

Správce balíčků Nix je hybridní balíčkovací systém. Pro každý balíček existuje speciální předpis, jak se má sestavit a Nix garantuje reprodukovatelnost sestavení. Součástí sestavení bývá často i kompilace a ta může trvat velmi dlouho. Proto jsou již sestavené balíčky uloženy v cache (na uživatelům dostupných serverech). Z pohledu uživatele se tak Nix jeví jako binary based balíčkovací systém.

Na rozdíl od standardních balíčkovacích systémů nepoužívá Nix standardní uložiště programů `/bin` nebo `/usr/bin`. Všechny balíčky jsou instalovány do *Nix store*, což je jednoduše jenom adresář, ve výchozím nastavení `/nix/store`. V uložišti ukládá tři typy objektů: derivace, výstupy derivací (derivát) a zdrojové soubory. *Derivace* je speciální objekt, který popisuje prostředí, zdrojové soubory a příkazy potřebné k sestavení nějakého balíčku. Vykonáním derivace získáme *výstup derivace*, tedy samotný balíček. Pro sestavení jsou potřeba již zmíněné *zdrojové soubory*, které jsou v `/nix/store` uloženy samostatně. Pojem balíček má v Nix obecnější význam, nemusí se jednat jen o spustitelný program. Balíček je výstup derivace, který může obsahovat cokoliv.

Každý objekt je uložen v `/nix/store` pod unikátním identifikátorem, skládajícího se z hashe a názvu samotného objektu. Používá se SHA-256 hash, který je oříznut na 160 bitů a zakódován pomocí notace Base32 [3]. Pokud se jedná o výstup derivace, je hash získán ze všech vstupů, které byly použity k sestavení (hash derivace). U zdrojových souborů a derivací je hash vytvořen jednoduše z obsahu souboru.

Díky hashi v názvu objektu, který je takřka bezkolizní a tedy unikátní, je možné, aby byl balíček uložen v `/nix/store` v různých verzích a variantách zároveň. Při instalaci nebo aktualizaci se nepřepisují soubory a nemění se závislosti již nainstalovaných balíčků, jako je tomu například u APT nebo RPM, ale jen se přidávají nové. Na rozdíl od klasických správců balíčků, tak Nix nemění stav systému a je tedy *bezstavový*.

Na obrázku 1 je příklad objektů v `/nix/store`. Každý objekt je soubor nebo adresář umístěný přímo v adresáři `/nix/store`. Šipky v obrázku znázorňují závislosti mezi objekty. Například objekt `rr3y0c6zyk...-hello-2.10` je závislý na objektu `6yaj6n8l92...-glibc-2.27`. Konkrétně obsahuje tento binární soubor řetězec `/nix/store/6yaj6n8l92...-glibc-2.27/lib/libc.so.6`. Závislosti jsou v binárních souborech uvedeny explicitně. Dynamický linker pak nevyhledává knihovny například v adresáři `/usr/lib`, ale rovnou je načte [2].

**Obrázek 1.** Objekty v `/nix/store` a závislosti mezi nimi. Pro lepší přehlednost jsou hashe v názvech objektů zkrácené a některé soubory jsou vynechané.

Aby bylo zaručeno, že objekt není závislý na jiných objektech mimo `/nix/store`, používá se k sestavení derivací čisté prostředí a při spuštění programu se používá upravený dynamický linker [4]. Protože názvy závislostí obsahují hash, jsou závislosti vždy explicitně a přesně určeny. Pokud vývojář zapomene specifikovat nějakou závislost, tak s velkou pravděpodobností program nepůjde sestavit nebo nebude fungovat. Nestane se ale to, že by program fungoval kvůli tomu, že tato závislost je v systému ve standardní cestě. To je dobře, protože díky tomu není tato závislost opomenuta. Závislosti jsou textově zapsány ve výstupech derivací a díky hashi se dají v souborech vyhledat. Použití Nix tak velmi pomáhá ke kompletní specifikaci všech závislostí balíčku [3].

### Deklarativní zápis balíčků

Každý standardní balíčkovací systém používá nějaký formát pro popis metadat a závislostí balíčků. Při deklarování závislostí je potřeba přesně specifikovat v jaké verzi a variantě tato závislost má být. Většinou tento formát umožňuje pouze deklarování verze závislosti. Jak bylo zmíněno výše, Nix používá kryptografické hashe, které jednoznačně určují verzi, variantu i způsob sestavení balíčku. Při definování nového balíčku by tedy stačilo deklarovat jeho závislosti ve formě hashů. Tyto hashe ale slouží k identifikaci výstupu, a pokud by nějaký tento hash nebyl v `/nix/store` a nebyl ani v cache, musela by se tato závislost znovu sestavit. Jenže chybí zpětná informace, jaká derivace produkuje daný hash a není tedy možné tuto derivaci sestavit a uložit výstup v `/nix/store`.

V Nix se proto v popisu balíčku závislosti nedeklarují, ale definují se. To znamená, že v případě po-

třeby mohou být vždy znovu sestaveny. Z toho důvodu musejí být součástí definice i tranzitivně definice všech závislostí. K tomuto účelu byl vytvořen stejnojmenný speciální programovací jazyk Nix, který umožňuje právě deklarativně popsat, jak se má balíček sestavit, jaké má závislosti a navíc, umožňuje i vytvářet kompozice balíčků.

Jazyk Nix je silně dynamicky typovaný funkcionální jazyk s podporou *lazy evaluation*. Je velmi jednoduchý a nenabízí tolik možností jako jiné funkcionální jazyky. Jedná se o *DSL (Domain Specific Language)* primárně určený pro deklarativní zápis balíčků. Důvodem pro jeho vytvoření byla hlavně možnost snadno tvořit varianty balíčků a vytvářet grafy závislostí mezi derivacemi.

Jednoduchý příklad definice balíčku je ve výpisu 1. Celý balíček je zadefinován jako funkce. Díky tomu může být při změně parametru balíček z funkce vrácen v jiné variantě. Funkce je *čistá*, nemá přístup k ničemu globálnímu a nemá žádné vedlejší účinky. Právě proto je Nix čistě funkcionální správce balíčků. Pokud je balíček na něčem závislý, musí obsahovat parametr, který tuto závislost zprostředkuje. To umožňuje snadno sestavit balíček s jinou verzí nebo variantou dané závislosti. Oficiální repozitář, obsahující takto definované balíčky dostupné pro Nix, je *Nixpkgs*[4].

```
{ stdenv, fetchurl, someDependency }:

stdenv.mkDerivation rec {
  pname = "example";
  version = "1.0";

  src = fetchurl {
    url =
  "https://example.org/${pname}-${version}";
    sha256 = "0ssi1wpafc...7c9lng89nd";
  };

  buildInputs = [ someDependency ];

  buildPhase = ''
    gcc example.c -o example
  '';
}
```

**Výpis 1.** Jednoduchý příklad definice balíčku. Hash zdrojových souborů byl pro přehlednost zkrácen.

Sestavení derivace probíhá v *sandboxu* – v čistém a reprodukovatelném prostředí. Nejdříve se vytvoří dočasný pracovní adresář. Dále jsou vymazány všechny proměnné prostředí a některé jsou znovu nastaveny na základě popisu derivace (`$out`, `$PATH`, ...). Poté je vykonán sestavovací skript, a nakonec jsou všechny výstupy derivace uloženy v `/nix/store`. U všech souborů uložených v `/nix/store` jsou dokonce znovu nastavena práva přístupu a nastaveny časové značky na hodnotu 1 (00:00:01 1/1/1970 UTC). Samotný sestavovací skript nemá přístup k souborům v systému. Má přístup pouze k `/nix/store`, svému dočasnému pracovnímu adresáři a k upraveným variantám systémových souborů, jako jsou soubory v adresáři `/proc` nebo `/dev`. Spuštěný proces je oddělen od systému a ostatních procesů pomocí *namespaces*. Má například vlastní namespace pro číslo procesu, souborový systém nebo síťová zařízení. Kvůli determinismu nemá skript vůbec žádný přístup k internetu. Navíc je každé sestavení derivace spuštěné pod jiným speciálním uživatelem, aby se více spuštěných sestavení nemohlo ovlivňovat. [3]

Pokud derivace specifikuje hash svého výstupu ve speciálním atributu `outputHash`, jedná se o *fixed output derivation (FOD)*. Sestavení se provádí stejně jako u normální derivace, jen je hash výstupu již předem znám. Pokud se hash výstupu derivace neshoduje s uvedeným hashem, skončí sestavení neúspěšně. FOD derivace má dokonce i se zapnutým sandboxem přístup k internetu. Reprodukovatelnost není narušena díky referenční transparentnosti celé derivace – je jedno co derivace provádí, její výstup musí odpovídat specifikovanému hashi. Například funkce `fetchurl` použitá ve výpisu 1 je FOD.

## 2.3 NixOS: čistě funkcionální linuxová distribuce

V klasických unix-like operačních systémech většinou není možné mít nainstalovaný program ve více verzích nebo variantách. Jedním z problémů je globální uložiště nainstalovaných programů v adresáři `/bin`. Tento problém se často řeší přidáním čísla verze k názvu programu a vytvořením symbolického odkazu na aktivní verzi. Jenže každá verze programu může vyžadovat jiné verze konfiguračních souborů. Může se jednat o jiný formát nebo jiné možnosti nastavení. Tudíž nastává stejný problém, akorát s globálním uložištěm konfiguračních souborů v adresáři `/etc`.

Hlavním problémem klasických unix-like systémů, je použití stavového správce balíčků. Při použití stavového správce balíčků jsou balíčky a konfigurace instalovány a upravovány pomocí imperativních kroků, které modifikují globální stav systému [5]. To ztěžuje sledování změn, reprodukovatelnost a možnosti provedení rollbacku. Navíc aplikování několika imperativních transformací není atomické.

Existují komplexní řešení, která automatizují tyto imperativní kroky a umožňují spravovat několik systémů zároveň. Patří do kategorie nástrojů pro správu

[4] https://github.com/NixOS/nixpkgs

konfigurace (CMS) a jsou to například nástroje Ansible a Puppet. Tyto nástroje ale neznají přesný stav sytému a jenom provádí zadané příkazy. Například v konfiguraci Ansible může být příkaz pro instalaci nějakého balíčku. Ansible tento příkaz vykoná a v systému bude tento balíček nainstalován. Později může být v konfiguračním souboru Ansible instalace tohoto balíčku odebrána. Po opětovném spuštění Ansible ale není provedena odinstalace na cílovém systému. Stav systému se tak imperativně mění a může se postupem času lišit od požadovaného stavu. Jedná se o tzv. *konvergentní model* [6].

Jak už název napovídá, *NixOS* používá bezstavový správce balíčků Nix. Celý systém je deklarativně popsán pomocí jednoho Nix výrazů a je možné ho reprodukovatelně sestavit. Při změně popisu systému se systém znovu sestaví a je ve stavu čisté instalace. Z pohledu správy konfigurace se tak jedná o *kongruentní model* [6].

V NixOS jsou všechny komponenty systému (včetně jádra, balíčků a konfiguračních souborů) sestaveny pomocí Nix [7]. Celý systém je sestaven na základě vyhodnocení výrazu v hlavním konfiguračním souboru systému `/etc/nixos/configuration.nix`. Příklad takovéhoto souboru je ve výpisu 2. Možnosti konfigurace celého systému jsou definovány v *modulech*. Ty jsou stejně jako hlavní konfigurační soubor napsané v jazyku Nix a jsou importovány (viz řádek 3 ve výpisu 2).

Konfigurační soubory programů jsou po sestavení uloženy v `/nix/store` stejně jako balíčky. Soubory v `/nix/store` jsou neměnné, takže konfigurační soubory nelze poté upravovat. Pokud je potřeba změnit nějakou konfiguraci, musí se upravit konfigurace v jazyku Nix a znovu je nechat sestavit. Díky tomu, má tento systém podobné vlastnosti jako správce balíčků Nix. Je reprodukovatelný, podporuje atomické změny a je možné jednoduše provést rollback. Kvůli použití balíčkovacího systému Nix, avšak NixOS nevyhovuje standardu Filesystem Hierarchy Standard. [5]

## 2.4 NixOps: infrastruktura jako kód založená na Nix

Při přístupu DevOps je potřeba průběžně upravovat infrastrukturu a automatizovaně konfigurovat jednotlivé stroje. Infrastruktura může být provozována ve vlastní síti nebo u cloudových poskytovatelů. V předchozí podsekci byl představen deklarativní a reprodukovatelný systém NixOS. Lze na něj nahlížet nejen jako na linuxovou distribuci, ale i jako na nástroj pro CMS. Pokud je provozovaných NixOS systémů více, které jsou mezi sebou propojené, tak je potřeba je konfigurovat společně.

```
{ config, pkgs, ... }: {

  imports = [ ./hello.nix ];

  fileSystems."/mnt" = {
    fsType = "ext4";
    device = "/dev/sda1";
  };

  services.openssh.enable = true;
  environment.systemPackages = with pkgs; [
    wget vim
  ];
}
```

**Výpis 2.** Příklad konfigurace systému NixOS. V systému je připojené zařízení `/dev/sda1`, povolena služba `openssh` a nainstalovány balíčky `wget` a `vim`.

Další technologií postavenou nad Nix je *NixOps*, která slouží jako nástroj pro IaC (Infrastructure as Code), neboli automatické nasazení infrastruktury. Tento nástroj je úzce spjatý s NixOS a umožňuje deklarativně popsat infrastrukturu jednotlivých strojů a jejich konfigurace. NixOps poté vykoná nutné kroky nebo akce k tomu, aby tento popis byl realizován. Uchovává si stav v jakém stroje jsou a při znovu nasazení provádí jenom ty akce, které jsou potřeba [8].

NixOps podporuje několik cílových prostředí pro nasazení:

- Amazon EC2
- Digital Ocean
- Google Compute Engine
- Hetzner
- Libvirtd (Qemu)
- Microsoft Azure
- NixOS
- VirtualBox VM

Podobně jako u deklarativního zápisu konfigurace NixOS, může být zápis infrastruktury strukturován do menších celků. K zápisu se používá jazyk Nix, stejně jako u NixOS a popisu balíčků. Příklad deklarativního zápisu infrastruktury je ve výpisu 3.

I když v předchozím textu převažují pozitivní vlastnosti, má Nix i několik nevýhod. Některé z nich se mohou v budoucnu vyřešit, ale některé už z principu návrhu Nix řešitelné nejsou:

- Pro vytvoření balíčku nebo integraci projektu s Nix musí být vytvořen Nix výraz.
- Jazyk Nix je DSL pro balíčky, ale jazyk nepoužívá žádný koncept balíčků (balíčky jsou funkce).
- Balíček jako funkce deklaruje nejen závislosti, ale i jiné parametry, což není rozlišeno.

```
{
  webserver = {
    deployment.targetEnv = "virtualbox";
    services.httpd.enable = true;
    services.httpd.virtualHosts = {
      "example.org" = {
        documentRoot = "/data";
    };
    fileSystems."/data" = {
      fsType = "nfs4";
      device = "fileserver:/";
    };
  };

  fileserver = {
    deployment.targetEnv = "virtualbox";
    services.nfs.server.enable = true;
    services.nfs.server.exports = "...";
  };
}
```

**Výpis 3.** Příklad infrastruktury sestávající se z webového serveru a souborového serveru. Webový server má připojený obsah souborového serveru do adresáře `/data`. Oba dva stroje budou virtualizovány pomocí VirtualBoxu.

- Pokud se změní některá nízkoúrovňová knihovna (například glibc), tak se musí skoro všechno znovu sestavit.
- Uložiště `/nix/store` není CAS (Content adressable storage) pro všechny objekty. Zdrojové soubory lze adresovat na základě jejich obsahu, ale výstupy derivací nikoliv.
- Problémová a zatím koncepčně nevyřešená je manipulace a uložení hesel a ostatních klíčů v konfiguračních souborech NixOS. Je potřeba si dát pozor, aby se klíče a hesla nezkopírovala do `/nix/store` při vyhodnocování výrazu.

# 3. Návrh použití Nix pro průběžnou integraci a nasazení software

V úvodu této práce byly nastíněny některé problémy existujících řešení CI/CD používaných při agilním vývoji. Většinou je zapotřebí použít při průběžné integraci a nasazení kontejnery, které nejsou vždy žádoucí.

Bez kontejnerizace není snadné reprodukovat prostředí CI/CD serveru a reprodukovatelně spustit proces průběžné integrace a nasazení lokálně. Není možné tak spolehlivě ověřit úspěšnost některých fází, ještě předtím, než jsou spuštěny na CI/CD serveru. Podobně je i problém bez kontejnerizace zajistit stejné prostředí použité na produkčním, CI/CD, testovacím nebo jiném serveru. To snižuje důvěru ke správnému nasazení a fungování aplikace. Pokud nastane chyba při nasazení nové verze aplikace, nemusí být vždy jednoduché provést rollback a to z toho důvodu, že se přepisují soubory aplikace nebo systému (používá se stavový správce balíčků). Nakonec není možné jednoduše změnit CI/CD server, protože každé CI/CD řešení, jako je Jenkins nebo CircleCI, má svůj vlastní formát zápisu jednotlivých fází.

Právě výše zmíněné problémy je možné eliminovat pomocí Nix. I když se tato práce zabývá pouze použitím Nix pro jednotlivé fáze agilního vývoje a CI/CD, je možné použít Nix i v jiných metodikách.

## 3.1 Použití Nix pro sestavení

Ve fázi sestavení jsou zásadní závislosti projektu. Podle podpory dané technologie v Nixpkgs je možné dodat k projektu závislosti třemi způsoby:

1. Závislosti jsou v repozitáři Nixpkgs jako balíčky. Například pro Haskell je dostupná většina závislostí.
2. Derivace závislostí je možné vygenerovat. Závislosti jsou zapsané strukturovaně v nějakém souboru a pomocí nástroje `*2nix` se vygenerují derivace závislostí. Příkladem je JavaScript a nástroj `node2nix`.
3. Závislosti se musí získat externím nástrojem a do definice balíčku dodat jako jedna derivace (FOD). Takový přístup se musí použít například u PHP a sestavovacího nástroje composer.

Poslední zmíněný způsob má několik problémů. Za prvé je nutné manuálně specifikovat hash pro výstup derivace závislostí. Za druhé nemusí externí nástroj podporovat reprodukovatelnost a po každém spuštění mohou být závislosti dodány v trochu jiné verzi. V takovém případě se musí vypnout sandbox a ztrácí se většina dobrých vlastností Nix.

## 3.2 Použití Nix pro testování

V rámci sestavení aplikace mohou být provedeny jednotkové testy. Po sestavení mohou přijít na řadu testy, které testují aplikaci jako celek. Tyto testy mohou být mnoha druhů a mohou mít různé potřeby. Jsou to například systémové testy, stress testy atd.

Většina testů by mělo být možné spustit pomocí shell příkazů. Takové testy jde v Nix provést pomocí speciální funkce `runCommand`. Příklad použití této funkce je ve výpisu 4. Může se klidně testovat i pomocí nějakého frameworku. Stačí tento framework uvést jako závislost, aby byl dostupný. Závislosti potřebné pro testování jsou tak oddělené od závislostí aplikace. Příkaz může i spustit virtuální stroj, a testovat tak aplikaci na jiných platformách.

```
{ pkgs ? import <nixpkgs> {} }:
with pkgs; jobs = rec {

  build = import ./default.nix {
    inherit pkgs;
  };

  tests = pkgs.runCommand
    "tests" { buildInputs = [ build ]; } ''
      echo "Hello world" > expected
      hello > given
      diff expected given > $out/result
    '';

  buildRaspberryPi = import ./default.nix {
    pkgs = pkgsCross.raspberryPi;
  };

  dockerImage = dockerTools.buildImage {
    name = "hello";
    tag = "latest";
    contents = [ build ];
    config = { Cmd = [ "/bin/hello" ]; };
  };
}
```

**Výpis 4.** Testování pomocí speciální derivace pro vykonání posloupnosti shell příkazů, příklad sestavení pro RaspberryPi pomocí cross-compilation a příklad vytvoření Docker image s aplikací.

Dále je možné využít možnosti, které nabízí NixOS. Repozitář Nixpkgs obsahuje pomocné funkce pro spuštění několika virtuálních NixOS systémů, které mohou být navíc propojené. Poté je možné například testovat dostupnost služeb na těchto systémech.

V neposlední řadě je možné využít pro účely testování i NixOps. Stačí nastavit jako cílové prostředí VirtualBox. Celá infrastruktura se pak zprovozní lokálně a může se dále ověřovat její korektnost. Případně je možné nasadit vše na kopii produkční infrastruktury. Protože je vše reprodukovatelné (aplikace, prostředí, infrastruktura), je tímto způsobem možné otestovat nasazení aplikace ještě před samotným nasazením. Spolehlivě a reprodukovatelně sestavit celou infrastrukturu, prostředí a balíčky umí jenom NixOps, jiné nástroje to nedokážou.

Mít ale další stejnou produkční infrastrukturu pro testování může být velmi nákladné. Je proto možné použít kompromisní řešení: nasadit část infrastruktury na jeden NixOS systém a zbytek virtualizovat nebo nasadit stejně jako v produkci. Záleží pak na typu aplikace, do jaké míry je možné se odchýlit od produkční infrastruktury pro systémové testy.

## 3.3 Použití Nix pro release

Sestavení může být provedeno v různých variantách. Stačí ve funkci balíčku deklarovat parametr pro změnu varianty, nebo zpětně přepsat nějaký atribut balíčku. Repozitář Nixpkgs dále nabízí mnoho pomocných funkcí, které usnadňují vytváření instalačních souborů, jako je .deb, .rpm, .snap, ... Pro použití aplikace v kontejnerech je možné jednoduše vygenerovat Docker image nebo OCI kontejner obsahující jenom aplikaci s jejími závislostmi a nic jiného. Díky podpoře cross-compilation je také velmi snadné sestavovat aplikaci pro jiné platformy. Příklad je ve výpisu 4.

Podobně jako při testování, je možné spustit jakýkoliv jiný příkaz a vygenerovat ve fázi release jiný výstup. Může se jednat například o generování programové dokumentace.

## 3.4 Použití Nix pro nasazení

Pro nasazení je možné použít správce balíčků Nix, je ale lepší použít NixOS, protože je pak možné s nasazovaným softwarem upravit i konfiguraci systému. Jednoduše se jenom aktivují a nastaví potřebné moduly aplikace. Konfigurace NixOS může být chápána jako analogie Dockerfile, protože z ní lze jedním příkazem vygenerovat celý systém, který bude obsahovat nasazovaný software. Na cílový NixOS pak už jenom stačí dodat tuto konfiguraci a přepnout na ni. Na rozdíl od Dockeru nebo virtualizace nemá toto řešení žádný vliv na výkon.

Pro jednotné prostředí se často používají kontejnery. Používání kontejnerů pro oddělení dat, systémových prostředků atd. není vždy zapotřebí. Naopak je většinou problém se sdílením dat mezi jednotlivými kontejnery a vyjádření jejich závislosti mezi nimi. I když použití NixOS a použití kontejnerů není vzájemně výlučné, nejsou kontejnery při použití NixOS prakticky potřeba. Oddělení a reprodukování prostředí pro aplikace zajistí samotný Nix potažmo NixOS. Pro jejich běh a orchestraci pak není potřeba další software a je možné využít standardní prostředky systému, jako je systemd. Virtualizační vrstva tak může být úplně eliminována a zároveň je zachovaná stejná funkcionalita jako při použití kontejnerů.

Nejkomplexnějším způsobem nasazení pomocí Nix je využít NixOS společně s NixOps. V takovém případě je možné nasadit software, změnit konfiguraci systému a případně změnit i infrastrukturu. Namísto NixOps je možné společně s NixOS samozřejmě použít i jiné nástroje, například Terraform.

# 4. Implementace sady příkladů

Příklady demonstrují použití Nix/NixOps pro různé druhy aplikací: desktopové, mobilní, webové, vícevrstvé, distribuované, ... Součástí je i krátký informativní soubor obsahující požadavky pro použití a přehled dostupných příkladů. Vše je zveřejněno jako open-source pod licencí GPL na autorově účtu na GitHubu[5].

## 4.1 Dostupné technologie v příkladech

Při implementaci příkladů, jsem se snažil pokrýt nejpopulárnější technologie. Příklady budu postupně doplňovat a upravovat tak, aby byly funkční s aktuální verzí Nix balíčků. Zatím příklady pokrývají tyto jazyky a sestavovací nástroje:

- C – Autotools
- Go – Go modules
- Haskell – Cabal
- Java – Maven, Ant, Gradle
- JavaScript – NPM
- PHP – Composer

Pro každý programovací jazyk může být používáno více sestavovacích nástrojů, které mohou nebo nemusí i fungovat jako nástroj pro správu závislostí. Například pro Javu je Ant pouze sestavovací nástroj bez správy závislostí a Ivy je pouze nástroj pro správu závislostí. Oproti tomu například Maven je sestavovací nástroj a zároveň i nástroj pro správu závislostí.

Právě reprodukovatelné získání závislostí je pro integraci projektu s Nix zásadní. Existují tři způsoby získání závislostí pomocí Nix, jak bylo popsáno výše v sekci 3.1. Přehled dostupných způsobů integrace v sadě příkladů, pro jednotlivé nástroje pro správu závislostí, je v tabulce 1.

| | Nixpkgs | `*2nix` | FOD |
|---|---|---|---|
| Go modules | | | ✓ |
| Cabal | ✓ | | |
| Maven | | | ✓ |
| Gradle | | | ✓ |
| NPM | | ✓ | ✓ |
| Composer | | | ✓ |

**Tabulka 1.** Dostupné příklady úrovně integrace s Nix pro jednotlivé nástroje pro správu závislostí.

Každý příklad je vytvořen tak, aby byl plně samostatný. Mnoho kódu se tak opakuje, ale adresář s projektem díky tomu není závislý na jiných souborech zvenčí. Kromě zdrojových souborů, obsahuje každý příklad pět až devět `.nix` souborů pro integraci s Nix a CI/CD:

- `nixpkgs.nix` – zafixování verze a konfigurace Nixpkgs. V tomto souboru je také možné zadefinovat vlastní balíčky.
- `app.nix` – popis sestavení balíčku jako funkce. Tento soubor je nezávislý na použitém repozitáři balíčků a může být snadno integrován do hierarchie Nixpkgs nebo do vlastního repozitáře balíčků.
- `default.nix` – volání funkce balíčku ze souboru `app.nix` a dodání závislostí z `nixpkgs.nix`.
- `shell.nix` – obaluje balíček v souboru `default.nix` pro vývoj. Mění atributy výsledného balíčku, jako je například atribut `src` a přidává nástroje potřebné pouze pro vývoj.
- `ci.nix` – sada dostupných akcí pro CI/CD.
- `module.nix` – NixOS modul aplikace, který může být jednoduše integrován do stromu NixOS modulů v repozitáři Nixpkgs, nebo do vlastního repozitáře modulů.
- `cd.nix` – logický popis infrastruktury.
- `cd-*.nix` – fyzický popis infrastruktury. Například soubor `cd-vbox.nix` může popisovat nasazení do virtuálních strojů pomocí VirtualBoxu a soubor `cd-cloud.nix` může popisovat způsob nasazení k nějakému z cloudových poskytovatelů.

Jednotlivé derivace (akce) lze komponovat do sebe a tvořit mezi nimi závislosti. Jedna z vlastností Nix je, že derivace, které na sobě nejsou závislé, mohou být vykonány paralelně. To umožňuje elegantní definování, které akce mohou být vykonány paralelně, a které musí být vykonány sekvenčně. Například lze vykonat různé testy nebo vytvořit různé instalační soubory zároveň, ale proces nasazení může být vykonán až po úspěšném vykonání všech testů. Příklad takové definice je ve výpisu 5. V samotném CI/CD systému se už pak vůbec nemusí specifikovat fáze integrace a nasazení a spustí se už jenom jedním příkazem akce `pipeline`.

## 4.2 Zhodnocení použití Nix pro CI/CD

V závěru sekce 2 byly shrnuty nevýhody Nix. Jednou z nich je i nutnost, pro každý projekt nebo balíček, vytvořit Nix výraz. Právě tuto nevýhodu řeší předkládaná sada příkladů, která nabízí obecný postup vytvoření balíčku pro jakýkoliv projekt a zároveň demonstruje komplexní použití Nix pro vybrané technologie. Pro vývojáře by neměl být problém najít v příkladech svou doménu zájmu a příklad snadno modifikovat a adaptovat na svůj projekt. Ne u všech technologií je ale zatím možné dodat závislosti pomocí Nix jednotlivě a musí být dodány jako jeden balíček.

[5] https://github.com/vlktomas/nix-examples

```
jobs = rec {
  ...
  # toto může být vykonáno paralelně
  long-tests = [
    stressTests platformsTests
  ];

  # toto jedině sekvenčně
  pipeline = mkPipeline [
    build tests release deploy
  ];

  # vytvoření závislostí mezi fázemi
  mkPipeline = phases: (
    foldl mkDependency null phases
  );
  mkDependency = prev: next:
    next.overrideAttrs
      (oldAttrs: { prev = prev; });
}
```

**Výpis 5.** Ukázka definice CI/CD pipeline a paralelního a sekvenčního vykonávání v `ci.nix`.

Při využití Nix, jako v příkladech, je možné reprodukovatelně spustit proces CI/CD lokálně bez potřeby kontejnerizace a otestovat tak například úspěšnost některých fází. Pokud se navíc použije jiný fyzický popis infrastruktury, je možné otestovat úspěšnost nasazení aplikace, ještě předtím, než bude nasazení provedeno CI/CD serverem na produkční infrastruktuře.

Díky NixOS je možné zajistit stejné prostředí pro testování aplikace a pro následné nasazení aplikace a kompletně tak odstranit problémy způsobené odlišným prostředím. Každé nasazení je aplikace je navíc nedestruktivní operace. Nové i staré verze software, knihoven a konfigurace jsou na cílovém stroji uloženy po nasazení v `/nix/store`. Případný rollback je tak jenom o aktivaci jiné verze aplikace.

Proces CI/CD není závislý na CI/CD řešení a je možné jednoduše použít jiné řešení. Stačí mít dostupný Nix a spustit jeden příkaz. Na druhou stranu, není integrace Nix s klasickými CI/CD řešeními bezproblémová. Je potřeba řešit uložiště `/nix/store`, způsob interpretace výsledků a zpřístupnění výstupů vytvořených pomocí Nix v CI/CD, jako jsou například instalační soubory nebo vygenerovaná dokumentace.

## 5. Závěr

V této práci byly popsány principy balíčkovacího systému Nix a jeho ekosystému a byly představeny silné a slabé stránky těchto technologií. Dále pak bylo navrženo použití Nix v jednotlivých fázích CI/CD při agilním vývoji.

Pomocí Nix je možné se vyvarovat problémům způsobeným odlišným prostředím a vytvořit tak mnohem spolehlivější proces průběžné integrace a nasazení. Výstupem práce je sada příkladů demonstrující použití a možnosti Nix, díky kterým může vývojář jednoduše použít Nix ve svých projektech. Tato sada příkladů je zveřejněna jako open-source na adrese https://github.com/vlktomas/nix-examples.

Zajímavým rozšířením by bylo vytvoření obsáhlé knihovny obsahující nejrůznější metody testování. Dále pak i rozšíření předkládané sady příkladů o další programovací jazyky, respektive sestavovací nástroje.

Použití Nix není jednoduché a je potřeba nastudovat velmi mnoho materiálů. Nicméně výhody, které Nix nabízí, nesporně převažují toto počáteční úsilí. Velká část práce bude potřeba ještě v oficiální podpoře sestavovacích nástrojů, přidávání chybějících balíčků a dokumentaci. Jinak je ale velmi pravděpodobné, že se Nix nebo podobné řešení stane postupem času standardem jak na serverech, tak i na desktopových stanicích.

## Poděkování

Chtěl bych poděkovat panu RNDr. Marku Rychlému, Ph.D. za vedení a cenné rady při vytváření této práce. Dále bych chtěl poděkovat své přítelkyni a rodině za podporu a trpělivost.

## Literatura

[1] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio. An updated performance comparison of virtual machines and linux containers. In *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 171–172, March 2015.

[2] Eelco Dolstra, Merijn de Jonge, and Eelco Visser. Nix: A safe and policy-free system for software deployment. In *LISA*, 2004.

[3] NixOS organization. Nix package manager guide. https://nixos.org/nix/manual/.

[4] Eelco Dolstra. *The purely functional software deployment model*. PhD thesis, Utrecht University, Utrecht, 2006.

[5] Eelco Dolstra, Andres Löh, and Nicolas Pierron. Nixos: A purely functional linux distribution. *J. Funct. Program.*, 20:577–615, 2010.

[6] Eelco Dolstra. Purely functional configuration management with nix and nixos, Jun 2014. https://infoq.com/articles/configuration-management-with-nix.

[7] NixOS organization. Nixos manual. https://nixos.org/nixos/manual/.

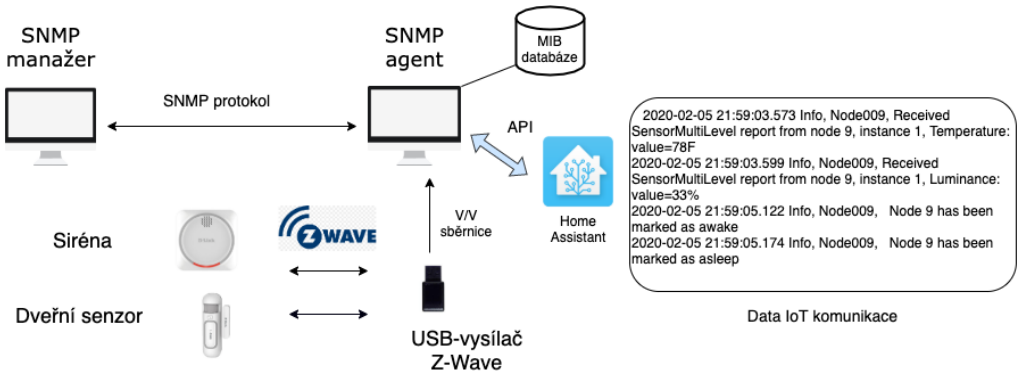
[8] NixOS organization. Nixops user's guide. https://nixos.org/nixops/manual/.

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

# Monitorování IoT sítě pomocí SNMP

Kateryna Polishchuk*




**Abstrakt**
Tato práce řeší implementaci SNMP agenta pro monitorování IoT zařízení, komunikujících přes radiový signál Z-Wave. Podobná zařízení nepodporují protokol SNMP, proto podstatou agenta je shromažďovat informaci o zařízeních z IoT komunikace a ukládat je do proměnných v databázi MIB. Postup implementace agenta bude demonstrován na zařízeních dveřní senzor a signalizační zařízení ze sady inteligentní domácí bezpečnosti D-Link. Zařízení komunikují s USB-vysílačem Z-Wave, který je přípojen k počítací, a posílají o sobě události, jako například teplota, stav, typ senzoru atd. Pro vytvoření vlastního agenta SNMP byl použit nástroj Net-SNMP. Data o zařízeních jsem zpracovávala z nástroje Home Assistant, vyhodnocovala jsem události o zařízeních a ukládala informaci do databáze MIB. Vytvořené řešení poskytuje možnost kdykoliv zjistit aktuální stav zařízení pomocí standardních SNMP manažerů. Výsledná práce může pomoct organizacím používajícím IoT zařízení předejít nežádoucímu ohrožení IoT sítě. Výsledkem této práce je rozšíření Z-Wave komunikace o systém SNMP.

**Klíčová slova:** SNMP — IoT zařízení — Monitorování sítě

**Přiložené materiály:** Kód ke stažení

* xpolis03@fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

## 1. Úvod

**[Motivace]** Téma Internet věcí (IoT) se stává každý rok populárnější. IoT zařízení slouží pro automatizaci a bezpečnost domácnosti, zejména pro řízení a správu obytných a komerčních budov. Internet věcí umožňuje zařízením, aby byly vzdáleně kontrolovány pomocí existující infrastruktury. Nicméně v dnešní době vznikají problémy s bezpečnosti IoT zařízení. Možnými hrozbami jsou například útoky na rozbočovač, zapnutí či vypnutí IoT zařízení, zničení zařízení, sledování domácího života. Vzdálená kontrola zařízení pomocí existujících komerčních nástrojů není dostatečná pro monitorování a zjištění skutečného stavu zařízení. Většina IoT zařízení komunikuje s kontrolérem pomocí bezdrátové komunikace typu ZigBee, Bluetooth, Z-Wave apod, což neumožňuje použití tradičních monitorovacích technik jako je System Network Management Protocol (SNMP) [1] či NetFlow [2], které pracují nad TCP/IP. Tento článek představuje návrh implementace SNMP pro IoT sítě.

V současné době ukládání a zpracování dat o IoT zaří-

zeních probíhá ne na samotném zařízení, ale v cloudu. Používání cloudu je důležité pro agregaci a analýzu dat, což při velkém množství dat může výrazným způsobem zlepšit výpočetní výkon. Senzory a zařízení shromažďují data a provádějí akce, samotné zpracování a analýza obvykle probíhá v cloudu. Přístup do cloudu umožňují mobilní aplikace, které poskytují jednotlivé výrobce IoT zařízení. Uchování dat na cloud serverech má svoje nevýhody, například úniky a poškození dat, útoky na servery. K zpracování datových proudů slouží cloudová brána, která filtruje a případně blokuje data před provedením analýzy na cloudu, což způsobuje riziko ztráty dat. Data přenášena v rámci protokolů nemusí být zabezpečená. Útočník může napadnout síť a vytvořit spojení mezi zařízením a bránou nebo mezi bránou a serverem. Další nevýhodou této technologie je nemožnost získat informace o stavu IoT sítě mimo dat z cloudu. Vhodným řešením je proto zabezpečit bezdrátovou komunikaci monitorovacím systémem, který umožní sběr dat ze zařízení lokálně.

Pro správu a monitorování síťových zařízení se používá systém SNMP [1], který umožňuje průběžný sběr nejrůznějších dat pro potřeby správy sítě a jejich následné vyhodnocování. IoT zařízení většinou nepodporují protokol SNMP, neboť nepoužívají TCP/IP ale komunikují přes radiový signál, například Z-Wave. Pro zajištění aktuální informace o stavu IoT zařízení je vhodné rozšířit IoT komunikaci o podporu systému SNMP. Tato podpora může pomoci organizacím používajícím IoT zařízení předejít nežádoucímu ohrožení IoT sítě.

Navržené řešení monitoruje zařízení komunikují přes protokol Z-Wave pomocí agenta SNMP tak, aby bylo možné sledovat z řídící stanice stav IoT zařízení. Postup bude demonstrován na zařízeních siréna a dveřní senzor ze sady inteligentní domácí bezpečnosti D-Link[1], které komunikují s USB-vysílačem přes radiový signál Z-Wave.

## 2. Systém SNMP

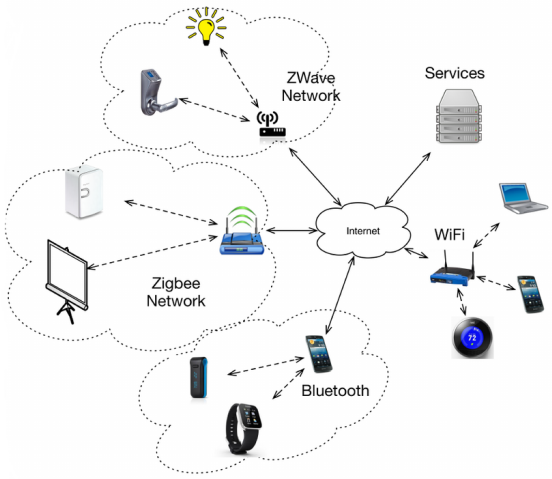
Systém SNMP označuje souhrn specifikací pro správu sítě, které tvoří samotný protokol SNMP, definici datových struktur a související koncepty [3]. Model správy sítě, který se používá pro správu sítě TCP/IP, zahrnuje následující klíčové prvky:

- monitorovací systém – řídící stanice (Network Management System, SNMP server) a SNMP agent,
- monitorované objekty definované pomocí jazyka Structure of Management Information (SMI) [4],
- uspořádání objektů do skupin, decentralizovaná správa objektů – databáze objektů MIB (Management Information Base) a
- komunikační protokol SNMP.

Systém SNMP vytváří strukturu agentů a řídící stanici NMS, která sleduje stav sítě. Řídící stanice a agenti komunikují pomocí protokolu SNMP, který pracuje na aplikační vrstvě. Agenti shromažďují informace o zařízeních a zapisují data do proměnných v databázi Management Information Base (MIB) [5]. Každý objekt je popsán ASN.1 a má své jméno, syntax a kódování. Řídicí stanice provádí monitorovací funkci získáním hodnoty objektů MIB. Komunikace mezi entitami protokolu se provádí výměnou zpráv přenášených v rámci jednoho UDP datagramu pomocí základních pravidel kódování ASN.1. Zpráva se skládá z verze identifikátoru, názvu komunity SNMP a datové jednotky protokolu (PDU) [1].

### 2.1 Struktura IoT sítě

Na obrázku č.1 je zobrazená obecná struktura bezdrátové komunikace. Protokol Z-Wave a Zigbee využí-




**Obrázek 1.** Přenos dat mezi prvky systému [6]

vají například termostaty, senzory pohybu, alarmy, osvětlení, klimatizace, IP kamery, zámky na dveře, ovladače audio/video techniky atd. Technologie Z-Wave a Zigbee používají smíšenou topologii (mesh), mají nízký výkon a jsou navržené pro přenos malého množství dat na krátké až střední vzdálenosti. Z-Wave využívá síťové pásmo menší než 1 GHz (frekvence se liší v různých zemích), což umožňuje bezkonfliktní provoz při souběhu s Wi-Fi a dalšími systémy založenými na standardu IEEE 802.11. Technologii Bluetooth využívají mobilní telefony, tablety, hodinky, sluchátka, počítače, USB atd. Bluetooth nepodporuje žádný vzdálený přístup, jen point-to-point. Nevýhodou

[1] Informace o produktu D-Link Security Kit

této technologie je přenos dat na krátkou vzdálenost, maximálně do 10 metrů, protože se k přenosu dat používá velmi malý výkon. Technologie Wi-Fi souvisí s připojením počítačů, tabletů a mobilních telefonů k internetu. Výhodou Wi-Fi technologie je její snadná integrace, možnost připojení i mimo běžné pracovní prostředí v případě propojení s veřejnou sítí a snadné rozšiřování sítě při přidávání dalších klientů.

V současném ekosystému IoT lze různé komponenty IoT široce rozdělit do tří tříd: uzly senzorů, uzly brány a služby IoT. Typické uzly senzorů sestávají z domácích spotřebičů nebo senzorů sledujících fyzické prostředí, které mají nízké výpočetní zdroje, přísná omezení energie a omezené komunikační zdroje. Uzel brány funguje jako agregátor dat senzorů a poskytuje připojení k jiným uzlům senzorů a poskytovatelům služeb. Služby IoT shromažďují data z různých uzlů brány, zpracovávají je v cloudu a poskytují služby specifické pro uživatele nebo události pomocí grafického rozhraní, oznámení nebo aplikace [6]. Chytré domácí mobilní aplikace pomáhají řešit problém správy více zařízení v inteligentních domácnostech z jednoho centrálního uživatelského ovládání. Uživatelé mohou například sledovat stav zařízení a upravovat ho podle svých požadavků vzdáleně. Nevýhodou technologie je nemožnost získat stav zařízení mimo dat z cloudu, protože technologie nepodporuje jiný monitorovací systém.

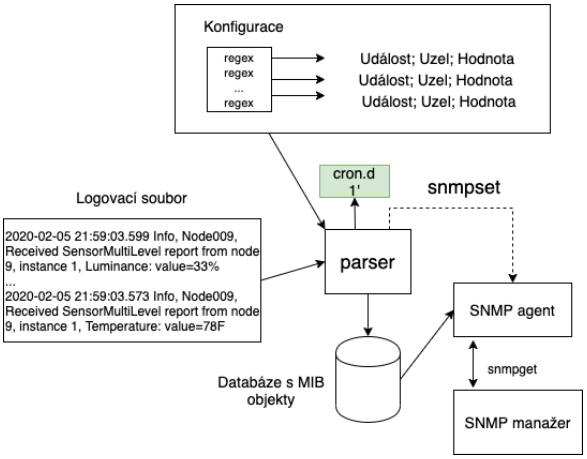
## 3. Návrh monitorování systému

Jeden z požadavků na navrhovaný systém je definice základních prvků, které budou systém tvořit. Pro demonstraci funkčnosti aplikace budeme používat testovací sadu, která obsahuje dveřní senzor a sirénu.

Zařízení posílají události o sobě přes USB-vysílač Z-Wave do nástroje Home Assistant[2]. Jsou to události typu stav zařízení, typ notifikace, typ senzoru, teplota, svítivost. Úkolem agenta je periodicky sbírat data z IoT komunikace, vyhodnocovat události (například pomoci předem vytvořených regulárních výrazu) a ukládat získané nejnovější hodnoty o zařízeních do hodnot proměnných v MIB databázi. Celkově navrhovaný systém se bude skládat z následujících částí:

- Konfigurační soubor – seznám pravidel pro výběr událostí z logu IoT kontroléru.
- Parsovací skript, který prochází logovací soubor, vyhodnocuje události pomocí vytvořených syntaktických pravidel a získané hodnoty ukládá do hodnot proměnných v databázi MIB.
- Démon `cron.d` spouští skript periodicky, například jednou za minutu.
- SNMP agent, který čte hodnoty objektů z MIB databáze a odpovídá na požadavky manažera vracením hodnot těchto objektů.
- MIB databáze s objekty.

Návrh systému je na obrázku č.2:




**Obrázek 2.** Přenos dat mezi prvky systému

Parsovací skript nebo jinak řečeno analyzátor nemusí běžet na stejném stroji s agentem, ale agent mu může poskytnout vzdálený přístup.

## 4. Implementace

Pro implementaci SNMP jsme použili knihovnu Net-SNMP. Net-SNMP[3] je nástroj pro vývoj nových aplikací SNMP v jazyku C, který podporuje rozšíření hlavního agenta za použití dynamicky načtených modulů, externích shell a Perl skriptů a protokolu AgentX. Pomocí Net-SNMP jsme vytvořili agenta, který spravuje IoT objekty ve své MIB databázi.

### 4.1 Sběr dat z IoT komunikace

Data z IoT komunikace ukládá do logovacího souboru nástroj Home Assistant. Každý řádek souboru obsahuje časovou značku události, uzel (zařízení) pro který je vytvořená událost, typ události a samotnou zprávu. Níže je uveden jeden řádek logovacího souboru:

```
2020-02-08 15:25:22.025 Detail, Node011,
Received: 0x01, 0x09, 0x01, 0x41, 0xd3,
0x9c, 0x01, 0x04, 0x10, 0x05, 0xe9
```

Některé aplikační zprávy mají formát 16-bitových hodnot a většinou je struktura následující [7]:

- 0x01 (1B, začátek zprávy)
- délka zprávy (1B)
- typ zprávy (1B, 0x00 - request, 0x01 - response)

[2] Instalace nástroje https://www.home-assistant.io/getting-started/

[3] Net-SNMP http://www.net-snmp.org/wiki/index.php/Tutorials#Coding_Tutorials

– funkce (1B, 0x04, 0x13)
– ID úzlu (1B)
– délka dat (1B)
– typ příkazu (1B, např. 0x84 Wakeup atd.)

Každý uzel také obsahuje seznam příkazů, které reprezentují atributy každého zařízení, například:

- COMMAND_CLASS_BASIC
- COMMAND_CLASS_SWITCH_BINARY
- COMMAND_CLASS_SENSOR_BINARY atd.

Nevýhodou nástroje Home Assistant je nečitelnost a složitá interpretace příkazu z logovacího souboru. Proto je potřeba implementovat formální pravidla, pomocí kterých se události vyhodnotí a uloží do MIB databáze v čitelném formátu. Úkolem této fáze je vybrat vhodné objekty, které chceme u zařízení monitorovat, definovat formát příkazů a vytvořit konfigurační soubor, který bude obsahovat regulární výrazy (dále jen RV) pro získání hodnot objektů. Soubor je možné sestavit ve formátu `csv` nebo `json`, aby ho bylo možné jednoduše parsovat.

Vytvořené řešení obsahuje konfigurační soubor, ve kterém každý řádek obsahuje RV události, název MIB objektu a RV příkazu, oddělené středníkem ve formátu `csv`.

Konfigurační soubor je pak vstupem pro parsovací skript, jinak řečeno analyzátor, který prochází nejnovější události logovacího souboru a hledá pro ně odpovídající pravidlo pro interpretaci. Analyzátor je navržen a implementován v jazyku Python 3. Pokud se pravidlo našlo, hledá se hodnota objektu z nalezeného řádku a ukládá se do odpovídajícího objektu v databázi MIB. Poslední čas události se ukládá do sdílené paměti, čímž se zajišťuje nalezení aktuální hodnoty události.

Aby aplikace vracela vždycky aktuální hodnoty objektů, je potřeba použít unixovou utilitu pro periodické automatické spouštění skriptu `cron.d`.

Příklad navrženého pravidla vypadá následně:

```
Received SensorMultiLevel report from node
\d{1,2},instance \d{1,2},
Luminance: value=\d{1,3}\%;
sensorLuminance;value=(.+?)\%
```

Pravidlo je určeno pro získaní hodnoty svítivosti u dveřního senzoru. Řádek obsahuje tři části oddělené středníkem, kde první část je RV, který odpovídá události:

```
2020-02-05 21:59:03.599 Info, Node009,
Received SensorMultiLevel report from node
9, instance 1, Luminance: value=33%
```

Druhá část je název MIB objektu *sensorLuminance*, do které se uloží hodnota získaná pomocí regulárního výrazu:

```
value=(.+?)\%
```

Další pravidlo, které získává stav dveřního senzoru, vypadá následně:

```
Node \d{1,2} has been marked as
\b(awake|asleep)\b;sensorState;\bas (.+?)$
```

Pravidlo pro získání stavu sirény:

```
Received SensorBinary report:
Sensor:\d{1,3} State=\b(On|Off)\b;
sirenState;State=(.+?)$
```

Pravidlo pro získání teploty dveřního senzoru:

```
Received SensorMultiLevel report from node
\d{1,2}, instance \d{1,2}, Temperature:
value=\d{1,3}F;sensorTemperature;
value=(.+?)F
```

## 4.2 Implementace SNMP agenta

Nástroj Net-SNMP umožňuje nakonfigurovat vlastní objekty do hlavního SNMP agenta nebo implementovat rozšířeného agenta (subagenta AgentX) s podporou vlastních objektů. Navržené řešení používá subagenta. Je to vlastní proces, který se připojuje k hlavnímu agentovi pomocí protokolu AgentX. Hlavního agenta je potřeba také nakonfigurovat vhodným způsobem, tzn. definovat community string, typ přístupu (read-write, read-only), název hosta, kterému se přístup povoluje, část stromu, ke které lze přistupovat, případně port (implicitně 161), uživatelé (pokud se používá SNMP verzi 3) a IP adresy, které mohou na portu naslouchat.

Rozšířený agent obsahuje implementaci inicializačních modulů pro objekty MIB, tj. přiřazení OID a následující registraci v MIB databázi při spouštění. Agent má hlavní funkci, která odpovídá na požadavky manažera získáním hodnoty MIB objektů.

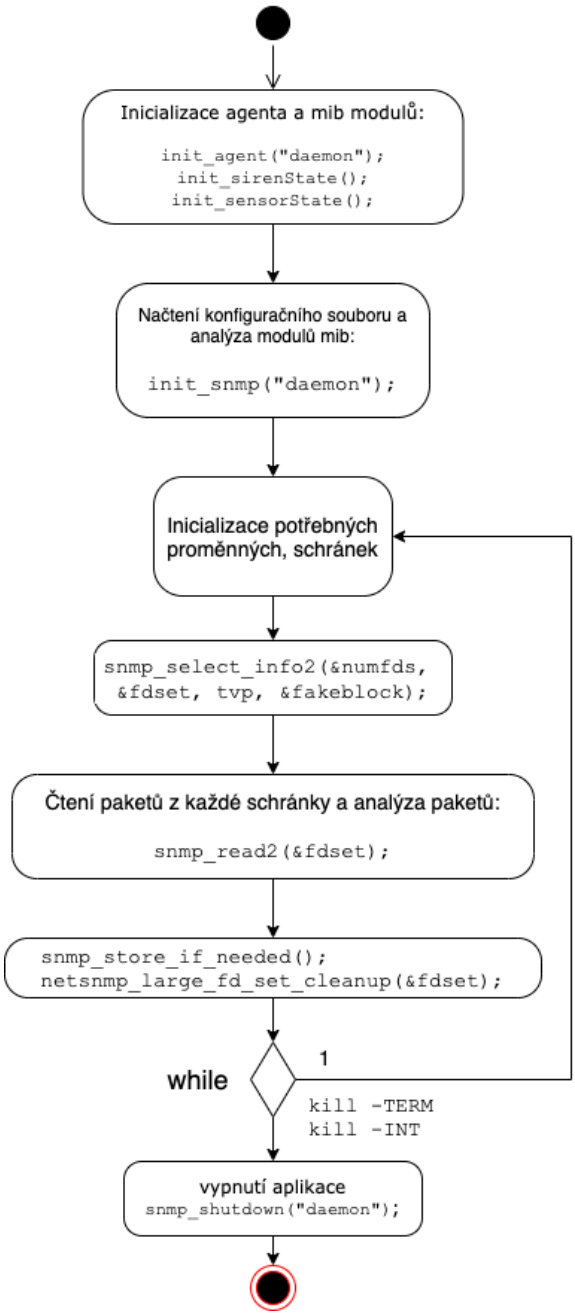
Agent používá knihovny *net-snmp-config, net-snmp-includes, net-snmp-agent-includes, large_fd_set* a *snmp_assert*.

Hlavní funkce kontroluje pakety přicházející na port SNMP a zpracovává je (funkce `snmp_read`), pokud jsou některé nalezeny.

Funkce `snmp_read()` přistupuje k množině schránek a čte z nich pakety, dále je analyzuje. Výsledná datová jednotka protokolu (PDU) je předána do rutiny zpětného volání pro relaci (funkce `snmp_session()`). Pokud se zpětné volání úspěšně vrátí, PDU a její požadavek budou odstraněny. Funkční diagram činnosti agenta je na obrázku č.3:

## 4.3 Objekty SNMP pro monitorování IoT sítě

Nejdůležitější částí pro vytvoření vlastní MIB databáze je seznam objektů, které budou uloženy v MIB. Tyto objekty musí striktně dodržovat syntax ASN.1. Pokud




**Obrázek 3.** Diagram činnosti

chceme definovat vlastní objekty, které chceme sdílet se světem, potřebujeme je zařadit do MIB stromu tak, aby nebyly v konfliktu s jinými objekty. Pokud definujeme objekty, které budeme používat pouze interně, můžeme je vložit do experimentálního stromu. Jen si musíme dát pozor na překrývání OID[4]. Při napsání vlastní MIB jsem použila ukázkový příklad Net-SNMP a definovala vlastní MIB do větve *netSnmpExamples*. Takto vypadá definice jednoho z objektů MIB databáze, konkrétně objekt popisující sirénu:

```
sirenState OBJECT-TYPE
SYNTAX  DisplayString(SIZE(0..3))
MAX-ACCESS  read-write
STATUS      current
DESCRIPTION "Status of the device
may take on values off/on"
DEFVAL { "off" }
::= { iotAgentModules 2 }
```

Tento objekt obsahuje hodnotu typu string o velikosti 0-255 znaků s implicitní hodnotou ”off”. Další objekty, které se použily pro testování funkčnosti agenta, jsou definované následně:
Objekt pro sledování svítivosti:

```
sensorLuminance OBJECT-TYPE
SYNTAX  Integer32
MAX-ACCESS  read-write
STATUS      current
DESCRIPTION
    "Door/Window luminance"
DEFVAL { 0 }
::= { iotAgentModules 4 }
```

Objekt pro sledování teploty:

```
sensorTemperature OBJECT-TYPE
SYNTAX  Integer32
MAX-ACCESS  read-write
STATUS      current
DESCRIPTION
    "Door/Window temperature"
DEFVAL { 0 }
::= { iotAgentModules 5 }
```

## 5. Testování

V práci byl vytvořen systém SNMP, který poskytuje monitorovací funkci zařízením komunikujícím přes radiový signál Z-Wave. Ve virtuálním prostředí běžel agent a rozšířený agent. Zároveň na počítací mimo virtuální prostředí běžel nástroj Home Assistant. K počítači byl také připojený USB-vysílač přes V/v sběrnici, který naslouchal na Z-Wave síti. Jednou za minutu se spouštěl parsovací skript. Při experimentech se SNMP manažer dotazoval agenta na hodnoty objektů zařízení. Bylo vidět, jak se mění hodnoty objektů zařízení, například teplota a svítivost, stav (asleep nebo awake) u dveřního senzoru a stav (On/Off) u sirény.

### Test 1

Logovací soubor obsahuje událost „stav dveřního senzoru“:

```
2020-02-05 21:59:05.122 Info, Node009,
Node 9 has been marked as awake
```

[4] Viz http://net-snmp.sourceforge.net/wiki/index.php/Writing_your_own_MIBs

kterou SNMP agent zpracoval jako:

```
$ snmpget -v2c -c public
192.168.251.2 1.3.6.1.4.1.8072.2.4.1.1.3.0

NET-SNMP-EXAMPLES-MIB::
netSnmpExamples.4.1.1.3.0 = STRING: "awake"
```

### Test 2

Později došlo ke změně hodnoty události:

```
2020-02-05 21:59:05.174 Info, Node009,
Node 9 has been marked as asleep
```

která se hned uložila do databáze

```
NET-SNMP-EXAMPLES-MIB::
netSnmpExamples.4.1.1.3.0 = STRING:
"asleep"
```

### Test 3

Logovací soubor obsahuje událost „teplota senzoru“:

```
2020-02-05 21:59:03.573 Info,
Node009, Received SensorMultiLevel report
from node 9, instance 1,
Temperature: value=78F
```

agent pak uložil hodnotu následně:

```
NET-SNMP-EXAMPLES-MIB::
netSnmpExamples.4.1.1.5.0 = INTEGER: 78
```

### Test 4

Logovací soubor obsahuje událost „svítivost senzoru“, která vypadá následně:

```
2020-02-05 21:59:03.599 Info, Node009,
Received SensorMultiLevel report from node
9, instance 1, Luminance: value=33%
```

do databáze se pak uložila hodnota:

```
NET-SNMP-EXAMPLES-MIB::
netSnmpExamples.4.1.1.4.0 = INTEGER: 33
```

### Test 5

Ukázka události „stav sirény“

```
2020-02-11 14:34:23.336 Info,
Node012, Received SensorBinary report:
Sensor:1 State=Off
```

kterou SNMP agent zpracoval jako:

```
NET-SNMP-EXAMPLES-MIB::
netSnmpExamples.4.1.1.2.0 = STRING: "Off"
```

### Test 6

Dále došlo ke změně hodnoty události:

```
2020-02-11 14:34:23.345 Info, Node012,
Received SensorBinary report: Sensor:1
State=On
```

která se následně zpracovala jako:

```
NET-SNMP-EXAMPLES-MIB::
netSnmpExamples.4.1.1.2.0 = STRING: "On"
```

## 6. Závěr

IoT zařízení v dnešní době používají spousta organizaci pro automatizaci a bezpečnost. Spolu s tím narůstá počet útočníku a hrozeb na zařízení, které neposkytují dostatečnou bezpečnost. Systém SNMP umožňuje správu sítě a průběžný sběr nejrůznějších dat o zařízeních. Je to standard sledovaní řízení bezpečnosti sítě, zpracování chyb a odhalení možných útoků. Některá IoT zařízení tento systém však nepodporují. Přínosem této práce je rozšíření Z-Wave komunikace o systém SNMP. Tento nástroj je důležitý pro organizace používající IoT zařízení pro zkoumání stavu zařízení na útoky, aby je bylo možné včas odhalit. Tato práce popisuje postup vytvoření SNMP agenta v jazyku C, MIB databáze a navrhuje analyzátor, který interpretuje data z logovacího souboru na hodnoty proměnných v databázi MIB. Daný postup lze aplikovat i na jiné IoT sítě, které lokálně používají radiové spojení, například typu Bluetooth, Zigbee apod. Je potřeba pouze adaptér pro danou síť, softwarový nástroj Home Assistant pro zpracování události. Dále je potřeba popsat události, které se budou odchytávat pomocí regulárních výrazů a vytvořit definici objektů MIB.

## 7. Poděkování

Ráda bych poděkovala svému vedoucímu Ing. Petrovi Matouškovi Ph.D., M.A., za odborné rady a vedení při vypracování této práce.

## Literatura

[1] J. Case et al. A Simple Network Management Protocol (SNMP). IETF RFC 1157, 1990.

[2] B. Claise. Cisco Systems NetFlow Services Export Version 9. IETF RFC 3954, 2004.

[3] William Stallings. *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*. Addison-Wesley, 3. edition, 1998. ISBN: 0-201-48534-6.

[4] M. Rose and K. McCloghrie. Structure and Identification of Management Information for TCP/IP-based Internets. IETF RFC 1155, 1990.

[5] R. Presuhn et al. Management information base (mib) for the simple network management protocol (snmp). IETF RFC 3418, 2002.

[6] Amit Sheth. Semantic gateway as a service architecture for iot interoperability, October 2014. https://arxiv.org/abs/1410.4977.

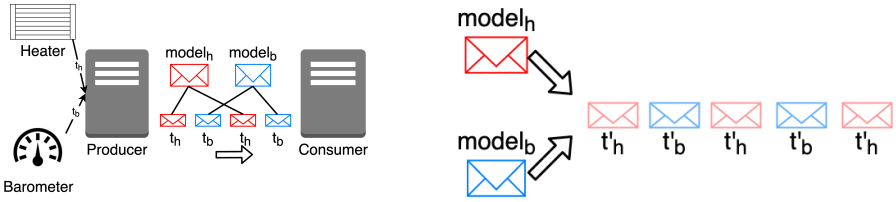
[7] Onur Dundar. *Home Automation with Intel Galileo*. Packt Publishing Ltd, 3. edition, 2015. ISBN: 1-785-28726-5.

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

# A Library for Tree Structures Analysis and Similar Structures Generation

Sergey Panov*




**Abstract**
Testing multicomponent systems in IT and IoT that process the sequences of different messages is a complicated task. Why it is complicated? Because of number of components, asynchronous interaction, different combinations of actions to test, test environment differs from real environment and others. This paper introduces an idea how to generate complex input data for system testing while requiring minimum intervention from a developer. The test data generation is based on analysis of traces of communication in a real system, and reproduction of similar traces for testing purposes. The paper also proposes a framework for initial analysis of messages transferred within the recorded communication. The problem can be solved using different abstract models: the message model, the communication model. The result of this work is the implemented library for creating a message model with a set of operations for working with this model.

**Keywords:** Abstract data structures — Testing

**Supplementary Material:** *N/A*

*xpanov00@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

## 1. Introduction

The reason for this project is the design of a reliable framework for testing the multicomponent IT/IoT systems that process massive streams[1] of messages.

This work is devoted to the design and implementation of the library that allows creating the message abstract model. The models can be used to generate similar messages. The generator can mutate the messages that sent to the component under the test, and its reaction to mutated messages can be observed. The message model can be evaluated using the ratio of the model size versus the training set size, its overhead, accuracy and expressiveness (explained below).

One attempt to implement something similar was made by Ondřej Znojil at the Faculty of Information Technologies at Brno University of Technology, described in his bachelor thesis [1]. The potential structure lose is the disadvantage of this implementation. Another cons is the inability of model creation at the runtime. This solution also requires a configuration file to start, that makes it need to have preliminary knowledge; and is not able to train on a set of messages.

Another attempt of implantation was made by Dušan Želiar at FIT BIT [2]. The solution can train from the set of messages, thanks to using the RabbitMQ[2]. The tight coupling with the components from TESTOS[3]

[1] In this article, the *stream* has a meaning of potentially infinite sequence.

[2] RabbitMQ is the open source message broker. https://www.rabbitmq.com/

[3] TESTOS (Test Tool Set) platform supports automation of software testing, see http://testos.org/.

is disadvantage of the solution. Another cons is the limited amount of meta-information about the messages that could be stored. The clustering[4] algorithms usage also makes the solution suffer from the potential structure loss.

With our solution, we are able to process sets as well as streams of messages. Thanks to the designed model structure, we do not lose the message structure. We are able to store different kinds of meta-information that can be added/removed at runtime. Moreover, the library has template methods for the message structure manipulation and those methods can be enriched using the implementations of prepared interfaces. The library is stand-alone and does not couple with other platforms and it can be easily expanded by new functionality and used in different projects.

The design of this library not only provides flexibility in use but also provides an opportunity to expand it. Thanks to different design patterns (such as *strategy*, *chain of responsibility*, *composite*), the library is easy to refine and expand with a new functionality. It is also a stand-alone project and can be used with a different purpose rather than a generation of the sequence of messages.

Section 2 starts with the working example and describes how to use hierarchical data structures for the message representation. Section 3 describes the designed functionality. Section 4 explains the experiments conducted with the library and summarizes the results.

## 2. Hierarchical Data Structures and Their Use for Message Representation

Let's start with a working example. Let there be a producer and consumer. The heater and the barometer periodically report their temperature and pressure to the producer. The producer enriches received messages by adding a timestamp. After enrichment, the producer sends the messages to the consumer. The consumer executes some logic based on the received messages. So this is our communication model.

The heater and the barometer send the messages of different types: one of them reports the temperature and other the pressure. Having a sample of communication, we can create models describing the messages. Those models describe a message structure and contain meta-information such as the number of messages, min/max value of the temperature/pressure, the standard deviation of temperature/pressure, and others.

[4] **Clustering** is a process of grouping a set of object in such a way that objects in the same group (cluster) are more similar to each other.

This idea is depicted in Figure 1. The red messages report the temperature and blue ones, the pressure.

The example described above is a simple industrial network. Now let's say that messages sent by the producer are documents in XML format. An XML document can be represented as a tree structure. So we can say that the producer generates a stream of tree structures and sends it to the consumer, and the stream consists of two types of messages.

In this work we consider only a specific subset of all possible XML documents. For us, the XML document can not have inner element and text content at the same time. If an element has an inner element and text content, the text content is ignored, see Listing 1. The `bar` is a text content of the `<outer>` element and is ignored. We do it because of the low probability of such messages and the formal model (described in the text below).

```
<outer>
   bar
   <inner>foo</inner>
</outer>
```

**Listing 1.** An example of XML document with ignored text content.

For a tree-like representation of XML document the next approach is chosen:

- a node without parent (*root* node) represents an outermost element,
- a *composite node* represents each element that has at least one inner element,
- a *primitive node* represents an element that has no inner elements,
- the *attribute node* represents an attribute of an XML element.

Listing 2 and Figure 2 give an idea of tree-like XML document representation.

It is worth noting that the messages of the same type not necessarily have the same structure. E.g., if the heating coil has not yet warmed up, the heater can send a message reporting the "warming up" mode till the coil have the sufficient temperature. In order to create a message model, we need to capture the structures of all possible messages of the same type as well as meta-information about the content. In this work, *the structure describing all trees carrying the messages of the same type* is called an **abstract-tree**. The abstract-tree *also contains a probability of the particular node present in the path*.

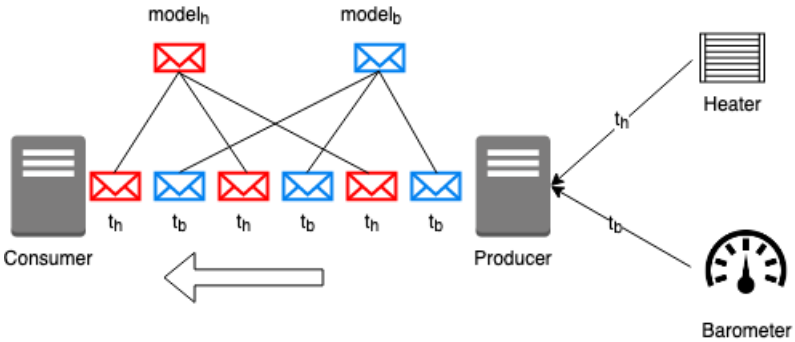



**Figure 1.** Example of a simple industrial network.

```
<device>
   <name>Heater</name>
   <id>H-42</id>
   <timestamp format="MMM dd HH:mm:ss">
       Mar 16 08:12:04</timestamp>
</device>
```

**Listing 2.** Example of reported XML document.

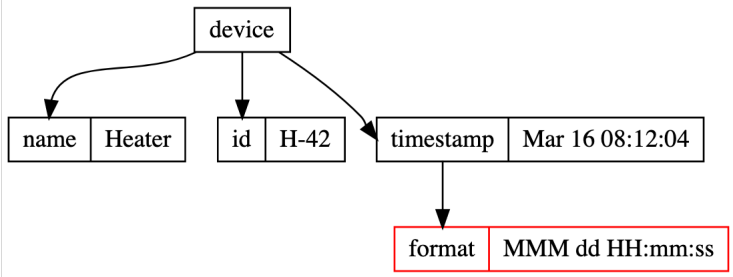



**Figure 2.** Tree-like representation of XML document from Listing 2.

## 3. The Library Design

The goal of this work is the design and implementation of a library for the creation of model of messages with tree-like structures. In the end, the implemented library should:

- be able to **represent as tree structures**,
- be able to **create the message model without structure loss**, i.e., construct an abstract tree above the messages set and collect meta-information,
- provide a set of **operations** (***group, reduce, merge***) to manipulate with a tree structure.

### 3.1 Unified Representation

A way of representing hierarchical data as tree structures was already depicted in Figure 2. To summarize and formalize it: we consider only the subset of XML documents and represent them as tree structures, a tree structure is a tuple

$$T = (C, P, A, E)$$

where:

- $C$ is a **composite node** and represents an XML element that contains at least one inner XML element,
- $P$ is a **primitive node** and represents an XML element that does not contain any inner XML element,
- $A$ is an **attribute node** and represents an attribute of an XML element,
- $E \subseteq \{(u,v) | u \in C \wedge v \in C \cup P \cup A\} \cup \{(u,v) | u \in P \wedge v \in A\}$ is a set of **edges** connecting all nodes and between two different nodes only one path[5] exists.

The text content of an element with an inner element is ignored. Table 1 describes a parent-child relationship between the nodes.

Any structured document in JSON or YAML formats can be transformed into XML with predefined node type translation.

**Table 1.** A tree structure hierarchy description. The *null* parent means that the node is a root.

| Node | Parent | Children | Type of value |
|---|---|---|---|
| $C$ | $\{C, null\}$ | $\{C, A, P\}^*$ | None |
| $P$ | $\{C, null\}$ | $A^*$ | Strings |
| $A$ | $\{C, P\}$ | None | Strings |

### 3.2 The Operations

The **group** operation is a unary operation and creates equivalence partitions on the collection of nodes based on the equivalence criteria (see Section 3.3). This operation is useful when it is needed to apply an abstract function on the cluster of nodes, e.g., the reduce operation or choosing particular nodes from the group. Figures 3 and 4 depict this operation.

[5] In graph theory a **path** is a sequence of edges which join a sequence of nodes.

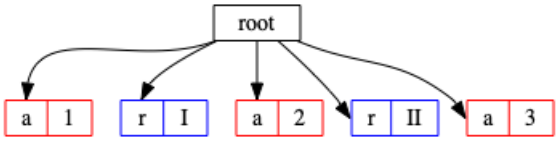



**Figure 3.** Before **group** operation.

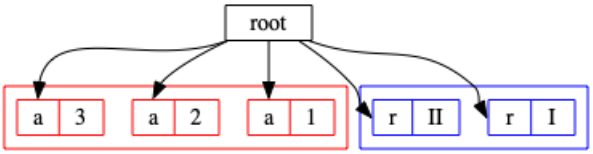



**Figure 4.** After **group** operation.

The **reduce** operation is a unary operation and creates a single representational node from multiple nodes. The representational node describes a set of nodes and holds enough data for generating a set of similar nodes. The nodes reduction leads to a more compact tree representation with no data loss, up to order of nodes in different equivalence classes. The reduction also collects meta-information such as the number of reduced nodes. Figures 5 and 6 depict this operation.

The **merge** is a binary operation and is essential for abstract-tree creation. The essence of this operation is to match the equivalent nodes on the same level among different trees and merge them. The two nodes that are to be merged carry out the same semantic information. The nodes that did not match any other node are just added into the result tree. While merging the meta-information is stored, e.g., numbers of times the node was present among all trees, min (max) value of the node, or a history of the values. Let there be two trees depicted in Figure 7 and 8. Figure 9 depicts the result of merging these trees.

### 3.3 Equivalence Criteria

An **equivalence criterion** describes a case when two or more nodes are equal. Such criteria are used for grouping and for merging. In the case of complex communication with lots of different messages, those criteria can possibly change at runtime; therefore, the library should provide predefined set of different criteria and means for combining them. Let there be two nodes $A, B$; for the purpose of this work, the following criteria were proposed:

- **by-name** states that $A, B$ are the same iff they have the same name,
- **by-value** states that $A, B$ are the same iff they have the same value,
- **by-children** is the recursive criterion and states that $A, B$ are the same iff they are same, and their corresponding (the place of the node in the tree defines correspondence) child-nodes are also the same; the nodes equivalence is defined by the *by-name*, *by-value*, and *by-attributes* criteria,
- **by-attributes** states that $A, B$ are the same iff the set of attribute-nodes of $A$ ($B$) is a subset of a set of attribute-nodes of $B$ ($A$).

## 4. Experiments

Firstly we focused on *functional requirements*. The simple proof of concept application was created to prove the library usability. This application uses a simple training set with 10 XML documents and works with the next steps:

- reads the documents one by one,
- for each document creates its tree-like representation,
- traverse the representation and **groups** the child nodes of each node by **by-name** criterion and **reduces** each group, so creates a single representational node, the representational nodes become child nodes for the current one,
- the result of the previous step is the compact representation of the tree-like representation,
- all those representations are **merged** into one abstract-tree, and each value of each node is stored,
- the created model is used to generate the similar XML documents to ones from the training set.

Among function requirements, some of the *non-functional requirements* were verified.

**Extensibility**. The library was designed with the idea that some of the components can be expanded, so using the design patterns such as *composite*, *command*, and the *chain of responsibility* give a wide potential for future extension.

**Maintainability**. The library was designed to be maintainable and self-documented. All operations are accessed by components through the interface, so a new implementation of an interface can easily replace the old one.

**Performance**. The library is be able to process a massive amount of messages; it is necessary to parse them and create the tree structures as well as perform operations like *group*, *merge*, *reduce* relatively fast. It also needs to create as smallest models as possible without the structure loss and with enough meta-information about messages in order to generate similar ones. For the performance tests of the library, the real sample of communication of the industrial network was chosen – the colleagues from VeriFIT[6]

[6] https://www.fit.vut.cz/research/group/verifit/.en

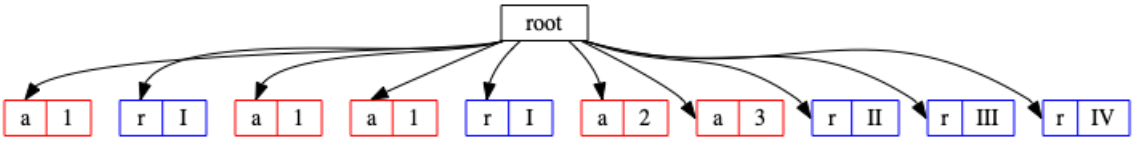



**Figure 5.** Before **reduce** operation.

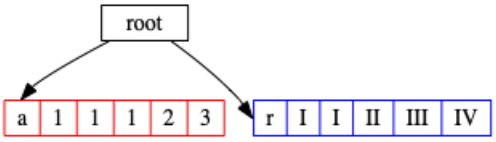



**Figure 6.** After **reduce** operation.

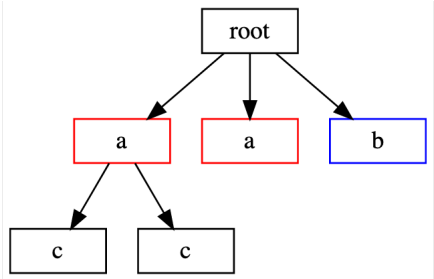



**Figure 7.** Left tree $T_1$.

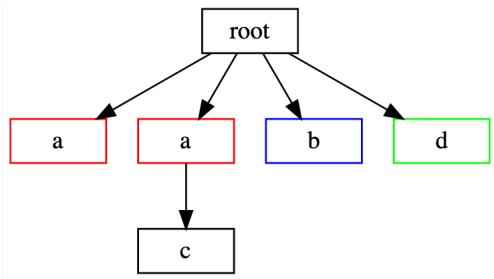



**Figure 8.** Right tree $T_r$

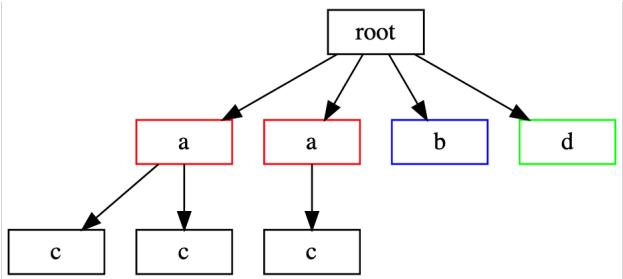



**Figure 9.** The merged trees $T_l$ and $T_r$.

research group preprocessed that sample and provided 7 clusters of XML messages.

The first set of experiments are experiments with the model sizes. The *worst-case* model simply stored seen values of each node, the *best-case* model does not store any value, but only the structure. The size of the *worst-case* model is almost 5 times more than the size of the original set. The reason for this is the object representation of XML elements and the need to serialize meta-information about the collections of values and references on other objects. The size of the *best-case* model is significantly less than the original size. Those experiments demonstrate that with an efficient values analyzer, that allows ignoring some values, it is possible to significantly reduce the model size without losing the exactness of generated values.

The second set of experiments is with the time of the creation of tree structure from the input XML document and merging two trees. Table 2 contains these measurements and Figure 10 depicts the ratio of sizes.

The third set of experiments is made to check the model's *accuracy*, *structure-overhead*, and *expressiveness*. Let there be an abstract-tree $AT = (C_A, P_A, A_A, E_A, \sigma)$ created from the collection of trees with the different structures $TC = \{T_1, T_1 \dots T_n\}$, where $T_i = (C_i, P_i, A_i, E_i)$, and let there be a set of trees describing XML documents with the different structures $TC' = \{T'_1, T'_2 \dots T'_n\}$, where $T'_i = (C'_i, P'_i, A'_i, E'_i)$, then:

- *accuracy* is a ratio between the number of nodes from $T'_i$ covered[7] by nodes from $AT$ and total numbers of nodes from $T'_i$,
- *structure-overhead* is a ratio between the number of nodes from $AT$ used to cover $T'_i$ and the total number of nodes in $AT$,
- *expressiveness* is a ratio between the number of nodes from $AT$ used to cover a $T'_i$ and the number of covered nodes from $T'_i$.

The experiments were conducted with the following steps:

1. create a model $AT$ of all trees $TC$ describing the XML documents from the training set,

[7] *Cover* means to match the nodes from $T'_i$ with the nodes from $AT$.

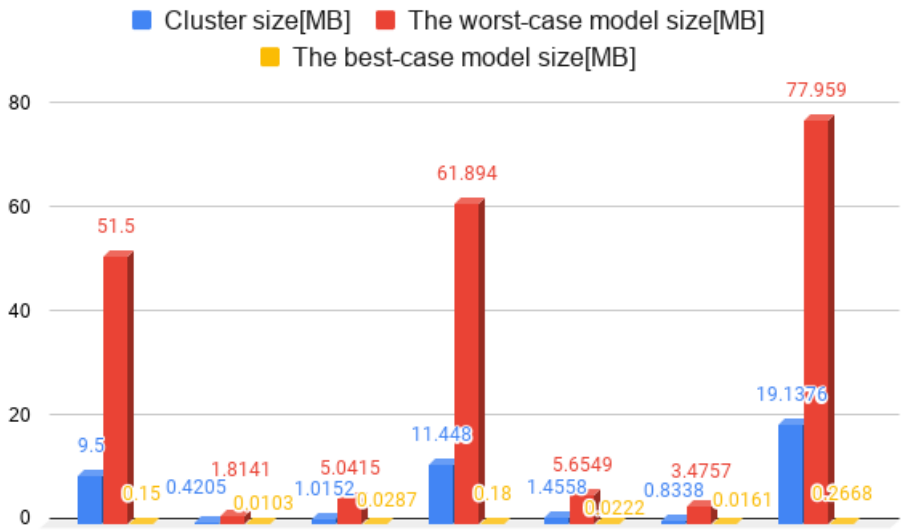



**Figure 10.** The comparison of the sizes of the original cluster, the worst-case model, and the best-case model.

**Table 2.** The sum-up of the experiments with models' sizes and times of the tree structures creation and trees merging.
**Num. of docs.** defines the number of documents in a cluster,
**Cluster size** defines the size of the whole cluster,
**Worst/best-model size** defines the size of the model in the worst/best cases,
**Avg. tree creation time** defines the average time of creating a tree-representation of one document in the cluster (+/- standard deviation),
**Avg. trees merging time** defines the average time of merging two trees from the same cluster (+/- standard deviation).

| **Cluster** | 1_432 | 2_425 | 3_432 | 4_432 | 5_432 | 6_432 | 7_432 |
|---|---|---|---|---|---|---|---|
| **Num. of docs.** | 432 | 425 | 432 | 432 | 432 | 432 | 432 |
| **Cluster size[MB]** | 9.5 | 0.42 | 1.015 | 11.448 | 1.455 | 0.833 | 19.137 |
| **Worst-model size[MB]** | 51.5 | 1.81 | 5.04 | 61.9 | 5.65 | 3.48 | 77.96 |
| **Best-model size[MB]** | 0.15 | 0.01 | 0.03 | 0.18 | 0.02 | 0.016 | 0.27 |
| **Avg. tree creation time (+/- std. deviation)[ms]** | 3.3 (+/- 2.6) | 0.02 (+/- 0.16) | 0.13 (+/- 0.6) | 4.18 (+/- 3.3) | 0.11 (+/- 0.6) | 0.04 (+/- 0.27) | 6.66 (+/- 3.67) |
| **Avg. trees merging time (+/- std. deviation)[ms]** | 1.97 (+/- 2.1) | 0.002 (+/- 0.05) | 0.08 (+/- 0.5) | 1.83 (+/- 2.03) | 0.02 (+/- 0.19) | 0.009 (+/- 0.096) | 3.5 (+/- 3.46) |

2. in order to check the *accuracy*, *structure-overhead*, and the *expressiveness* of $AT$ cover each tree, from $TC'$ that represent the documents from the test set, and record:
   - the number of nodes in $T_i'$,
   - the number of covered nodes of $T_i'$,
   - total nodes of $AT$,
   - nodes from $AT$ used to cover $T_i'$.
3. calculate *accuracy*, *structure-overhead*, and *expressiveness* based on recorded values,
4. remove one document from the training set and repeat the loop start with step 1 until the training set is not empty.

Some of results of experiments are present in tables 3, 4, 5, and 6 where:

- **the representational from the cluster** is a cluster containing $T_i'$,
- **used nodes from** $AT$ is a number of nodes from the abstract-tree used to cover the nodes from $T_i'$,
- **total nodes in** $T_i'$ is a total number of nodes of $T_i'$
- **the covered nodes of** $T_i'$ is a number of nodes from the $T_i'$ that are covered by nodes from $AT$.

If a document is a part of the training set, another document from the same cluster is 100% covered, so *accuracy* is 1. If the document is not present in the training set, the *accuracy* is 0 and the *structure-overhead* is

1 (no one model node was used). The *expressiveness* equal to 0 means that $N$ nodes from $AT$ cover $N$ nodes of $T_i'$, so there is a bijection between the model nodes subset and $T_i'$ nodes subset. The *accuracy* in the interval $(0;1)$ means that some parts of XML documents have the same structure starting with the root element.

## 5. Conclusions

This paper suggests an approach of how to analyze the streams of messages in multicomponent systems and generate a similar ones with the test purpose. In order to do it, it needs to create models describing the streams from different perspectives: the communication model and the message model. Here is described the implemented library for creating a message model.

In order to create a message model, it is necessary to have a set of messages that do not necessarily have the same structure. Merging the tree representations of those messages and storing meta-information that matters for a particular domain, it is possible to create a model without the structure loss and content loss. This library is already used in a more significant project that intended to develop a system for automatic analysis of streams of messages and generating similar ones. This library also can be used with the purpose of anomaly detection in the message structures.

## Acknowledgements

This work was supported by National Cybersecurity Competence Centre, by TAČR project no. TN01000077/06. I would like to thank my supervisor Aleš Smrčka and colleagues Tomáš Fiedor and Martin Hruška from VeriFIT research group for weekly consultations and discussions.

## References

[1] Znojil Ondřej. Detektory strukturovaných dat pro účely testování software. Master's thesis, Brno university of technology, Božetěchova 1, 612 00 Brno-Královo Pole, 2018.

[2] Želiar Dušan. Automatizovaná syntéza stromových struktur z reálných dat. Master's thesis, Brno university of technology, Božetěchova 1, 612 00 Brno-Královo Pole, 2019.

**Table 3.** Experiment with $|TC| = 7$ (size of the training set), and $|C_A \cup P_A \cup A_A| = 266$ (total number of nodes in the model).

| **The representational from the cluster** $(T_i')$ | 1_432 | 2_425 | 3_432 | 4_432 |
|---|---|---|---|---|
| **Used nodes from** $AT$ | 83 | 29 | 25 | 54 |
| **Total nodes in** $T_i'$ $(\|C_i' \cup P_i' \cup A_i'\|)$ | 673 | 29 | 119 | 798 |
| **The covered nodes of** $T_i'$ | 673 | 29 | 119 | 798 |
| **Accuracy** | $\frac{673}{673} = 1$ | $\frac{29}{29} = 1$ | $\frac{119}{119} = 1$ | $\frac{798}{798} = 1$ |
| **Structure-overhead** | $1 - \frac{83}{266} \approx 0.69$ | $1 - \frac{29}{266} \approx 0.89$ | $1 - \frac{25}{266} \approx 0.9$ | $1 - \frac{54}{266} \approx 0.79$ |
| **Expressiveness** | $1 - \frac{83}{673} \approx 0.88$ | $1 - \frac{29}{29} = 0$ | $1 - \frac{25}{119} \approx 0.79$ | $1 - \frac{54}{798} \approx 0.93$ |

**Table 4.** Experiment with $|TC| = 6$, and $|C_A \cup P_A \cup A_A| = 237$.

| **The representational from the cluster** $(T_i')$ | 1_432 | 2_425 | 3_432 | 4_432 |
|---|---|---|---|---|
| **Used nodes from** $AT$ | 83 | 0 | 25 | 54 |
| **Total nodes in** $T_i'$ $(\|C_i' \cup P_i' \cup A_i'\|)$ | 673 | 29 | 119 | 798 |
| **The covered nodes of** $T_i'$ | 673 | 0 | 119 | 798 |
| **Accuracy** | $\frac{673}{673} = 1$ | 0 | $\frac{119}{119} = 1$ | $\frac{798}{798} = 1$ |
| **Structure-overhead** | $1 - \frac{83}{237} \approx 0.65$ | 1 | $1 - \frac{25}{237} \approx 0.89$ | $1 - \frac{54}{237} \approx 0.77$ |
| **Expressiveness** | $1 - \frac{83}{673} \approx 0.88$ | 0 | $1 - \frac{25}{119} \approx 0.79$ | $1 - \frac{54}{798} \approx 0.93$ |

**Table 5.** Experiment with $|TC| = 5$, and $|C_A \cup P_A \cup A_A| = 225$.

| **The representational from the cluster** $(T_i')$ | 1_432 | 2_425 | 3_432 | 4_432 |
|---|---|---|---|---|
| **Used nodes from** $AT$ | 83 | 0 | 13 | 54 |
| **Total nodes in** $T_i'$ $(\|C_i' \cup P_i' \cup A_i'\|)$ | 673 | 29 | 119 | 798 |
| **The covered nodes of** $T_i'$ | 673 | 0 | 99 | 798 |
| **Accuracy** | $\frac{673}{673} = 1$ | 0 | $\frac{99}{119} = 0.83$ | $\frac{798}{798} = 1$ |
| **Structure-overhead** | $1 - \frac{83}{225} \approx 0.63$ | 1 | $1 - \frac{13}{225} \approx 0.94$ | $1 - \frac{54}{225} = 0.76$ |
| **Expressiveness** | $1 - \frac{83}{673} \approx 0.88$ | 0 | $1 - \frac{13}{99} \approx 0.87$ | $1 - \frac{54}{798} \approx 0.93$ |

**Table 6.** Experiment with $|TC| = 4$, and $|C_A \cup P_A \cup A_A| = 150$.

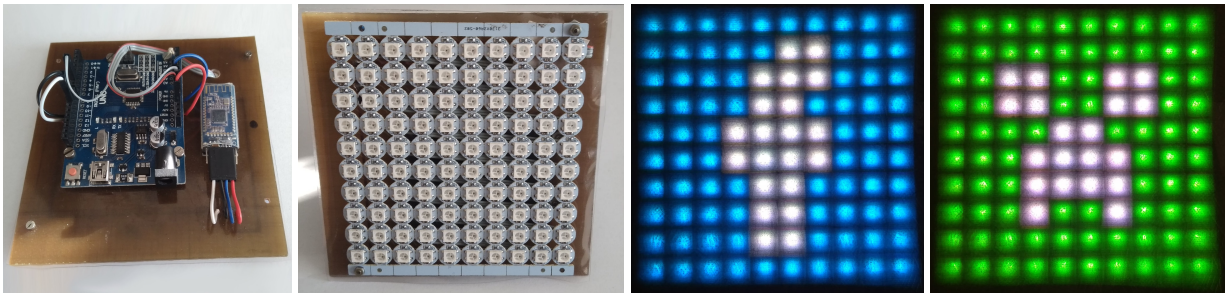
| **The representational from the cluster** $(T_i')$ | 1_432 | 2_425 | 3_432 | 4_432 |
|---|---|---|---|---|
| **Used nodes from** $AT$ | 8 | 0 | 13 | 54 |
| **Total nodes in** $T_i'$ $(\|C_i' \cup P_i' \cup A_i'\|)$ | 673 | 29 | 119 | 798 |
| **The covered nodes of** $T_i'$ | 8 | 0 | 99 | 798 |
| **Accuracy** | $\frac{8}{673} = 0.012$ | 0 | $\frac{99}{119} = 0.83$ | $\frac{798}{798} = 1$ |
| **Structure-overhead** | $1 - \frac{8}{150} \approx 0.95$ | 1 | $1 - \frac{13}{150} \approx 0.91$ | $1 - \frac{54}{150} = 0.64$ |
| **Expressiveness** | $1 - \frac{8}{8} = 0$ | 0 | $1 - \frac{13}{99} \approx 0.87$ | $1 - \frac{54}{798} \approx 0.93$ |

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

# Hardwarový indikátor oznámení pro telefony se systémem Android

Roman Bártl*



**Abstrakt**
Cílem této práce je navrhnout a vytvořit notifikační zařízení, které bude zobrazovat nejrůznější upozornění z chytrých telefonů. Jedná se o upozornění na příchozí hovory, stav baterie a notifikace nainstalovaných aplikací. Jádrem zařízení je jednodeskový počítač Arduino. Součástí práce je i implementace aplikace pro operační systém Android, která slouží jak pro komunikaci s tímto zařízením, tak i k samotnému zachytávání zmíněných událostí. Pro komunikaci je použita technologie Bluetooth Low Energy. Je kladen důraz na jednoduchost a srozumitelnost celého systému včetně zajištění co největší možné kompatibility napříč verzemi systému Android.

**Klíčová slova:** Arduino — Android — Bluetooth Low Energy — Vestavěný systém — Notifikace

**Přiložené materiály:** Demonstrační video

*xbartl06@fit.vutbr.cz, *Fakulta informačních technologií, Vysoké učení technické v Brně*

## 1. Úvod

Motivací pro tuto práci je fakt, že se v poslední době začalo ustupovat od notifikačních LED diod u většiny mobilních zařízení kvůli tzv. „bezrámečkovým“ displejům, a uživatelé tak ztrácí možnost být bez vyrušení informování o příchozích upozorněních. A i když mobilní zařízení tuto diodu obsahuje, bývá často její nastavení dosti omezené a pro některé uživatele toto může být nedostačující. Proto podobné notifikační zařízení může přijít náročnějším uživatelům velmi vhod. Dále může toto zařízení například sloužit sluchově postiženým, jež jsou závislí na vizuálních podnětech.

Vestavěný systém je tvořen ze dvou částí – zobrazovací zařízení a aplikace pro telefony se systémem Android. Základním prvkem zobrazovacího zařízení je malý jednodeskový počítač Arduino. Zobrazovacím prvkem je matice tvořená z RGB LED diod. Tento maticový displej zobrazuje ikony, které indikují jednotlivá upozornění. Pro komunikaci mezi tímto zařízením a aplikací se využívá bezdrátová technologie Bluetooth Low Energy (dále jen BLE), proto musí zařízení obsahovat BLE modul.

V aplikaci má uživatel možnost specifikovat, jaká upozornění se budou zobrazovat a těm nastavit libovolnou ikonu, která bude promítána na maticovém displeji zařízení. Mezi zachytávaná upozornění patří příchozí hovory, stav baterie nebo notifikace nainstalovaných aplikací. K zachytávání těchto událostí využívá aplikace komponenty *BroadcastReceiver* a *NotificationListenerService* (viz kapitola 2.2). Pro dosažení perzistentního spojení k zařízení je součástí aplikace i služba na pozadí, která zůstane spuštěna i v případě

vypnutí aplikace (respektive všech aktivit aplikace).

Kromě návrhu a implementace tohoto vestavěného sytému bude v tomto článku i zmínka o testování na různých modelech telefonů a verzí systému Android.

## 2. Funkce systému Android

### 2.1 Bezdrátová komunikace

Je zřejmé, že pro dlouhodobě fungující aplikaci je vhodné využít komunikační technologii, která je zaměřena na úsporu baterie. Jednou z těchto technologií je NFC (Near Field Communication), která umožňuje oboustranné zasílání dat mezi zařízeními s krátkým dosahem (4 cm). Na platformě Android se můžeme s touto technologií setkat od verze 4.0, kde je označována jako funkce Android Beam [1]. Zasílání dat pomocí Android Beam se aktivuje přiložením dvou telefonů s podporou NFC zády k sobě. Teprve po té se na zařízení, ze kterého chceme odesílat data, objeví výzva „Tap to beam“. Klepnutím na obrazovku se data začnou přenášet do cílového zařízení. Není tedy možné pomocí této technologie vytvářet automatizované systémy.

Druhou komunikační technologií zaměřenou na šetření baterie je BLE. To je zcela odlišné od klasického Bluetooth a nejsou kompatibilní. Proto je nutné zkontrolovat, zda smartphone tuto funkci umožňuje. Mezi základní prvky architektury BLE patří ATT (Attribute Protocol) a jeho implementační rozšíření GATT (Generic ATT) [2]. ATT je jednoduchý bezstavový protokol klient/server. Každý server má data organizovaná ve formě atributů a každému z nich je přiřazeno identifikační číslo UUID (Universally Unique Identifer). Jednotlivé identifikátory určují typ přenášených dat. Tyto UUID jsou vysílány serverem za účelem poskytnutí informací o svých nabízených atributech (typech dat). GATT přidává abstrakční model atributů. Hlavní logická jednotka protokolu se nazývá služba. Každá služba se váže ke konkrétní funkcionalitě zařízení a skládá se z charakteristik, které určují různé hodnoty služby. Příkladem služby může být *Monitor srdečního tepu* obsahující charakteristiku *Frekvence srdečního tepu*. Výchozí délka paketu přenášených zpráv (MTU – Maximum Transmission Unit) je nastavena na 20 bajtů. V některých případech lze MTU nastavit na vyšší hodnotu, ovšem za předpokladu, že to hardware umožňuje.

### 2.2 Zachytávání systémových událostí

Na operačním systému Android existuje mnoho způsobů, jak lze sledovat nejrůznější události, které mohou na zařízení nastat [3].

*BroadcastReceiver* (dále jen receiver) je komponenta, která slouží k přijímání broadcast zpráv zasílaných ostatními částmi aplikace, jinými aplikacemi nebo systémem samotným. Při registraci receiverů se určuje, jakým typům broadcast zpráv budou naslouchat. Registrace může být buď statická, nebo dynamická. Statická registrace se provádí v manifestačním souboru *AndroidManifest.xml*[1]. Tyto receivery je možné spouštět i v případě, že je aplikace vypnutá. Když systém vytvoří nějakou událost, které staticky registrovaný receiver naslouchá, vytvoří se jeho instance a příslušné operace se provedou na pozadí. Oproti tomu dynamické registrování se provádí přímo v kódu aplikace nejčastěji pak v aktivitách nebo službách. Výhodou oproti statické registraci je možnost odregistrace receiveru v případě, že již není potřebný.

*NotificationListenerService* [4] (dále jen NLS) je systémový nástroj, umožňující aplikacím zachytávat informace o všech příchozích notifikacích v zařízení. Při nově příchozím upozornění je službě prostřednictvím systému zaslán objekt, zapouzdřující veškeré informace, které jsou prezentovány uživateli ve stavovém řádku systému. Jedná se například o název balíčku, nadpis a text notifikace, čas zobrazení, případně ikonu. Na rozdíl klasických oprávnění automaticky vyžadovaných prostřednictvím systémových dialogových oken je pro NLS zavedena zvláštní část v nastavení systému. Jedná se o zvláštní opatření, které zabraňuje úniku citlivých údajů, které mohou notifikace poskytovat (například zprávy SMS). Toto může být pro běžného uživatele velice nepohodlné, ale naštěstí je možné toto zjednodušit pomocí odkazu v aplikaci, který uživatele do nastavení přímo nasměruje.

### 2.3 Služby

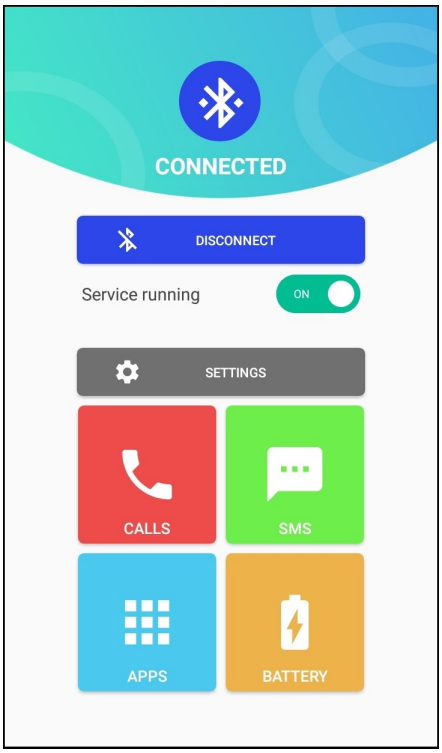
Služba [5] je komponenta aplikace, která umožňuje provádět libovolné operace na pozadí bez nutnosti interakce s uživatelem. Speciálním typem služby je tzv. *ForegroundService*. Tento typ je dle oficiální dokumentace doporučeno používat vždy, když aplikace provádí nějakou dlouhodobou operaci, o které by měl uživatel vědět (indikováno nesmazatelnou notifikací ve stavovém řádku systému). Důležitou metodou, kterou služba může implementovat, je *onStartCommand()*. Ta vrací jednu z konstant, které určují, jestli se služba restartuje, když ji systém ukončí kvůli nedostatku prostředků.

[1] Tento soubor musí obsahovat každý projekt ve svém kořenovém adresáři. Poskytuje základní informace o aplikaci samotnému systému. Obsahuje seznam všech aktivit a služeb, ale také i vyžadovaná oprávnění.

## 3. Návrh aplikace

U grafické stránky aplikace je kladen důraz především na jednoduchost a srozumitelnost a aby byly veškeré prvky co nejvíce intuitivní. Uživatel bude ve většině případů aplikaci používat pro připojování k notifikačnímu zařízení. Zbylé procento vykonaných akcí bude nastavování, jak se má tento systém chovat.

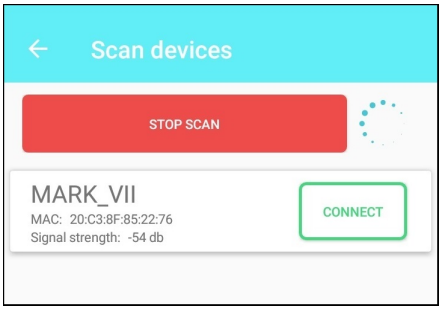
Nejvyužívanější částí aplikace bude hlavní aktivita[2] (viz obrázek 1). V této aktivitě se uživatelé připojují k zařízení. V horní části obrazovky je zobrazen stav připojení (*Připojeno*/*Připojování*/*Odpojeno*). Dále aktivita obsahuje odkazy do dalších aktivit, které slouží k nastavování upozornění. Uživatel může nastavovat upozornění pro příchozí hovory, stav baterie a notifikace nainstalovaných aplikací na telefonu. Dále si může uživatel určit jas LED diod (tlačítko *Settings*).




**Obrázek 1.** Hlavní aktivita aplikace, ve které se uživatel bude připojovat k zařízení. Obsahuje odkazy do aktivit pro nastavení upozornění.

Seznam nainstalovaných aplikací se generuje pomocí *PackageManageru*, který získává informace o nainstalovaných aplikacích. Ke každé aplikaci ukládá nastavení do lokální databáze SQLite[3]. Uživatel také může nastavit pro jednotlivá upozornění (i jednotlivé aplikace), zda je chce na zařízení zobrazovat.

Před samotným zahájením komunikace je zapotřebí se k zařízení připojit. Nejprve musí aplikace najít všechna dostupná zařízení pomocí skenování. Ta zobrazí uživateli do seznamu seřazeného dle síly signálu (viz obrázek 2). Uživatel si následně vybere, ke kterému zařízení se chce připojit. Pro komunikaci se zařízením využívá speciální formát zpráv. Tento protokol umí rozlišovat jednotlivé typy upozornění. Důvod pro toto rozdělení je především kvůli prioritám těchto typů upozornění. Například příchozí hovory se budou na displeji zobrazovat tak dlouho, jak bude vyzvánět telefon. Notifikace aplikací se budou zobrazovat do doby, než budou smazány ze stavového řádku telefonu.




**Obrázek 2.** Aktivita pro skenování zobrazující dostupná zařízení seřazené sestupně podle síly signálu. Uživatel může vybírat, které z nich chce používat.

## 4. Návrh zařízení

Zařízení se skládá ze tří částí – klon Arduino UNO, BLE modul a maticový RGB LED displej. U výběru komponent bylo zapotřebí si dávat pozor na spotřebu obou modulů, protože maximální výstupní proud desky Arduino je 500 mA při napájení z USB. V případě Bluetooth modulu to nebyl žádný problém, jelikož jeho odběr činí maximálně 400 $\mu$A. Ovšem odběr vybraného maticového modulu může dosahovat až 6 A – displej se skládá z 10 $\times$ 10 LED diod typu *WS2812B* a každá z nich při nejvyšším jasu odebírá 60 mA (každá barevná složka RGB 20 mA). Při průběžném testování diod jsem ale zjistil, že maximální jas je velmi oslnivý a pro oči nepříjemný. Maximální jas, který půjde nastavit bude zhruba 7,8%[4] maximálního možného jasu LED diod a tedy výsledný odběr modulu bude přibližně 468 mA. Pro zjednodušení práce těmito diodami navrhla společnost Adafruit Industries knihovnu *NeoPixels* [7]. Pomocí PWM (pulzně šířkové modulace) umožňuje jednotlivým diodám snadno měnit barvu i jas.

Datový vodič pro LED diody musí být zapojen do pinu, který umožňuje PWM (viz obrázek 3). Pro

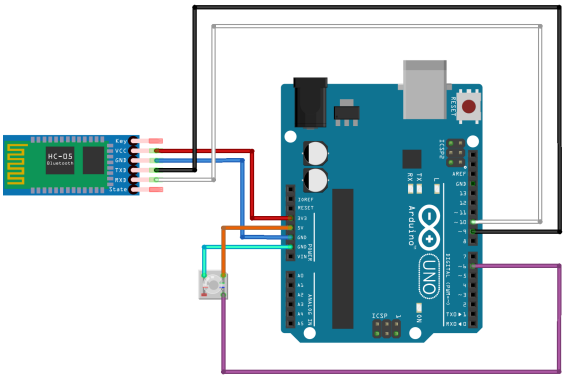
[2] Aktivity umožňují uživatelům skrze grafické rozhraní ovládat aplikaci a komunikovat s ní.

[3] SQLite je relativně malá, výkonná multiplatformní knihovna. Samotná data jsou uložena v lokálním souboru, ke kterým se přistupuje za pomoci klientské aplikace [6].

[4] Maximální hodnota jasu, která lze za pomocí knihovny *NeoPixels* nastavit je 255, v případě tohoto zařízení je hodnota nastavena na 20

datové přenosy mezi BLE modulem a Arduinem se využívá sériové komunikace, proto datové vodiče mohou být připojeny na jakékoliv digitální piny desky Arduino.

Pro zařízení bude také vytištěné pouzdro, které je prozatím ve fázi návrhu.



**Obrázek 3.** Diagram zapojení modulů k desce Arduino. Maticový modul je zde reprezentován jednou diodou. Datový vodič diody je připojen na pin 6, který umožňuje PWM. Datové vodiče BLE modulu jsou zapojeny na piny 9 a 10. Zbylé vodiče slouží pro napájení modulů.

# 5. Implementace

V této části článku se zaměřím na realizaci nejdůležitějších částí aplikace a zařízení.

## 5.1 Služby na pozadí

Aplikace obsahuje dvě služby. První z nich je využita pro komunikaci se zařízením pomocí Bluetooth. Přijímá broadcast zprávy od ostatních komponent aplikace a provádí určité akce Jedná se například připojení se k zařízení nebo odeslání zprávy na zařízení po té, co byla zachycena některá z událostí. Tato služba je spouštěna jako *ForegroundService*. Její implementace obsahuje metodu *onStartCommand()*. Aby služba zůstala běžet i v případě ukončení aplikace uživatelem, musí tato metoda vracet konstantu `START_STICKY`, která oznamuje systému, aby byla služba vždy restartována [5].

Druhá služba fungující na pozadí je potomek třídy *NotificationListenerService*. Jejím jediným úkolem je zasílat broadcast zprávy ohledně zachycení přidané nebo smazané notifikace výše zmíněné službě pro komunikaci. Ve zprávě se posílá název balíčku aplikace, která notifikaci vytvořila, a informaci, zda byla vytvořena nová nebo odebrána již existující.

Řešení problémů u testovaných zařízení je popsáno v kapitole 6.1.

## 5.2 Zachytávání událostí

Zachytávání notifikací, jak již bylo zmíněno, probíhá pomocí služby *NotificationListenerService*. Pro získávání stavu baterie a zachytávání příchozích hovorů jsou využity staticky registrované *BroadcastReceivery*. Přístup k informacím o příchozích hovorech je limitován na aplikace, kterým uživatel přidělil oprávnění, jelikož se jedná o citlivé údaje. Toto oprávnění se uvádí v manifestačním souboru a je pak vyžadováno po uživateli při spuštění aplikace pomocí systémového dialogového okna.

## 5.3 Komunikace přes BLE

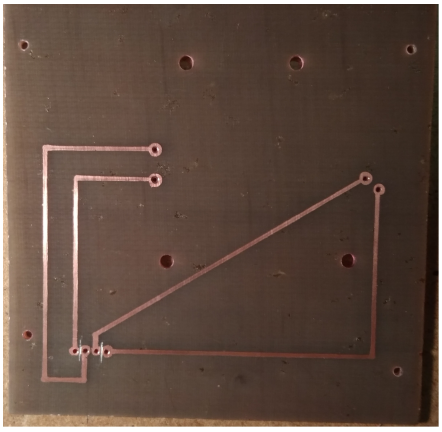
Před zahájením komunikace je nutné zjistit všechna dostupná zařízení v okolí pomocí skenování. Aby však aplikace našla pouze zařízení pro zobrazování notifikací, bude nejefektivnější změnit BLE modulům jejich názvy. Aplikace pak při skenování bude ignorovat veškerá zařízení s rozdílným názvem – tzv. *filtrované skenování*. Díky skenování pak aplikace může zjistit MAC adresu zvoleného zařízení (modulu), která je využitá pro připojení.

Jakmile se podaří úspěšně k zařízení připojit (indikováno pomocí callbacku), pokusí se aplikace najít službu a její charakteristiku, která bude využívána pro přenos dat na zařízení. Využitý BLE modul poskytuje službu pro sériovou komunikaci a charakteristiku pro zasílání dat na modul. Služba i charakteristika jsou pro všechny moduly tohoto typu identifikovány pomocí stejných *UUID*, proto je možné tyto hodnoty „natvrdo“ napsat do kódu aplikace. Nejprve potřeba pomocí *UUID* nalézt danou službu a až díky ní její charakteristiku. Po nalezení charakteristiky pomocí metody *BluetoothGattService.getCharacteristic(UUID uuid)*, vrátí tato metoda objekt, který se bude využívat pro zasílání dat na zařízení.

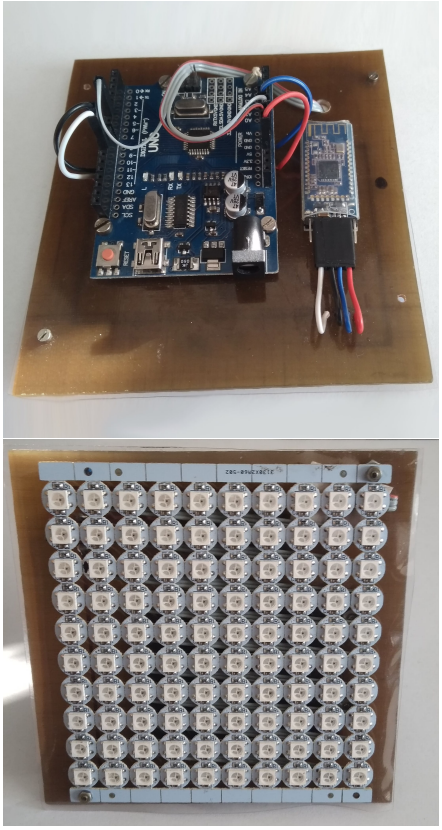
Jak jsem uvedl v kapitole 2.1, výchozí velikost jednotlivých packetů v případě technologie BLE je 20 bajtů a ne všechen hardware podporuje zvýšení MTU (včetně telefonů) [8]. Dle dokumentace mnou vybraný modul tuto možnost neposkytuje. Také nelze předpokládat, že každý telefon, na kterém aplikace bude běžet, bude umožňovat změnu MTU. Proto jsem se rozhodl vždy posílat data po packetech délky 20 bajtů. Po odeslání dat na zařízení, je zavolán callback označující stav přenosu dat (metoda *onWriteCharacteristics()*). Pokud přenos packetu proběhl v pořádku, odešle aplikace následujících 20 bajtů zprávy, v případě neúspěšného přenosu se pokusí odeslat packet znovu. Protože může nastat situace, že přijde nová notifikace v době, kdy již probíhá přenos dat, použil jsem semafor, který vždy pozastaví nově vytvořené vlákno do doby, než budou data odeslána.

## 5.4 Základní deska a finální zapojení

Kvůli jednoduchému přenášení zařízení jsem umístil všechny moduly na jednu základní desku vytvořenou z cuprextitu, jež obsahuje plošné spoje (viz obrázek 4). Tyto spoje slouží k přehlednému propojení mezi komponenty. Propojení plošných spojů a komponent je tvořeno drátovými propojkami (viz obrázek 5).



**Obrázek 4.** Základní deska s plošnými spoji propojující jednotlivé moduly zařízení.



**Obrázek 5.** Nahoře: zadní strana základní desky s připojenými moduly k desce Arduino. Dole: na přední straně základní desky je připevněn maticový RGB LED displej.

# 6. Testování

## 6.1 Testování kompatibility

Testování doposud proběhlo na osmi různých zařízeních s různými verzemi operačního systému Android:

- Xiaomi Redmi 4a (Android 7.1)
- Xiaomi Redmi 4a (Android 6.0)
- Xiaomi Redmi 7a (Android 9.0)
- Xiaomi Redmi Note 5 (Android 9.0)
- Doogee X5 Pro (Android 5.1)
- Zopo C2 (Android 6.0)
- Samsung Galaxy J5 2016 (Android 7.1)
- Lenovo A2010 (Android 5.1)

Důvod proč jsem pro testování vybral ve většině smartphony vyrobené v Číně (*Xiaomi*, *Doogee*, *Zopo*), je ten, že většina nadstaveb na Android u těchto telefonů obsahuje sytém šetření baterie, který automaticky vypíná všechny procesy běžící na pozadí v případě ukončení aktivit aplikace [9]. Tím pádem není možné ponechat službu typu *ForegroudService* běžet bez aktivit, pokud nemá uživatelem přidělená patřičná oprávnění. Stejně je na tom i NLS, která je od aplikace odpojena, a tedy nemůže zasílat aplikaci informace o zachytávaných notifikacích.

Aby mohla služba zůstat nepřetržitě spuštěná, je nutné v případě telefonů značky *Xiaomi* přidělit aplikaci oprávnění k automatickému spouštění. Do kódu aplikace jsem přidal odkaz do tohoto nastavení podobně jako v případě NLS. Tento odkaz může být použit pouze u telefonů této značky, čehož je docíleno pomocí třídy *Build*, která zpřístupňuje informace o Android zařízení, na kterém je aplikace právě spuštěna. Bohužel v případě smartphonu *Xiaomi Redmi 4a* s verzí Androidu 7.1 se mi nepodařilo, aby služba na popředí fungovala, když uživatel aplikaci vypne, a to ani po udělení oprávnění. Stejně tak přestane fungovat i NLS. Takovéto chování je nejspíše způsobené chybou dané verze operačního systému (případně nadstavby). U ostatních modelů *Xiaomi* fungovaly služby na pozadí dle očekávání po udělení práv.

V případě telefonu *Doogee X5 Pro* se mi opět nepodařilo, aby *ForegroundService* fungovala i bez zapnuté aplikace. Také jsem nenašel v nastavení žádná oprávnění k automatickému spouštění jako tomu bylo v případě telefonů *Xiaomi*. Paradoxně však služba NLS zůstala vždy spuštěná (otestováno ladícími zprávami v Android Studiu).

Dalším problémem, na který jsem v průběhu testování narazil, byl u telefonu *Zopo C2*. Na tomto zařízení nebyla aplikace schopna najít služby a charakteristiky BLE modulu hned po ustavení spojení. Efektivním

řešením bylo začít hledat služby (a tedy charakteristiky) až po uplynutí určité doby po ustavení spojení s modulem [10]. V aplikaci jsem nastavil čekání na 1 s.

U zbylých testovaných zařízeních nebyly nalezeny žádné další problémy.

### 6.2 Testování uživatelů

V této části textu se zmíním o testování layoutu uživateli a jeho následné úpravě, a také o požadavcích uživatelů na přidání nových funkcí do aplikace/zařízení.

Největším problémem původního návrhu grafického uživatelského rozhraní bylo, že uživatelé nevěděli, jak se k zařízení připojit. Ikona, která nyní indikuje stav připojení k zařízení, fungovala původně i jako tlačítko pro manuální připojení/odpojení se k/od zařízení. Toto řešení se ukázalo jako nepoužitelné, protože žádný z testovaných uživatelů nebyl schopen poznat, že se jedná o tlačítko. Proto jsem nakonec do layoutu přidal pro tyto účely separátní tlačítko, jak je zobrazeno v konečném návrhu.

Druhá změna GUI se týkala výběru číselných hodnot v případě nastavení priority notifikací a procentuálního stavu baterie, při kterém se pošle upozornění na zařízení. Původní komponenta pro výběr čísel byla *NumberPicker*, která se zobrazovala v dialogovém okně. Dle některých testerů by bylo lepší spíše využít posuvník (*SeekBar*), který nakonec v konečném návrhu layoutu zůstal, protože je na první pohled srozumitelnější.

Některým uživatelům přišlo nepohodlné a zdlouhavé hledání aplikace pro SMS zprávy mezi ostatními systémovými aplikacemi, proto jsem přidal do hlavní aktivity odkaz v podobě tlačítka.

Jedním z požadavků na funkcionalitu celkového systému bylo, aby si uživatelé mohli vyzkoušet vzhled ikony, která se bude zobrazovat na maticovém displeji. Proto jsem do aktivit pro nastavení upozornění přidal tlačítko, které odešle ikonu daného upozornění na zařízení, kde bude zobrazena.

Dalším žádaným rozšířením byla možnost upravit jas LED diod, protože se některým uživatelům stále připadaly diody velice jasné, i přes to, že byly sníženy na téměř 8% své maximální svítivosti.

Po úpravách dle výše zmíněných nedostatků a požadavků proběhlo druhé testování. Všichni uživatelé již byli schopni se bez pomoci na zařízení připojit. Dále také přidání odkazu na nastavení SMS aplikace byla velmi vítanou změnou. Největší ohlasy u uživatelů však získala možnost vyzkoušet vzhled jimi nastavených ikon.

Testování proběhlo veskrze pozitivně, protože pomohlo odhalit nedostatky a zároveň přineslo některá vylepšení pro celý tento vestavěný systém.

## 7. Závěr

Cílem této práce bylo navrhnout a realizovat zařízení zobrazující notifikace včetně aplikace pro telefony, která tato upozornění zachytává a zasílá na zařízení prostřednictvím Bluetooth Low Energy.

Aplikace byla testována na různých modelech chytrých telefonů s různými verzemi operačního systémů Android. Některé chyby, které byly způsobeny ať už právě konkrétní verzí operačního systému, nadstavby nebo hardwarovou implementací telefonu, byly ošetřeny.

Toto zařízení by mohlo být používáno v místech, kde by zvukové notifikace byly obtíží jako například kanceláře (samozřejmě by záleželo na jednotlivcích). Dalšími možnými uživateli by mohli být sluchově postižení, kteří jsou závislí na vizuálních podnětech. Nebo by jej mohli využívat techničtí nadšenci. Navíc díky rozšíření kompatibility napříč různými modely telefonů s operačním systémem Android může být toto zařízení používáno větším množstvím uživatelů.

Další vývoj zařízení by mohl zahrnovat podrobnější testování aplikace na vícero mobilních zařízeních a umožnit tak vícero uživatelům využívat tento vestavěný systém. Dále by se mohlo experimentovat s různými variantami zobrazování upozornění například využití LCD displejů.

## Poděkování

Rád bych poděkoval mému vedoucímu bakalářské práce prof. Ing. Adamu Heroutovi, Ph.D. za odborné vedení, cenné rady a připomínky při konzultacích. Dále bych rád poděkoval všem, kteří se podíleli na testování a za jejich zpětnou vazbu a především pak Jiřímu Polákovi za ochotu při tisknutí pouzdra pro hardwarové zařízení.

## Literatura

[1] Nathan Chandler. What is android beam? how stuff works? blogpost (english), Feb 2012. https://electronics.howstuffworks.com/android-beam1.htm.

[2] Kevin Townsend, Cufí Carles, Akiba, and Robert Davidson. *Getting started with Bluetooth low energy*. OReilly Media, 2014.

[3] Jan Piskáček. Možnosti sledování událostí na systému android a jejich využití pro odhalování podezřelého chování aplikací. Bakalářská práce, České vysoké učení technické v Praze, Fakulta elektrotechnická, 2015. https://dspace.cvut.cz/bitstream/handle/10467/

61674/F3-BP-2015-Piskacek-Jan-piskaja2_2015_bp.pdf.

[4] Inc. Google. Notificationlistenerservice. online (english), 2020. https://developer.android.com/reference/android/service/notification/NotificationListenerService.

[5] Inc. Google. Services overview. online (english), 2020. https://developer.android.com/guide/components/services.

[6] Ľuboslav Lacko. *Mistrovství – Android*. Computer Press, 2017.

[7] Adafruit Industries. Some finer points of neopixels. online (english), 2020. https://learn.adafruit.com/sipping-power-with-neopixels/insights.

[8] Inc. GitHub. Mtu limit of 20 bytes on some android devices. blogpost (english), May 2016. https://github.com/don/cordova-plugin-ble-central/issues/234.

[9] Reddit. Workmanager reliability for periodic tasks on chinese roms (xiaomi, huawei, and so on). blogpost (english), Oct 2018. https://www.reddit.com/r/androiddev/comments/9ra0iq/workmanager_reliability_for_periodic_tasks_on/.

[10] Stack Overflow. onservicesdiscovered never called while connecting to gatt server. blogpost (english), Jan 2017. https://stackoverflow.com/questions/41434555/onservicesdiscovered-never-called-while-connecting-to-gatt-server.

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

# EnzymeMiner: Web Server for Automated Mining of Soluble Enzymes

Simeon Borko*

**Abstract**
Millions of protein sequences are being discovered at an incredible pace, representing an inexhaustible source of biocatalysts. Despite genomic databases growing exponentially, classical biochemical characterization techniques are time-demanding, cost-ineffective and low-throughput. Therefore, computational methods are being developed to explore the unmapped sequence space efficiently. Selection of putative enzymes for biochemical characterization based on rational and robust analysis of all available sequences remains an unsolved problem. To address this challenge, I have developed EnzymeMiner – a web server for automated screening and annotation of enzymes that enables selection of hits for wet-lab experiments. EnzymeMiner prioritizes sequences that are more likely to preserve the catalytic activity and are expressible in a soluble form in heterologous host organism *Escherichia coli*. EnzymeMiner reduces the time devoted to data gathering, multi-step analysis, sequence prioritization and selection from days to hours. EnzymeMiner is a universal tool applicable to any enzyme family that provides an interactive and easy-to-use web interface freely available at https://loschmidt.chemi.muni.cz/enzymeminer/.

**Keywords:** enzyme mining — novel biocatalysts — web server

**Supplementary Material:** *N/A*

*xborko02@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

## 1. Introduction

There are currently more than 259 million unique protein sequences in the NCBI nr [1] database (release 2020-02-10). Despite their enormous promise for biological and biotechnological discovery, experimental characterization has been performed on only a small fraction of the available sequences. Currently, there are about 560,000 protein sequences reliably curated in the UniProtKB/Swiss-Prot [2] database (release 2020_01).

The low ratio of characterized to uncharacterized sequences reflects the sharp contrast in low-throughput biochemical techniques versus high-throughput next-generation sequencing technology. Although more efficient biochemical techniques employing miniaturization and automation have been developed [3, 4, 5], the most widely used experimental methods do not provide sufficient capacity for biochemical characterization of proteins spanning the ever-increasing sequence space. Therefore, computational methods are currently the only way to explore the immense protein diversity available among the millions of uncharacterized sequence entries.

Two different computational strategies are generally used for exploration of the unknown sequence space. The first strategy takes a novel uncharacterized sequence as input and predicts functional annotations.

The method involves annotating the unknown input sequences by predicting protein domains [6], Enzyme Commission (EC) number [7] or Gene Ontology terms that are a subject of the initiative named the Critical Assessment of Functional Annotation [8]. These methods are often universal and applicable to any protein sequence. However, they often lack specificity as the automatic annotation rules or statistical models need to be substantially general. A significant advantage of these methods is their seamless integration into available databases. Submission of a query sequence to a database is sufficient, with no need for running computation- and memory-intensive bioinformatics pipelines locally. A model example of this approach is the automatic annotation workflow of the UniProtKB/TrEMBL database [2].

The second strategy takes a well-known characterized sequence as an input and applies a computational workflow, typically based on a homology search, to identify novel uncharacterized entries in genomic databases that are related to the input query sequence [5, 9]. The homology search is often followed by a filtration step, which checks the essential sequence properties, e.g., domain structure or presence of essential residues. The main advantage of these methods is the higher specificity of the analysis. A disadvantage is that it may be complicated to apply the developed workflow to protein families other than those for which it was designed. Moreover, these workflows typically require running complex bioinformatics pipelines and are usually not available through a web interface.

The fundamental unsolved problem is how to deal with the overwhelming number of sequence entries identified by these methods and select a small number of relevant hits for in-depth experimental characterization. For example, a database search for members of the haloalkane dehalogenase model family using the UniProt web interface yields 3,598 sequences (UniProtKB release 2020_01). It is impossible to rationally select several tens of targets for experimental testing without additional bioinformatics analyses to help prioritize such a large pool of sequences.

To address the challenge of exploring the unmapped enzyme sequence space and rational selection of attractive targets, I have developed the EnzymeMiner web server. Enzymes are specific proteins that serve as biocatalysts for chemical reactions. They have broad application in both industry and academy, mostly in biosynthetic pathways for ecological production of chemical compounds. EnzymeMiner identifies novel enzyme family members, comprehensively annotates the targets and facilitates efficient prioritization and selection of representative hits for experimental characterization. To the best of my knowledge, there is currently no other tool available that allows such a comprehensive analysis in a single easy-to-run integrated workflow on the web.

## 2. Method

EnzymeMiner implements a three-step workflow: (i) homology search, (ii) essential residue based filtering, and (iii) hits annotation (Figure 1). To execute these tasks, the server requires two different types of input information: (i) query sequences, and (ii) essential residue templates. The query sequences serve as seeds for the initial homology search. The essential residue templates, defined as pairs of a protein sequence and a set of essential residues in that sequence, allow the server to prioritize hits that are more likely to display the enzyme function. Therefore, the essential residues may be the catalytic and ligand- or cofactor-binding residues that are indispensable for proper catalytic function. Each essential residue is defined by its name, position and a set of allowed amino acids for that position.

In the first homology search step, a query sequence is used as a query for a PSI-BLAST [10] two-iteration search in the NCBI nr database [1]. If more than one query sequence is provided, a search is conducted for each sequence separately. Besides a minimum E-value threshold 10-20, the PSI-BLAST hits must share a minimum of 25% global sequence identity with at least one of the query sequences. Artificial protein sequences, i.e., sequences described by the term artificial, synthetic construct, vector, vaccinia virus, plasmid, halotag or replicon, are removed. EnzymeMiner sorts the PSI-BLAST hits by E-value and passes a maximum of 10,000 best hits to the next steps in the workflow. The default parameters for the homology search step, as well as the other steps, can be modified using advanced options in the web server.

In the second essential residue based filtering step, the homology search hits are filtered using the essential residue templates. First, the hits are divided into template clusters. Each cluster contains all hits matching essential residues of a particular template. Essential residues are checked using global pairwise alignment with the template calculated by USEARCH [11]. When multiple essential residue templates match, the hit is assigned to the template with the highest global sequence identity. Second, for each cluster, an initial multiple sequence alignment (MSA) is constructed using Clustal Omega [12]. The MSA is used to revalidate the essential residues of identified hits

by checking the corresponding column in the MSA. Sequences not matching essential residues of the template are removed from the cluster. Third, the MSA is constructed again for each template cluster and the essential residues are checked for the last time. The final set of identified sequences reported by EnzymeMiner contains all sequences left in the template clusters. In the third annotation step, the identified sequences are annotated using several databases and predictors: (i) transmembrane regions are predicted by TMHMM [13], (ii) Pfam domains are predicted by InterProScan [14], (iii) source organism annotation is extracted from the NCBI Taxonomy [15] and the NCBI BioProject database [16], (iv) protein solubility is predicted by the in-house tool SoluProt for prediction of soluble protein expression in *Escherichia coli*, and (v) sequence identities to queries, hits or other optional sequences are calculated by USEARCH [11].

The sequence space of the identified hits is visualized using representative sequence similarity networks (SSNs) generated at various clustering thresholds using MMseqs2 [17] and Cytoscape [18]. SSNs provide a clean visual approach to identify clusters of highly similar sequences and rapidly spot sequence outliers. SSNs proved to facilitate identification of previously unexplored sequence and function space [19]. The SSN generation method used in EnzymeMiner is inspired by the EFI-EST tool [20]. The minimum alignment score to include an edge between two representative sequences in an SSN is 40.

## 3. Validation

The EnzymeMiner workflow has been experimentally validated using the model enzymes haloalkane dehalogenases [5]. The sequence-based search identified 658 putative dehalogenases. The subsequent analysis prioritized and selected 20 candidate genes for exploration of their protein structural and functional diversity. The selected enzymes originated from genetically unrelated Bacteria, Eukarya and, for the first time, also Archaea and showed novel catalytic properties and stabilities. The workflow helped to identify novel and very interesting haloalkane dehalogenases, including (i) the most catalytically efficient, (ii) the most thermostable enzyme showing a melting temperature of 71 °C, (iii) three different cold-adapted enzymes active at near to 0 °C, (iv) highly enantioselective enzymes, (v) enzymes with a wide range of optimal operational temperature from 20–70 °C and an unusually broad pH range from 5.7–10, and (vi) biocatalysts degrading the warfare chemical yperite or different environmental pollutants. The sequence mining, annotation, and visualization steps from the workflow published by Vanacek and co-workers [5] were fully automated in the EnzymeMiner web server. The successful use case for the haloalkane dehalogenase family is described in the form of an easy-to-follow tutorial available on the EnzymeMiner web page. Additional extensive validation of the fully automated version of EnzymeMiner, experimentally testing the properties of another 50 genes of the haloalkane dehalogenases, is currently ongoing in the Loschmidt laboratories.

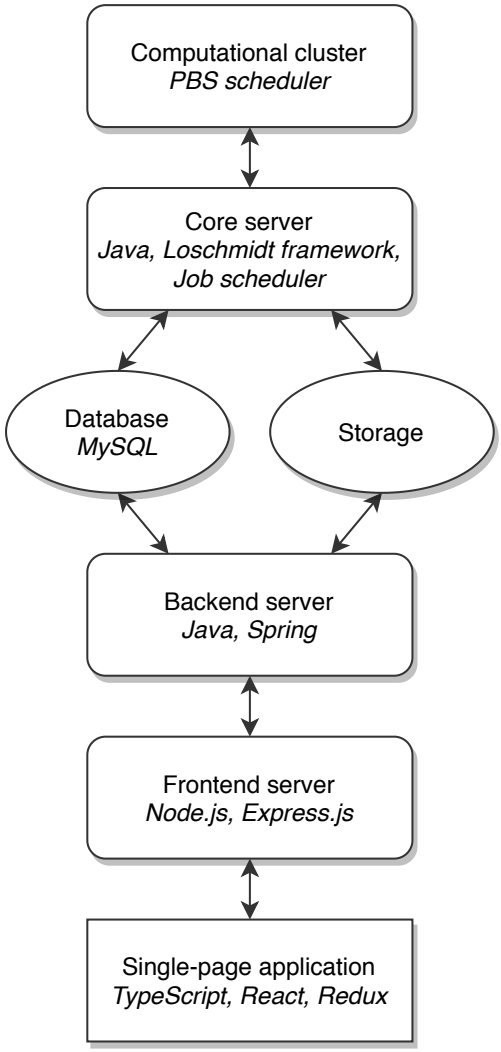



**Figure 1.** Multiple-server architecture of the EnzymeMiner web server. Only the *Frontend* server is publicly accessible.

## 4. Implementation

As many steps of the workflow are computationally demanding, a special architecture of the web server is needed. EnzymeMiner has a layered multiple-server architecture, where each server has a specific purpose and responsibility (Figure 1).

The *Core* server implements the main EnzymeMiner workflow using an in-house Java framework for

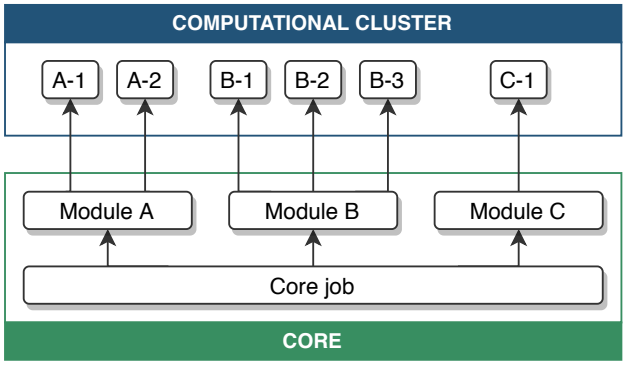



**Figure 2.** EnzymeMiner computing units. There are three types of computing units: (i) core jobs, (ii) core modules and (iii) cluster jobs.

bioinformatic applications from Loschmidt Laboratories. The *Core* reads jobs submitted to the *Database* and corresponding input files from the *Storage*. For each job, the *Core* instantiates and schedules twelve core modules that implement the three-step workflow. Independent modules are executed in parallel to speed up the calculation. To compute extremely demanding tasks, core modules submit cluster jobs to the *Computational cluster*. The cluster uses the Portable Batch System (PBS) to maximize parallelism and optimize usage of the available computing resources. To summarize, three types of computing units (Figure 2) are used in EnzymeMiner: (i) user-submitted core job, (ii) core modules scheduled by the *Core* server, and (iii) cluster jobs scheduled by the *Computational cluster*.

The *Backend* server implements a REST API for the single-page application. It implements endpoints to (i) submit new jobs to the *Database* and save input files to the *Storage*, (ii) download the job status and job results, and (iii) fetch server usage statistics.

The *Frontend* serves the static files of the single-page application. It is the only server in the architecture that is publicly accessible. The *Frontend* server also serves as a proxy to the *Backend* server.

The *Single-page application* provides the graphical user interface for the EnzymeMiner workflow. It is running in the user's browser and it is implemented in TypeScript using React framework for HTML rendering and Redux library for state management. The application uses Bootstrap and Material-UI components. Webpack builds the JavaScript and CSS files for the browser.

## 5. Conclusions and outlook

The EnzymeMiner web server identifies putative members of enzyme families and facilitates their prioritization and well-informed selection for experimental characterization to reveal novel biocatalysts. The identified enzymes might have broad application in both industry and academy, mostly in biosynthetic pathways for ecological production of chemical compounds. Such a task is difficult using the web interfaces of the available protein databases, e.g., UniProtKB/TrEMBL and NCBI Protein, since additional analyses are often required. The output of EnzymeMiner is an interactive selection table containing the annotated sequences divided into sheets based on various criteria. The table helps to select a diverse set of sequences for experimental characterization. Two key annotations are (i) the predicted solubility score, which can be used to prioritize the identified sequences and increase the chance of finding enzymes with soluble protein expression, and (ii) the sequence identity to query sequences complemented with an interactive SSN displayed directly on the web, which can be used to find diverse sequences. EnzymeMiner is a universal tool applicable to any enzyme family. It reduces the time needed for data gathering, multi-step analysis and sequence prioritization from days to hours. All the EnzymeMiner features are implemented directly on the web server and no external tools are required. The fact that EnzymeMiner is a web application brings both advantages and disadvantages. User is able to run the pipeline using a user-friendly web interface without the need to install software and download databases. On the other hand, user cannot modify the pipeline and the speed of the computation depends on the current load of the computing cluster.

In the next EnzymeMiner version, I plan two major improvements. First, I will implement automated tertiary structure prediction based on homology modeling and threading for all identified sequences. The structural predictions will allow subsequent analysis of active site pockets/cavities and access tunnels that will help to identify additional attractive targets for experimental characterization. Second, I will implement automated periodical mining. When enabled, EnzymeMiner will rerun the analysis periodically and inform the user about novel sequences found since the last search.

EnzymeMiner web server was submitted to highly impacted *Nucleic Acids Research* journal in March 2020.

## 6. Acknowledgement

I thank Ing. Jiří Hon for consultations, guidance, support and valuable advice. I also thank the Loschmidt laboratories for their scientific supervision and the participants of the 1st Hands-on Computational Enzyme Design Course for giving valuable feedback on the En-

zymeMiner user interface. Their comments inspired me to make the sequence similarity network visualization much more interactive. Computational resources were supplied by the project "e-Infrastruktura CZ" (e-INFRA LM2018140) and ELIXIR (LM2015047) provided within the program Projects of Large Research, Development and Innovations Infrastructures.

## References

[1] Eric W. Sayers, Richa Agarwala, Evan E. Bolton, et al. Database resources of the National Center for Biotechnology Information. *Nucleic Acids Research*, 47(D1):D23–D28, January 2019.

[2] The UniProt Consortium. UniProt: a worldwide hub of protein knowledge. *Nucleic Acids Research*, 47(D1):D506–D515, January 2019.

[3] Pierre-Yves Colin, Balint Kintses, Fabrice Gielen, et al. Ultrahigh-throughput discovery of promiscuous enzymes by picodroplet functional metagenomics. *Nature Communications*, 6(1):1–12, December 2015.

[4] Thomas Beneyton, Stéphane Thomas, Andrew D. Griffiths, et al. Droplet-based microfluidic high-throughput screening of heterologous enzymes secreted by the yeast Yarrowia lipolytica. *Microbial Cell Factories*, 16(1):18, January 2017.

[5] Pavel Vanacek, Eva Sebestova, Petra Babkova, et al. Exploration of Enzyme Diversity by Integrating Bioinformatics with Expression Analysis and Biochemical Characterization. *ACS Catalysis*, 8(3):2402–2412, March 2018.

[6] Sara El-Gebali, Jaina Mistry, Alex Bateman, et al. The Pfam protein families database in 2019. *Nucleic Acids Research*, 47(D1):D427–D432, January 2019.

[7] Yu Li, Sheng Wang, Ramzan Umarov, et al. DEEPre: sequence-based enzyme EC number prediction by deep learning. *Bioinformatics*, 34(5):760–769, March 2018.

[8] Naihui Zhou, Yuxiang Jiang, Timothy R. Bergquist, et al. The CAFA challenge reports improved protein function prediction and new functional annotations for hundreds of genes through experimental screens. *Genome Biology*, 20(1):244, November 2019.

[9] Wai Shun Mak, Stephen Tran, Ryan Marcheschi, et al. Integrative genomic mining for enzyme function to enable engineering of a non-natural biosynthetic pathway. *Nature Communications*, 6(1):1–10, November 2015.

[10] S. Altschul. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–3402, September 1997.

[11] Robert C. Edgar. Search and clustering orders of magnitude faster than BLAST. *Bioinformatics*, 26(19):2460–2461, October 2010.

[12] Fabian Sievers, Andreas Wilm, David Dineen, et al. Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega. *Molecular Systems Biology*, 7(1):539, January 2011.

[13] Anders Krogh, Björn Larsson, Gunnar von Heijne, and Erik L.L Sonnhammer. Predicting transmembrane protein topology with a hidden markov model: application to complete genomes11Edited by F. Cohen. *Journal of Molecular Biology*, 305(3):567–580, January 2001.

[14] E. Quevillon, V. Silventoinen, S. Pillai, et al. InterProScan: protein domains identifier. *Nucleic Acids Research*, 33(Web Server):W116–W120, July 2005.

[15] S. Federhen. The NCBI Taxonomy database. *Nucleic Acids Research*, 40(D1):D136–D143, January 2012.

[16] T. Barrett, K. Clark, R. Gevorgyan, et al. BioProject and BioSample databases at NCBI: facilitating capture and organization of metadata. *Nucleic Acids Research*, 40(D1):D57–D63, January 2012.

[17] Martin Steinegger and Johannes Söding. MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. *Nature Biotechnology*, 35(11):1026–1028, November 2017.

[18] P. Shannon. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome Research*, 13(11):2498–2504, November 2003.

[19] Janine N. Copp, Eyal Akiva, Patricia C. Babbitt, and Nobuhiko Tokuriki. Revealing Unexplored Sequence-Function Space Using Sequence Similarity Networks. *Biochemistry*, 57(31):4651–4662, August 2018.

[20] John A. Gerlt, Jason T. Bouvier, Daniel B. Davidson, et al. Enzyme Function Initiative-Enzyme Similarity Tool (EFI-EST): A web tool for generating protein sequence similarity networks. *Biochimica et Biophysica Acta (BBA) - Proteins*

*and Proteomics*, 1854(8):1019–1037, August 2015.

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

# Behavior-Based Network Device Tracking

Michael Adam Polák*

**Abstract**
With the constantly growing number of devices on private and corporate networks, it is becoming increasingly more important for network administrators to track devices based on their behavior with limited feature availability due to the increasing security risks. This paper analyzes methods used to create device profiles that are subsequently used to identify devices using frequency analysis and the k-Nearest Neighbors algorithm with cosine similarity as the distance metric. Lastly, the results of this method are presented with possible improvements to the existing algorithm.

**Keywords:** Network Device Identification — Frequency Analysis — k-Nearest Neighbors

**Supplementary Material:** *N/A*

*xpolak31@fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

## 1. Introduction

With the constantly growing number of devices that are connected to the network, it is becoming increasingly more important for network administrators to be able to identify devices on the network. The root of this necessity lies in the escalating number of attacks that are performed per day. Therefore, methods that identify devices based on their behaviour are being developed to identify and verify whether the behaviour of a device is similar to the already existent user profile, or the device is malicious and the communication needs to be blocked. The goal of this paper is to propose an algorithm that provides administrators an automatized method to track network devices on the networks based on their behavior without the need of tracking any additional information about the devices.

This work explores the current state of the art solutions in Section 2 and proposes new approaches to tracking network devices based on their behavior, with better performance, using frequency analysis to create device profiles and the k-Nearest Neighbors algorithm with cosine similarity as the distance metric. Device profiles are created based on the usernames and source IP addresses, since MAC addresses that would uniquely identify devices are rarely available in a production environment. However, in the case of this paper the MAC addresses are always available and are used to measure the performance of the proposed algorithm (e.g., MAC addresses are used as labels). This combination of features (source IP address and username) to create device profiles has been chosen since it provides a very simple and effective way to distinguish devices in a given time-window. However, this combination of features is not sufficient to identify a specific device, since a user can own multiple devices and the source IP address of a given device may change over time, therefore, behavior based device tracking is necessary.

A further advantage of the proposed approach to device identification is that compared to the methods described in Section 2, it is not reliant on the availability of all of the information, such as visited URLs, and performs well even in the absence of these features. The absence of features is pretty common since network administrators have different goals with what information they need to track and maintain, therefore, robustness and the absence of the necessity to track any additional information is a welcome improvement compared to the existent methods.

## 2. State of the Art

In the past, methods such as HTTP or TLS fingerprinting have been used to reliably identify devices. However, the constantly increasing network traffic loads result in a significant increase of the size of the databases. Furthermore, it becomes difficult to

maintain these large databases up to date and accurate, since the values of the fingerprints change over time and these values need to be kept for each device and visited website separately [1].

In addition, research that has been done up to this point explores only methods that measure similarity with previously seen behaviour [2]. Kumpost's algorithm achieves an overall accuracy of 78.3%, however, has high spatial and time demands over large datasets, which renders these methods impractical in an environment with many network devices. Another disadvantage of his approach is that the aforementioned method works only on SSH traffic, where all of the features are always available, which is rarely the case in a production environment.

Another approach to tracking devices has been proposed in the dissertation thesis of Kohno et al. [3] using clock-skew to uniquely identify devices. Clock-skew relies on the uniqueness of crystal frequencies (deviations from the factory frequency are introduced due to manufacturing tolerances) that computers use to measure time, and measure the time measurement deviation from a base device. Kohno et al. proposed to estimate clock-skew by using the slope of the offset points, which is the difference from the beginning of the measurement (e.g., packet transmission) until the observer receives the packet. They have shown that the slope of the upper bound is similar to the slopes of the offsets. This approach, however, often requires synchronized sampling [4] to measure clock-skew, which is an active method to tracking devices.

## 3. Proposed Approach

The goal of this work is to propose an algorithm that will be accurate at identifying network devices and have low time and spacial requirements. The first Subsection 3.1 describes the proposed approach, and the second Subsection 3.2 describes the data-set on which the experiments are performed.

### 3.1 Proposed Algorithm

Before applying the algorithm to the data, it is necessary to analyze the data-set in order to gain insight into the behaviour and nature of the features. The analysis revealed that it is possible to determine the duration of the DHCP lease, and the websites visited by the devices. Since the URLs often contain the path, queries and ports which are unique to the specific request, they are removed from the URL and only the domain is extracted to reduce the dimensionality without the loss of information provided by the URL.

Since the data-set contains tens of millions of flows, it becomes computationally difficult to classify each flow by itself. Therefore, it is necessary to aggregate the flows into a profile that will be stored and later compared to the current device behavior. As it is necessary to aggregate multiple flows into one profile, the nature of the data-set (and aggregated profiles) very closely resembles written text.

One of the most common methods used to analyze text and compare similarity of texts is using frequency analysis in combination with a similarity measure, such as Jaccart or cosine similarity. Due to the resemblance of the aggregated flows to text, it enables us to create profiles using frequency analysis for these devices. Therefore, the exported data is separated into 5-minute segments and profiles are created based on the usernames and source IP addresses, since MAC addresses that would uniquely identify devices are rarely available in a production environment. They are, however, used as labels for the purposes of this paper. This combination of features (source IP address and username) to create device profiles has been chosen, since it provides a very simple and effective way to distinguish devices in a given time-window. However, this combination of features is not sufficient to identify a specific device, since a user can own multiple devices and the source IP address of a given device may change over time.

Since some of the data might contain duplicates of the same value but in a different context (such as source and destination IP addresses), these features are analyzed separately and then appended to the vector containing frequencies of all of the features (an example of flow aggregation based on source IP address of a device can be seen Table 1 and Table 2). This approach enables the creation of user profiles that are memory efficient and accurate for the given 5-minute window. Afterwards, it is possible to compare the similarity of these vectors in any of the following 5-minute time-frames.

| Flow 1 | Flow 2 | Flow 3 | Flow 4 |
|---|---|---|---|
| User 1 | User 1 | User 1 | User 1 |
| src IP A | src IP A | src IP A | src IP A |
| dst IP B | dst IP B | dst IP C | dst IP D |
| port 1 | port 1 | port 1 | port 2 |
| google.com | google.com | - | facebook.com |
| TLS 1 | TLS 1 | TLS 2 | - |
| - | HTTP f 1 | - | - |

**Table 1.** Example of a 5-minute aggregated profile for a device based on its source IP address.

The behaviour of users and devices can change over time, therefore, the previously created profiles need to be updated over time and the older profiles

| Term | Frequency |
|---|---|
| src IP A | 4 |
| dst IP B | 2 |
| dst IP C | 1 |
| dst IP D | 1 |
| port 1 | 3 |
| port 2 | 1 |
| google.com | 2 |
| facebook.com | 1 |
| TLS 1 | 2 |
| TLS 2 | 1 |
| HTTP f 1 | 1 |
| - | 5 |

**Table 2.** Example of frequency analysis including only the features used in classification. The column frequency represents the vector used in similarity comparison.

need to be replaced by newer ones. Device users tend to change the context of their work approximately every hour, therefore, the number of 5-minute profiles that are stored in the memory is limited to 12. If there are 12 profiles, each time a device is classified as a specific device, the oldest profile is deleted and the current one is added into the profile pool. In the case that there are less than 12 profiles, the newest profile is directly added to the profile pool.

Since each of the devices has multiple profiles created for it, it is possible to use the k-Nearest Neighbors algorithm with cosine similarity as the distance metric (defined by Equation 1 [5]) to find the nearest behaviour profile for a given device. The value k has been chosen to be 3, since less than 1% of devices have communicated in only one 5-minute time-frame during the initialization phase. The initialization phase consists of profile creation in the first hour of the exported data using the frequency analysis method described earlier in this section. This approach, however, does not account for the new devices that appeared later than the first hour of the export. Therefore, flows that contain a previously unseen username have a profile created for that device. In the case when 2 of the 3 highest similarity values are below a threshold (set to 0.9, value selection is described in Section 4), the device is considered to be a new, previously unseen device.

$$similarity = cos(\theta) = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \quad (1)$$

Device profiles are stored in memory in the form of cascading hash tables to improve searching efficiency. The following Figure 1 describes the system how device profiles are stored.

```
User 1
  Device 1
    [Profile 1, Profile 2, ...]
  Device 2
    [Profile 1, Profile 2]
User 2
  Device 1
    [Profile 1, ...]
...
```

**Figure 1.** Hierarchy describing the structure of profiles stored in memory.

### 3.2 Data-set Description

Data-sets that are used for experiments of the proposed algorithm are provided by Cisco Systems and contain live data, therefore, the data-set itself cannot be published. The data has been collected using a combination of NetFlow, DNS queries and by using HTTP and TLS fingerprinting methods. However, features such as URLs, HTTP and TLS fingerprints are not always available, therefore, it is necessary to propose an algorithm that is robust and can overcome the sparsity of available data. The following features are available: username, source IP address, destination IP address, port, URL, HTTP fingerprint, TLS fingerprint, and the MAC address. The subsequent Table 3 describes basic information regarding the used data-set.

| | |
|---|---|
| Total flows | 64,806,407 |
| Length of export [h] | 16 |
| Total users | 7,385 |
| Total devices | 10,737 |
| Total unique domains | 53,226 |
| Total unique TLS fingerprints | 1,105,546 |
| Total unique HTTP fingerprints | 0 |
| Availability of URL [%] | 0.159 |
| Availability of TLS fingerprint [%] | 0.170 |
| Availability of HTTP fingerprint [%] | 0.0 |

**Table 3.** Basic information regarding the tested data-set.

## 4. Experimental Results

Due to the collected data from the 16-hour period missing values, it is first necessary to determine how do the available or unavailable features influence the accuracy of the proposed algorithm. Therefore, the first experiment evaluates the accuracy of the algorithm when the first hour is used for profile creation, and the second hour for scoring the accuracy without the

detection of new devices, where k-NN is used with k equal to 3. As it is always available, the source IP address is used to aggregate data for the profile in a given 5-minute period. Subsequently, the MAC addresses are used as the labels to evaluate the accuracy of the prediction. The following Table 4 describes the accuracy of the algorithm. From the table it is possible to come to the conclusion that even features that are often not available, such as TLS or HTTP fingerprints, improve the accuracy of the algorithm significantly.

| Features | Accuracy [%] |
|---|---|
| source IP, destination IP, TLS fingerprint, HTTP fingerprint port, domain | 93.65 |
| source IP, destination IP, HTTP fingerprint port, domain | 93.61 |
| source IP, destination IP, TLS fingerprint port, domain | 92.90 |
| source IP, destination IP, port, domain | 13.22 |
| source IP, destination IP, port | 0.0 |

**Table 4.** Results of the experiments evaluating the accuracy, depending on the used features.

The following step was to verify that the cosine similarity that is used as the distance metric creates a gap in similarity values between devices with the same and different MAC address. The following Figure 2 describes the distribution of values between the same and different devices. The device pairs that have been used for the comparison of similarity have been chosen at random to get the best representation of the distribution.

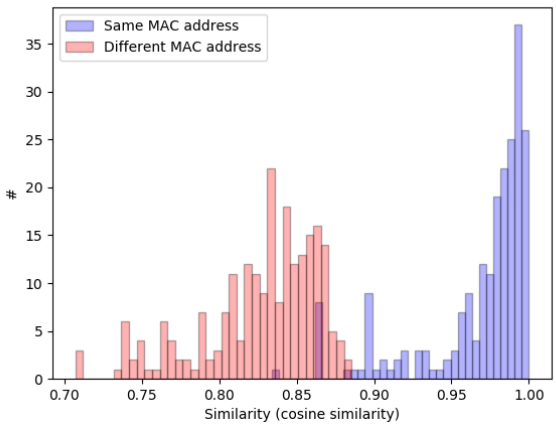



**Figure 2.** Histogram describing the distribution of cosine similarities between device profiles with the same MAC addresses (blue) and different MAC addresses (red).

The mean for the similarity of devices with the same MAC addresses is 0.98 and for devices with different MAC addresses the mean is 0.83. From this graph it is possible to conclude that there is a gap between the values of similarity between devices with the same and different MAC addresses, therefore, cosine similarity is the appropriate distance metric to be used in device identification. The following Figure 3 describes the distribution of device profiles, and visualizes how the profiles create clusters. Due to the device profile clustering, k-NN is an appropriate approach towards solving the problem.

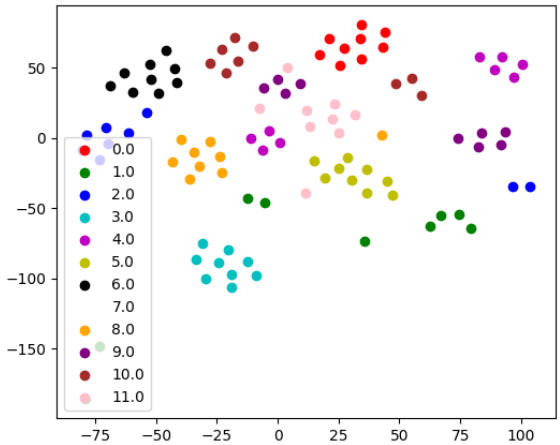



**Figure 3.** t-Distributed Stochastic Neighbor Embedding (t-SNE) used to visualize how device profiles are clustered. Each of the number labels represents one device and each point is a device profile.

Then, the effect of new device detection on accuracy needs to be evaluated. During this experiment the first hour has been used to create the initial profiles, as described in Section 3, and the effect of detection of new devices over time is evaluated. A device is classified as a new device, when all of the nearest neighbor distances are bellow the 0.9 threshold, chosen based on the histogram in Figure 2. The Table 5 describes the accuracy of the base algorithm. The following Table 6 then describes the accuracy during the given hour, without the replacement of old profiles.

| Hour | Accuracy | Precision | Recall | F1 |
|---|---|---|---|---|
| 2 | 0.91304 | 0.88956 | 0.92931 | 0.89126 |
| 3 | 0.87210 | 0.84849 | 0.90754 | 0.84961 |
| 8 | 0.75068 | 0.72366 | 0.84124 | 0.72577 |

**Table 5.** Experimental results for the base algorithm without device detection or profile updates, and its accuracy over time.

| Hour | Accuracy | Precision | Recall | F1 |
|---|---|---|---|---|
| 2 | 0.92779 | 0.89310 | 0.93256 | 0.89548 |
| 3 | 0.90626 | 0.85449 | 0.91313 | 0.85642 |
| 8 | 0.82190 | 0.73724 | 0.84763 | 0.73786 |

**Table 6.** Experimental results with new device detection without profile updates, and its accuracy over time.

Table 6 illustrates that the detection of new devices using the methods described in Section 3 provides a significant improvement in accuracy of the device identification algorithm.

The last experiment evaluates the accuracy of the proposed method when the algorithm is exchanging the profiles for the newer ones after each classification. This experiment also uses the concept of learning new devices during the classification phase as described in the previous experiment. The first hour is used for the initial profile creation and the following hours of the export are evaluated. The following Table 7 describes the achieved results. Lastly, Figure 4 compares the achieved accuracy of the aforementioned algorithms.

| Hour | Accuracy | Precision | Recall | F1 |
|---|---|---|---|---|
| 2 | 0.92855 | 0.89815 | 0.93714 | 0.90031 |
| 3 | 0.90696 | 0.86116 | 0.91963 | 0.86295 |
| 4 | 0.88689 | 0.81799 | 0.89923 | 0.82030 |
| 5 | 0.87956 | 0.79107 | 0.88519 | 0.79327 |
| 6 | 0.86123 | 0.76723 | 0.87079 | 0.76893 |
| 7 | 0.84820 | 0.75270 | 0.86247 | 0.75451 |
| 8 | 0.83831 | 0.74328 | 0.85536 | 0.74492 |
| 9 | 0.83319 | 0.74001 | 0.85298 | 0.74167 |

**Table 7.** Accuracy and further metrics of the final proposed algorithm with the addition of the sliding window of profiles.

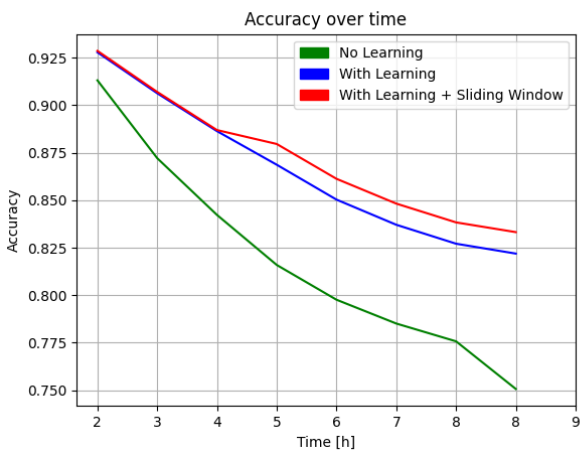



**Figure 4.** Comparison of all of the mentioned algorithms and their achieved accuracy over time.

Overall the average accuracy of the proposed algorithm is 89.26% over the tested 8 hour period. The decrease in accuracy over time is caused by incorrectly classified profiles, which cause a skew towards incorrect device classifications over the time period. However, the decrease in accuracy appears to level out at hour 9, which implies a consistent error rate in the future predictions.

## 5. Conclusion

This paper describes the implementation of the network device tracking algorithm using frequency analysis and the k-Nearest Neighbors algorithm with cosine similarity as the distance metric. The steps taken during the development of the algorithm are explained, along with the improvements over the previous version of the algorithm. Generally, the algorithm has proven to be effective over the tested 8 hour period, achieving the average tracking accuracy of 89.26%. The proposed algorithm provides a significant improvement compared to the currently used tracking methods, such as the method proposed by Kumpost achieving the accuracy at 78.3%. Furthermore, the proposed approach does not require any active querying, as clock-skew does, and proves to be an effective approach to passively track devices.

The only downfall of the algorithm is the limited ability to detect new devices based on their similarity. Therefore, algorithms such as outlier detection should be explored further to improve the accuracy of the proposed approach.

## Acknowledgements

I would like to sincerely thank my supervisor Ing. Libor Polčák, Ph.D. for his valuable advice and guidance. Furthermore, I would like to thank Mgr. Jan Kohout and Cisco Systems for their advice and patience throughout the development of the algorithm.

## References

[1] Blake Anderson. Tls fingerprinting in the real world, April 2019. Online; visited 3.4. 2020. Available at: https://blogs.cisco.com/security/tls-fingerprinting-in-the-real-world.

[2] Marek Kumpošt. *Context information and user profiling*. Masaryk University, Faculty of Informatics. Supervisor Vašek Matyáš, Brno, 2009. PhD thesis.

[3] Tadayoshi Kohno, Andre Broido, and K.C. Claffy. *Remote Physical Device Fingerprinting*. IEEE, 2005. ISSN 1545-5971.

[4] Steven J. Murdoch. Hot or not: Revealing hidden services by their clock skew. *Computer and Communications Security*, pages 27–36, 2006.

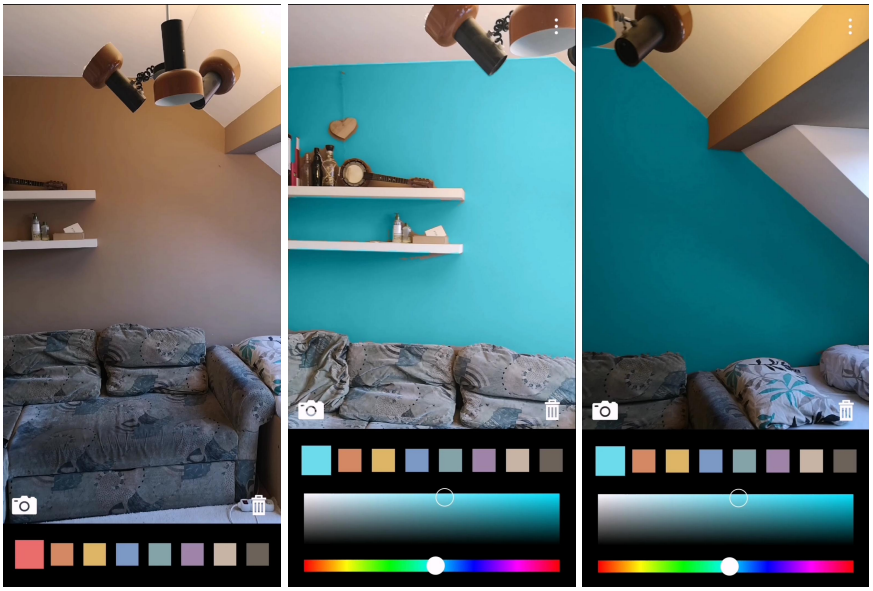
[5] Jun Ye. Cosine similarity measures for intuitionistic fuzzy sets and their applications. *Mathematical and Computer Modelling*, pages 91–97, 2011. ISSN: 0895-7177.

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

# Náhľad farby na stene pomocou rozšírenej reality v mobile

Dominik Vagala



**Abstrakt**
Cieľom tejto práce je návrh a implementácia mobilnej aplikácie pre Android, ktorá by umožňovala meniť farby na stene pomocou rozšírenej reality. Užívateľ si tak môže vyskúšať rôzne farby priamo v miestnosti, kde sa nachádza a následne sa rozhodnúť, ktorá farba sa mu najviac páči na vymaľovanie stien. Na rozpoznanie hraníc steny je použitý Sobelov detektor hrán, kde sa ohraničený úsek steny vypĺňa farbou pomocou upraveného Queue-Linear Flood Fill algoritmu. 2D súradnice, kde užívateľ klikol na stenu, sa približne prepočítajú na 3D súradnice v priestore. Tie sa následne sledujú pomocou knižnice ARCore, vďaka čomu stena zostane zafarbená, aj keď sa užívateľ pohybuje po miestnosti.

**Kľúčové slová:** Paint my room — Change wall color — Náhľad farby na stene — Rozšírená realita — AR — ARCore — Mobilná aplikácia — Android

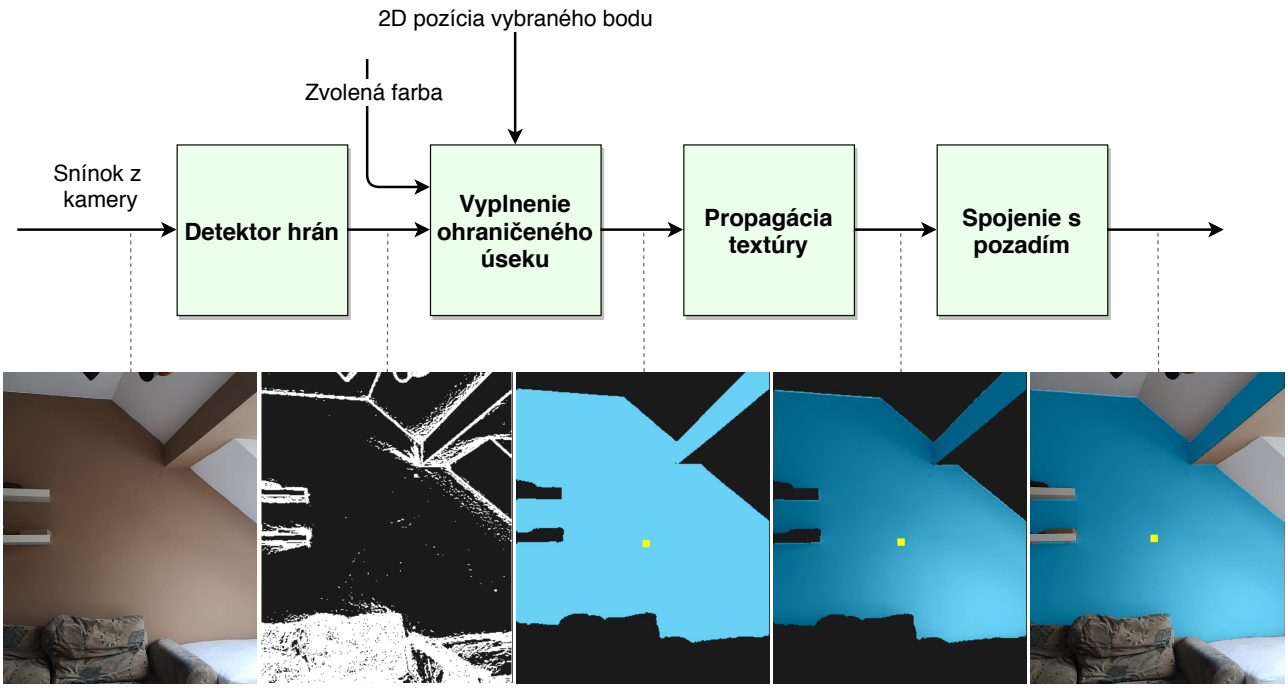
**Priložené materiály:** Aplikácia na Google Play, Demonštračné video

*xvagal00@fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

## 1. Úvod

Ľudská vizuálna predstavivosť má svoje limity, a preto sú častokrát potrebné riešenia, ktoré vedia očakávanú víziu zobraziť. Moja aplikácia ponúka riešenie v modelovej situácii, keď si človek chce ísť vymaľovať izbu, a nevie sa rozhodnúť, akú farbu si vybrať.

V aplikácií riešim, čo najviac realistické zobrazenie farby na stene, ktorú si užívateľ zvolil. Je to kľúčový faktor, pretože ak vizualizácia farby nebude vyzerať dostatočne reálne, užívateľ si nebude vedieť predstaviť, ako by miestnosť skutočne vyzerala vymaľovaná s danou farbou. Výsledná aplikácia by preto




**Obrázok 1.** Bloková schéma vykresľovania.

mala spoľahlivo rozlíšiť hranicu steny, aby sa farbou neprekrývali okolité objekty, a aby boli v prekrývanej farbe zachované tiene a odlesky steny. Ďalším dôležitým faktorom je, aby vizualizácia prebiehala v reálnom čase, čiže užívateľ sa môže voľne pohybovať po miestnosti a sledovať vyfarbené steny z rôznych uhlov. Užívateľ si vždy najskôr klikne na stenu, ktorú si chce vymaľovať.

V blokovej schéme na obrázku 1 je zobrazený zjednodušený postup, ktorý sa aplikuje na každý nový snímok. Na zistenie hraníc steny je použitý detektor hrán, ktorého výstupom je binárna maska. Na sledovanie bodov, ktoré si užívateľ vybral, je použitá knižnica ARCore [1]. Z týchto bodov sa následne vychádza pri vyfarbovaní steny, kedy sa vyplní ohraničený segment v binárnej maske, kde sa daný bod nachádza.

## 2. Existujúce riešenia

Existuje niekoľko mobilných aplikácií na Android, ktoré vedia zmeniť farbu steny na fotke, ale len dve, ktoré riešia tento problém v reálnom čase: Dulux Visualizer[1] a ColorSnap[2] (na väčšine zaradení spadne). V obidvoch prípadoch je priestor na zlepšenie. Pri pohybe sa vyfarbovaný úsek nepropaguje ďalej, ale zmizne hneď, ako sa prejde za bod, kde používateľ klikol na stenu. Tiež je v užívateľskom rozhraní potrebných príliš veľa krokov na to, aby si užívateľ zmenil zvolenú farbu steny. Je dôležité, aby bol tento proces čo najrýchlejší a užívateľ mohol jednoducho alternovať medzi farbami.

[1] https://play.google.com/store/apps/details?id=com.akzonobel.cz.dulux

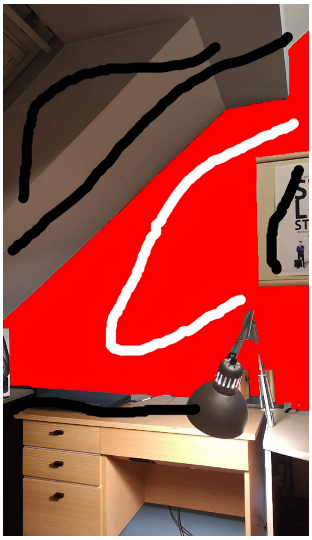
[2] https://play.google.com/store/apps/details?id=com.colorsnap

## 3. Rozpoznávanie steny

Základným problémom, ktorý bolo treba riešiť, je rozpoznanie úseku steny, ktorý má byť zafarbený.

Prvotný návrh bol, robiť toto rozpoznanie len na základe farby jednotlivých pixelov v snímke (obrázok 2). Od miesta, na ktoré užívateľ klikol na stenu, by sa postupne vyplňovacím algoritmom prechádzalo jeho okolie, a ak je daný pixel v určitej farebnej tolerancii, zafarbí sa. Vyskúšal som rôzne prístupy porovnania farieb, euklidovskú vzdialenosť v RGB, alebo CIE94.

**Obrázok 2. Vľavo:** Pôvodný snímok z kamery. **Vpravo:** Nekvalitné rozpoznanie steny – porovnávanie zložky U a V.



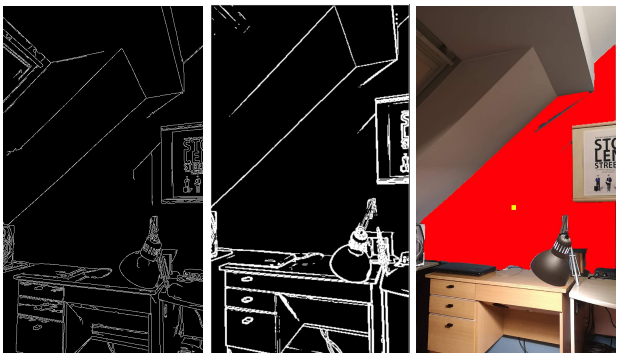
**Obrázok 3.** Rozpoznanie steny – algoritmus GrabCut s čiarami, ktorými užívateľ označil úsek steny.

Počas experimentovania som ale zistil, že najvhodnejšie je porovnávať vo farebnom modeli YUV [2] iba zložky U (intenzita modrej) a V (intenzita červenej), pričom Y (jas) sa ignoruje. To trochu pomohlo riešiť problém s tieňmi a odleskami na stene, avšak stále to nebolo postačujúce.

Ďalej som zvážil použitie segmentačných algoritmov. Vyskúšal som algoritmus GrabCut [3] implementovaný v knižnici OpenCV. Výsledok bol najlepší z použitých prístupov. Problém bol v tom, že nestačilo iba kliknúť na stenu, ale užívateľ by musel celkom detailne vyznačiť úsek steny, aby algoritmus vyprodukoval kvalitný výstup. Na obrázku 3 je to znázornené čiarami: biela – tento úsek vyber, čierna – tento úsek zahoď. Na statickom snímku by sa to dalo použiť, ale pri použití v reálnom čase, by bolo problém sledovať tieto vyznačené čiary. Navyše doba výpočtu sa pohybovala rádovo v sekundách.

Finálne riešenie spočíva v použití detektoru hrán. Algoritmus je založený na Sobelovom detektore hrán [4] (obrázok 4), ktorého výstup je prahovaný, aby sa získala binárna maska. Následne sa z miesta, kde užívateľ klikol na stenu vyplňovacím algoritmom, vyplní táto ohraničená časť. Hlavný problém tohto prístupu je, že niekedy ohraničený úsek steny obsahuje medzery a tým pádom sa vyplnia aj úseky, ktoré nemajú byť vyplnené. To spôsobuje nepríjemné preblikovanie chybne vyfarbených častí. Snažil som sa to riešiť rozšírením hrán v maske (dilation) s následným stiahnutím (erosion), aby sa tieto medzery vyplnili. Vo výsledku to ale spôsobilo, že malé parazitné hrany na stene, spôsobené tieňmi sformovali súvislé bloky, čiže tam zostala stena nevyfarbená. Preto zatiaľ jediným riešením je vhodne určiť prah tohto detektoru.

Vyskúšal som aj algoritmus Canny Edge Detection [5] (obrázok 4) v OpenCV, no je výpočetne náročnejší a medzery v ohraničenom úseku sa vyskytovali ešte častejšie.



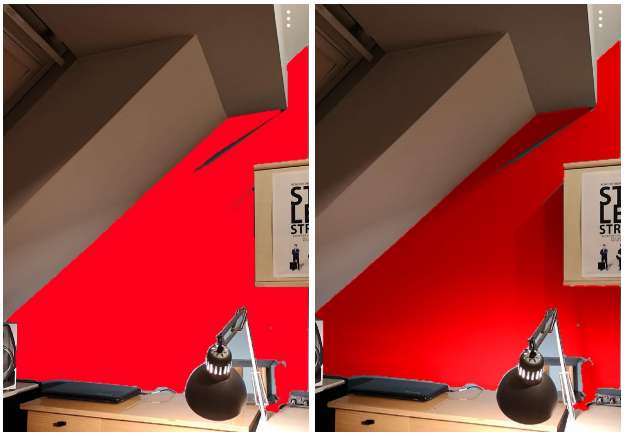
**Obrázok 4.** Porovnanie detektorov hrán. **Vľavo:** Canny edge detector. **V strede:** Sobel Operator. **Vpravo:** Sobel operator – vyplnenie úseku – finálne riešenie.

## 4. Vyplnenie ohraničeného úseku

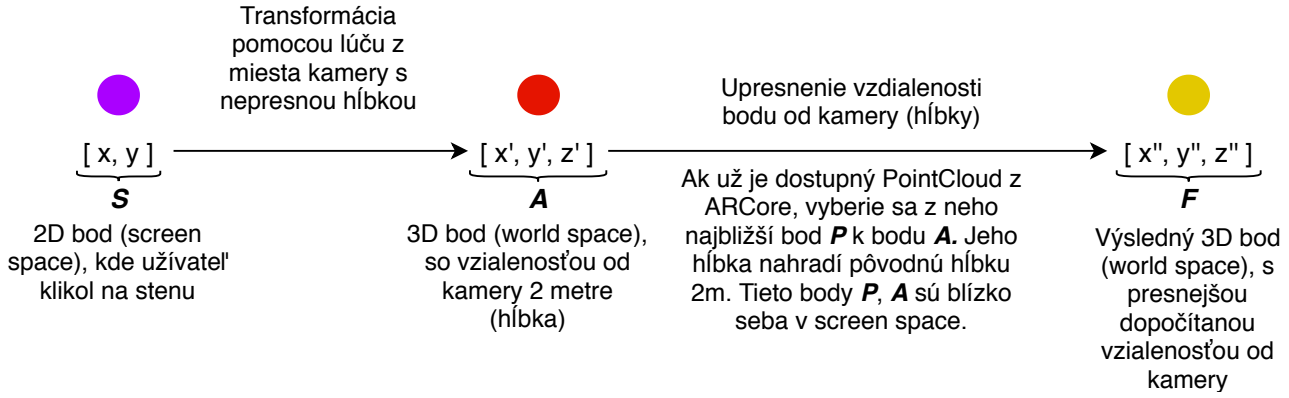
Na vyplnenie ohraničeného úseku, ktorý poskytne detektor hrán je použitý trochu upravený algoritmus Queue-Linear Flood Fill [6]. Je pomerne rýchly a nie je pamäťovo náročný. Tento vyplnený úsek je následne rozšírený o fixnú veľkosť, aby farba doliehala presne až ku okrajom steny a aby sa zakryli prípadné parazitné hrany vo vnútri steny.

## 5. Propagácia textúry steny

Aby vyfarbený úsek steny pôsobil reálne, musí obsahovať tiene a odlesky pôvodnej steny (obrázok 5). V použitom algoritme sa z pôvodného pixelu steny, zoberie komponenta jasu (Luminance) vo farebnom modeli YUV a primieša sa do prekrývanej farby.



**Obrázok 5. Vľavo:** Bez propagácie textúry. **Vpravo:** S propagáciou textúry.




**Obrázok 6.** Transformácia vybraného bodu.

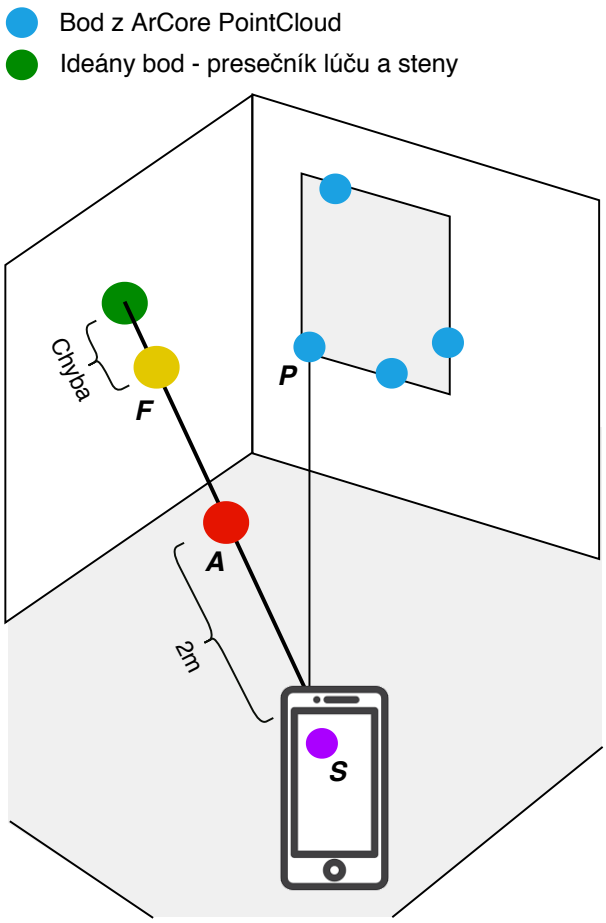
## 6. Sledovanie vybraných bodov

V mieste, kde užívateľ klikne na stenu, sa začína s vyplňovaním ohraničeného úseku. Aby sa užívateľ mohol voľne pohybovať po miestnosti, je potrebné tieto 2D súradnice previesť na bod v 3D priestore a sledovať ho. Postup tejto transformácie je načrtnutý v obrázkoch 6 a 7. Tento 3D bod (world space) je pri každom novom snímku prevedený späť na bod v 2D (screen space), ktorý sa odovzdá vyplňovaciemu algoritmu. Táto transformácia sa vykoná pomocou aktuálnych matíc *projection* a *view* z ARCore.

Spomenutý PointCloud [7] sú body v priestore, ktoré poskytuje ARCore. Tieto body sú typicky dostupné až vtedy, keď sa užívateľ začne pohybovať po miestnosti a sú na miestach, ktoré vie ARCore jednoducho odlíšiť od okolia. Preto sa nevyskytujú na holej stene, ale napríklad na nábytku s textúrou alebo na okrajoch objektov (obrázok 7). Kvôli tomuto faktu som nemohol použiť Plane z ARCore a robiť iba jej priesečník s lúčom z kamery. Užívateľ by musel najprv detailne prejsť celý priestor, aby sa mu Plane spočítala a navyše by mu vykreslená prekážala vo výhľade na stenu.

Vďaka prístupu, ktorý som použil, môže užívateľ používať aplikáciu aj bez toho, aby musel vopred zmapovať okolie. Ak sa kamera nehýbe, alebo ak rotuje, umelá vzdialenosť (2 m) neprekáža, pretože z pohľadu kamery je na tom istom mieste. Ak sa kamera začne pohybovať, tak sa vzdialenosť upresní, pretože je už dostupný PointCloud.

Z tohto prístupu je zrejmé, že môže nastať situácia, kedy bude mať žltý bod **F** veľkú *chybu* (obrázok 7), a kamera môže zmeniť svoju pozíciu tak, že bod bude na chybnom úseku steny a vyfarbí sa. Riešenie spočíva v tom, že v momente ako sa v mieste snímku daného 2D súradnicou bodu **F** vyskytne iná farba, ako tá, ktorá tam bola pri vzniku tohto bodu, tento bod sa zahodí. Táto farebná podobnosť je založená na algoritme popísanom v kapitole 3.

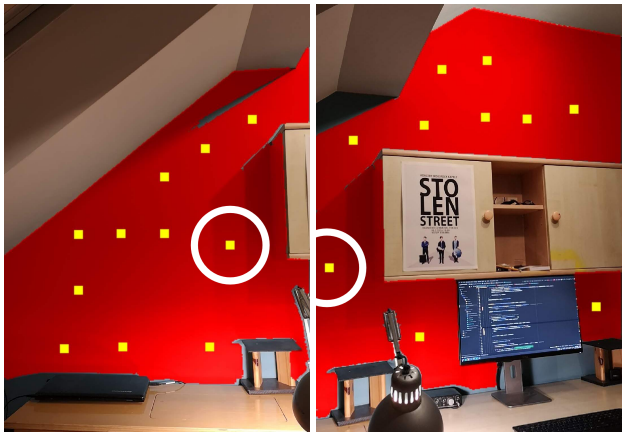



**Obrázok 7.** Lúč vychádzajúci z kamery, zobrazenie jednotlivých bodov.

## 7. Propagácia vybraných bodov

Pre jednoduchosť bolo doteraz popisované, že vybrané body sa vytvárajú iba užívateľom, ktorý klikne na stenu. V skutočnosti sa ale vždy k takémuto bodu pridajú ďalšie automaticky generované body v primeraných rozostupoch. Tie sa generujú iba na mieste, kde je stena už zafarbená. Pre výpočet týchto bodov platia rovnaké princípy ako som popisoval doteraz. Tieto body sa automaticky generujú v prípade, že v danom momente je viditeľných menej vybraných bodov ako

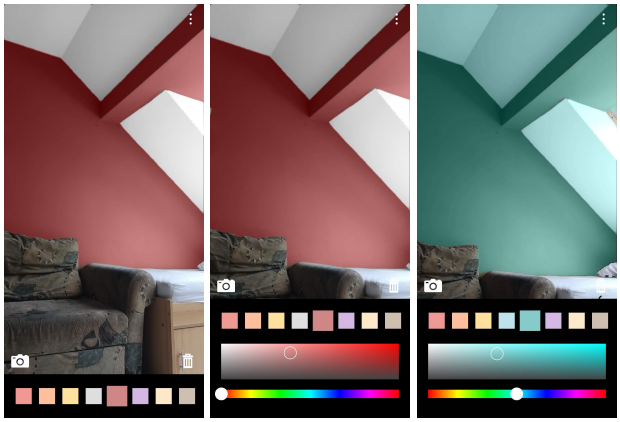
určitá konštanta. Vďaka tomu dochádza ku automatickej propagácii vybraných bodov pri pohybe (obrázok 8).




**Obrázok 8.** Propagácia vybraných bodov pri zmene uhlu kamery. Bod v krúžku znázorňuje jediný bod, ktorý bol vytvorený užívateľom kliknutím na stenu. Ostatné boli generované automaticky.

## 8. Návrh užívateľského rozhrania

Návrh užívateľského rozhrania (obr. 9) som priebežne testoval na užívateľoch, aby som zistil, ktoré prvky sú nejasné. Najväčší problém bol, ako docieliť, aby užívateľ mohol používať viaceré farby naraz na jednotlivé úseky steny a zároveň, aby mohol rýchlo tieto farby meniť a zisťovať, ktorá sa mu páči najviac. Použité widgety na výber farby som si navrhoval a implementoval sám, s existujúcimi riešeniami som nebol spokojný.



**Obrázok 9.** Návrh užívateľského rozhrania. Ukážka vysúvacej palety.

## 9. Optimalizácie

Všetky použité algoritmy sú pomerne rýchle. V čase písania tohto článku dosahuje vykresľovanie 30 fps na zariadení Huawei p20 pro. Čiže zhruba 33 ms trvá spracovanie jedného snímku. Väčšina algoritmov v mojom riešení prebieha na jednom vlákne v pozadí, iba niektoré sa dali implementovať na GPU. Do budúcna chcem zvážiť rozdelenie týchto sekvenčných výpočtov do viacerých vlákien. Chcel by som tiež optimalizovať detektor hrán, pretože je to najslabší článok celého riešenia. Do aplikácie ešte pridám uloženie fotky vyfarbenej miestnosti, mať možnosť meniť farby aj na statickej fotke a zobrazenie kódu danej farby, aby si ju užívateľ mohol v predajni kúpiť.

## 10. Zverejnenie aplikácie na Google Play

Aplikácia *Paint my Room* bola najskôr zavedená v uzavretom Alfa kanáli, kde som ju testoval prostredníctvom rôznych užívateľov a na rôznych zariadeniach. Teraz je pridaná do otvoreného Beta kanálu[3], kde sa dá aplikácia voľne stiahnuť.

## 11. Záver

Cieľom tejto práce bol návrh a implementácia mobilnej aplikácie pre Android, ktorá umožňuje meniť farby na stene pomocou rozšírenej reality. Vďaka tomu si užívateľ môže farbu lepšie predstaviť a vybrať naozaj takú, aká sa mu hodí do jeho miestnosti.

Hlavnou časťou tejto práce bolo experimentovanie s rôznymi prístupmi a algoritmami, ktoré sa dajú použiť v reálnom čase. Neoddeliteľnou súčasťou bol taktiež návrh intuitívneho užívateľského rozhrania. Výsledkom je teda funkčná aplikácia, zverejnená na Google Play v otvorenom Beta kanáli[3].

Vo vývoji aplikácie pokračujem ďalej, chcem sa zamerať na optimalizovanie rýchlosti vykresľovania a pridanie možnosti uložiť a zdieľať snímku vyfarbenej miestnosti (viď kapitolu 9). Po pridaní týchto funkcií a otestovaní, bude aplikácia zverejnená do produkčného kanálu, kde bude voľne dostupná užívateľom.

## Poďakovanie

Rád by som sa poďakoval môjmu vedúcemu prof. Ing. Adamovi Heroutovi, PhD. za cenné rady a čas strávený na konzultáciách.

## Literatúra

[1] Arcore overview — google developers. https://developers.google.com/ar/discover.

[2] Yuv color system. https://www.hisour.com/yuv-color-system-25916/.

[3] https://play.google.com/store/apps/details?id=com.bonc.paintmyroom

[3] Opencv: Interactive foreground extraction using grabcut algorithm. https://docs.opencv.org/trunk/d8/d83/tutorial_py_grabcut.html.

[4] Ashish. Understanding edge detection (sobel operator) - data driven investor - medium. https://medium.com/datadriveninvestor/understanding-edge-detection-sobel-operator-2aada303b900, Sep 2018.

[5] Canny edge detection step by step in python — computer vision. https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123, Jan 2019.

[6] J. Dunlap. Queue-linear flood fill: A fast flood fill algorithm - codeproject. https://www.codeproject.com/Articles/16405/Queue-Linear-Flood-Fill-A-Fast-Flood-Fill-Algorith, Nov 2006.

[7] Pointcloud — arcore — google developers. https://developers.google.com/ar/reference/java/arcore/reference/com/google/ar/core/PointCloud.

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

# Simulace sněhových a ledových struktur

Ondřej Čech*

**Abstrakt**
Cílem práce je vytvořit simulaci toho, jak vznikají ledovcové struktury známé jako Kajícníci neboli Penitentes. Tyto útvary vznikají obvykle ve vysokohorském prostředí a podle posledních výzkumů jsou způsobeny odrážením slunečního záření uvnitř prohlubní na povrchu ledovce a jejich prohlubování sublimací při teplotách pod bodem mrazu. Byl implementován matematický model popisující toto chování, který vytvořila Meredith D. Betterton. Na tomto modelu byly poté provedeny experimenty, které měly za úkol potvrdit nebo vyvrátit, zda skutečně vede k vytvoření těchto útvarů. Jelikož původní model pracuje pouze s jednotlivými řezy, bylo třeba tento model upravit, aby se ovlivňovaly řezy navzájem.
Vznik těchto útvarů stále není dokonale prozkoumán a práce, které se jím v průběhu let zabývaly, si v některých částech i protiřečí. Přitom vlastnosti těchto struktur zřejmě zpomalují tání ledovce, na kterém se nacházejí, což by v současné době mohly být cenné znalosti.

**Klíčová slova:** Penitentes — Simulace tání — Ledovcové struktury

**Přiložené materiály:**

*xcecho06@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

## 1. Úvod

Kajícníci jsou struktury z ledu nebo sněhu, které mají podobu tenkých zašpičatělých štítů dosahujících výšky až 5 metrů. Jedná se o skupiny štítů orientovaných obecně ve směru z východu na západ se zdmi obrácenými k severu a jihu. Jejich první popis pochází z třicátých let 19. století, kdy je poprvé uviděl a popsal Charles Darwin. Během času přišli různí vědci s teoriemi jejich růstu, bylo provedeno mnoho měření a pokusů, V současné době obecně panuje přesvědčení, že za vznik může sluneční záření, které ve spolupráci s dalšími podnebními faktory může za úbytek sněhu v okolí, zatímco tato struktura přežívá. V nedávné době se povedl experiment vytvoření kajícníků uměle v laboratoři na základě této teorie.

Význam znalosti vzniku kajícníků spočívá v jejich schopnosti zpomalit tání ledovce, protože tvoří větší povrch, který se prouděním studeného vzduchu ochlazuje, a rovněž rozdíly mezi teplotami uvnitř prohlubní a na vrcholech mohou vést k tomu, že část odpařené vody v prohlubních opět zkondenzuje na vrcholech a voda tak zůstane v ledovci [1]. Tato práce má za cíl vytvořit na základě známých informací o tomto jevu simulaci vzniku těchto struktur.

## 2. Kajícníci

Útvary nazývané Kajícníci (španělsky Penitentes) vznikají ve vysoké nadmořské výšce v období ablace, což je časový interval, kdy ledovec ztratí více objemu, než získá. Běžně se vyskytují v Andách mezi Chile a Argentinou v nadmořské výšce 4000 až 5200 metrů nad mořem, také se objevují v Himalájích, pohoří Sierra Nevada, na Kilimandžáru a jiných horách s vy-

hovujícím klimatem, rovněž byly takovéto útvary nalezeny na Plutu a mohly by se nalézat v rovníkových oblastech Jupiterova měsíce Europa [2]. Klima, které vyhovuje růstu Kajícníků lze popsat jako suché, studené, slunečné a stabilní. Ve výškách nad 5200 metrů zůstává sníh příliš prašný, aby se v něm mohli kajícníci vytvořit [3]. Tyto struktury tvoří skupiny štítů orientovaných obecně ve směru z východu na západ se zdmi obrácenými k severu a jihu, jejich vrcholy se orientují k zenitálnímu úhlu slunce během poledne, kdy je jeho záření nejvyšší. Ukázka vyvinutého pole kajícníků je na obrázku 1. Název kajícníků pochází od jejich vzhledu, který připomíná účastníky procesí během Velikonočních svátků ve Španělsky mluvících zemích. Ti jsou oblečeni v dlouhých bílých hábitech a vysokých špičatých pokrývkách hlavy, nazývaných *capirote*.

**Obrázek 1.** Pole kajícníků v Andách, Argentina [1].

Poprvé tyto útvary popsal britský přírodovědec Charles Darwin, který se s nimi setkal při cestě ze Santiago de Chile do Argentinského města Mendoza v březnu 1835. Podle místních obyvatel za jejich tvorbou stojí silné větry [4]. Ohledně jejich vzniku vzniklo časem několik dalších teorií. Louis Lliboutry roku 1954 zjistil, že za vznikem kajícníků stojí sluneční záření ve chvíli, kdy je rosný bod pod bodem mrazu, což vede k ablaci ledu sublimací [3]. Roku 2006 Vance Bergeron, Charles Berger, a M. D. Betterton poprvé vytvořili kajícníky v laboratoři [5]. Ve velkém mrazáku zakrytém plexisklem svítili na kus ledu, přičemž vzduch vcházející dovnitř byl chlazen tekutým dusíkem. Jako zdroj záření byl použit světlomet, světelný zdroj umístěný v zrcadlovém odražeči, který směřuje všechny paprsky záření stejným směrem. Během několika hodin se na ledu objevily 1–5 cm vysoké ostny. Podobné experimenty prováděl už ve třicátých letech německý geolog Carl Troll, který za suché chladné noci svítil na čerstvý sníh žárovkou [3].

Proces, který vede ke vzniku kajícníků se nazývá diferenciální ablace. Při ní mění některé části sněhu své skupenství rychleji než jiné. Na počátku je silná vrstva sněhu, která není dokonale rovná, jsou v ní propadliny. Na tento sníh dopadá sluneční záření, které je zachycováno vícenásobnými odrazy mezi zdmi těchto sníženin více, než jinde. Tento jev se nazývá pozitivní zpětná vazba. Jak již bylo zmíněno, klíčový faktor pro tvorbu kajícníků je rosný bod pod bodem mrazu. Takto nevzniká voda v tekutém skupenství, ale sníh sublimuje, což je energeticky náročnější, a výdej této energie vede k ochlazování ledovce. Propadliny se vlivem větší přijaté energie prohlubují, stávají se téměř černým tělesem, které pohlcuje téměř všechno světlo, co na něj dopadá. Zároveň uvnitř roste hodnota rosného bodu, která se může dostat až nad bod mrazu, a sníh uvnitř začne tát. Oproti tomu vrcholy kajícníků stále jen sublimují a jejich strmé stěny zachytí jen velmi málo slunečního záření. Jedním z dalších faktorů ovlivňujících růst kajícníků je znečištění. Tmavé nečistoty pohlcují sluneční záření, ale velmi záleží na síle vrstvy těchto nečistot. Tenkou vrstvou část záření projde, ale už se neodrazí zpět, což ablaci urychluje, naopak moc silná vrstva znečištění ablaci zpomaluje, protože záření se na sníh nedostane. V rámci laboratorního pokusu popsaného dříve také jeho autoři zkusili posypat led tenkou vrstvou sazí. Na vrcholcích tvořících se kajícníků se saze koncentrovaly a tvořily tak stále silnější vrstvu, která je začala chránit před zářením. V prohlubních se zvětšovala plocha a tedy se koncentrace sazí snižovala. Toto vedlo k rychlejší tvorbě útvarů, jelikož vrcholy tály ještě pomaleji, než kdyby byly z čistého sněhu, zatímco údolí se prohlubovala zhruba stále stejně rychle. V přírodě tímto způsobem vznikají takzvané *dirt cones*, známé například z Islandu díky časté sopečné aktivitě, která vede k pokrytí ledovce sopečným prachem.

## 3. Jednorozměrný matematický model

Matematický model vzniku kajícníků vytvořila Meredith D. Betterton [6]. Tento model uvažuje sluneční záření jako primární zdroj tepla. Model se skládá ze dvou částí: výpočtu sněhové ablace a odrazivosti světla. Teplo, které dopadá na povrch, vede k ablaci. Výška povrchu $h$ se snižuje, jak led taje nebo sublimuje. Předpokládá se, že odtátý led se vsákne nebo vypaří, tedy nedochází k toku vody po povrchu a jejímu opětovnému zmrznutí. Bohužel se nepodařilo jiné práce se stejným zaměřením, a není ani známo, že by byl zde popsaný model nějak implementován.

Tento model počítá výšku ledové vrstvy $h(x)$ v daném bodě na základě diferenciální rovnice:

$$\frac{\delta h}{\delta t} = \frac{P(x)}{L} + D\frac{\delta^2 h}{\delta x^2}, \tag{1}$$

která popisuje změnu výšky ledu v čase v závislosti

na světelném záření a tvaru ledu. $P(x)$ je množství energie přijaté v daném okolí bodu x, $L$ je latentní teplo, potřebné k ablaci objemové jednotky ledu. Latentní teplo je uvažováno jako konstantní, což je zjednodušení, jelikož teplota, hustota sněhu a vlhkost vzduchu se mohou v různých místech lišit. Byla vybrána hodnota $L = 10^9\, erg\ cm^{-3}$. *Erg* je jednotka energie v soustavě CGS, kde je její rozměr $g\, cm^2 s^{-2}$. Odpovídá tedy hodnotě $10^{-7}\, J$. Bez vyhlazovacího členu $\frac{\delta^2 h}{\delta x^2}$ může model tvořit libovolně malé struktury, což neodpovídá realitě. Proto je třeba mít nějakou velikost nejmenší vytvořitelné struktury. Přirozeným omezením je například délka zániku slunečního světla, která definuje tloušťku sněhové vrstvy, ve které probíhá rozptyl světla. $D$ je difúzní konstanta a její hodnota byla vybrána $2,5 \times 10^{-5}\, cm^{-2} s^{-1}$.

Světlo dopadá na povrch přímo seshora a přenáší energii. Tato energie je konstantní vůči ploše kolmé na směr paprsků. Při dopadu na led se část světla odráží. Celkové množství světla $\Phi$ odraženého z intervalu okolo bodu $x_1$ do intervalu mezi body $x$ a $dx$ je vypočítáno pomocí rovnice:

$$\Phi = \frac{\alpha I}{\pi} \int d\theta\, dx_1, \tag{2}$$

kde $d\theta$ je úhel s vrcholem v bodě $x_1$ a rameny do bodů $x$ a $x + dx$, $I$ je solární konstanta, která udává intenzitu solárního záření v nejvyšším bodě atmosféry. Jako typická hodnota konstanty $I$ v jasných slunečných podmínkách je bráno $10^6\, ergcm^{-2} s^{-1}$. Parametr $\alpha$ reprezentující odrazivost povrchu se nazývá *albedo*. Albedo označuje, jak velká část světla se odrazí, jeho hodnota byla zvolena $\alpha = 0,5$. Světlo se od sněhu odráží izotropně, tedy rovnoměrně do všech směrů. $d\theta$ lze vyjádřit jako:

$$d\theta = \frac{dl}{p} = \frac{|\mathbf{p} \times ds|}{p} = \frac{\Delta h - \Delta x h'(x)}{\Delta h^2 + \Delta x^2}, \tag{3}$$

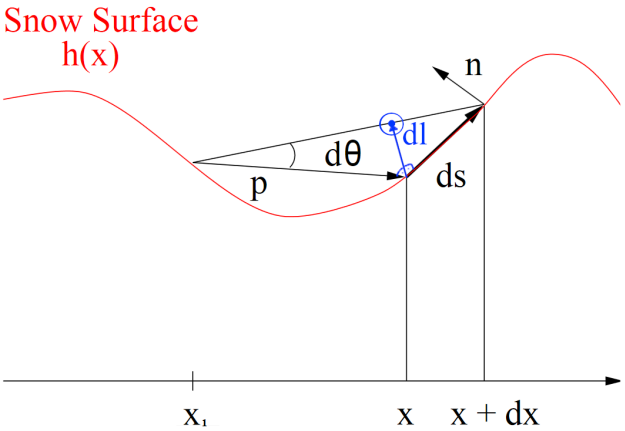
kde je definován vektor **p**, který směřuje z bodu $x_1$ do bodu $x$. Platí:

$$p = |\mathbf{p}| = \sqrt{\Delta h^2 + \Delta x^2}. \tag{4}$$

Pro zjištění celkové energie odražené do bodu $x$ je třeba připočítat intenzitu ze všech bodů z okolí bodu $x_1$

$$P_r(x) = \frac{\alpha I}{\pi} \int_{line\, of\, sight} \frac{dx_1 (\Delta h - h'(x) \Delta x)}{\Delta h^2 + \Delta x^2}. \tag{5}$$

Integrovaná funkce je propagátor intenzity a popisuje, jak je intenzita přenášena z jednoho bodu do druhého. Text „line of sight“ (přímá viditelnost) u symbolu integrálu upozorňuje na nelineární omezení viditelnosti, integruje se pouze přes ty body $x_1$, které jsou viditelné




**Obrázek 2.** Schéma ablace povrchu sněhu. Rozptyl světla z bodu $x_1$ do intervalu mezi $x$ a $x + dx$ závisí na úhlu $d\theta$. Vektor **p** směřuje z bodu $x_1$ do bodu $x$ a vektor **dl** je normála na **p** tak, že $d\theta = dl/p$. Vektor **n** je normála k povrchu v bodě $x$ a vektor $d$**s** je inkrement podél povrchu mezi $x$ a $x + dx$ [6].

z bodu $x$, tedy mezi nimi mohou procházet paprsky světla.

Celkový model pak kombinuje rovnice světelné odrazivosti 5 a ablace 1 takto:

$$\frac{\delta h}{\delta t} = -\frac{\alpha I}{L} \mathscr{I}(x) + D \frac{\delta^2 h}{\delta x^2}, \tag{6}$$

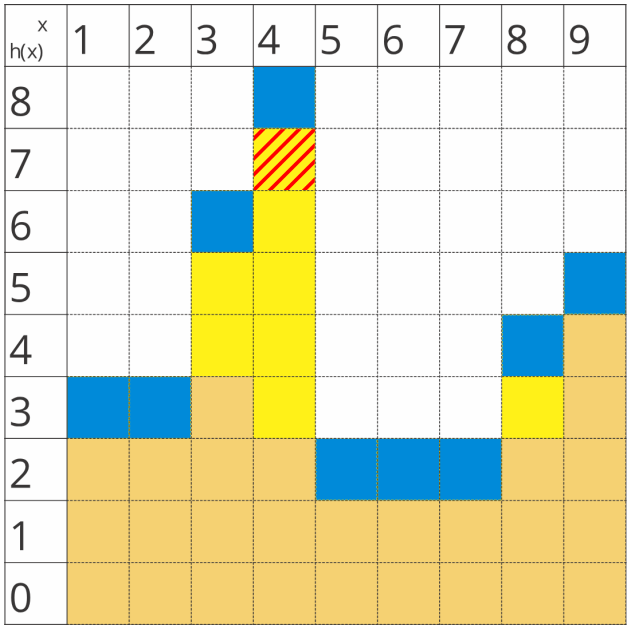
kde je definován integrál

$$\mathscr{I}(x) = \frac{1}{\pi} \int_{line\, of\, sight} \frac{dx_1 (\Delta h - h'(x) \Delta x)}{\Delta h^2 + \Delta x^2}. \tag{7}$$

## 4. Implementace jednorozměrného modelu

Výpočet předchozích rovnic je klíčovou částí simulace, v každém kroku je pro všechny body povrchu určena rychlost klesání. Délka kroku je konstantní čas, jelikož se jedná o spojitou simulaci, a z ní bude vypočítána změna výšky povrchu v bodě za jeden krok simulace. Vstupem simulace je rastrový obrázek ve stupních šedi představující výškovou mapu ledové vrstvy.

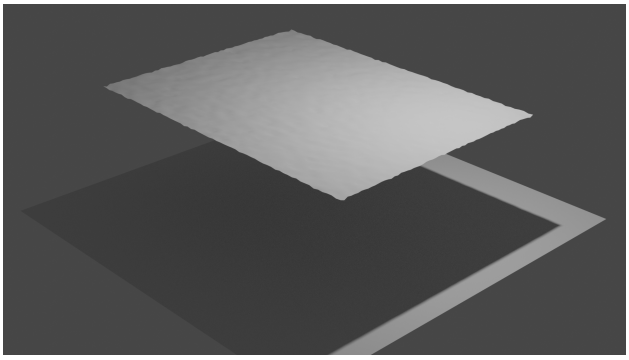
Jelikož je ale pro potřeby výpočtu nutné znát zejména povrch této struktury, je třeba vstupní pole výšek převést na pole, kde jsou uvedeny všechny body v takovém pořadí, kudy by vedla diskretizovaná funkce $y = h(x)$ daného řezu. Bodem je zde myšlen vzorek této funkce, který si lze pro potřeby programu představit jako kvádr odpovídající voxelu v trojrozměrném objektu, jehož délka a šířka je rovna vzdálenosti mezi 2 sousedními vzorky. Výška kvádru je 1. Program postupně prochází řezem povrchu (může se jednat o řádek nebo sloupec v mapě) a ukládá souřadnice těchto bodů. Bod je jednoznačně definován svou souřadnicí $x$ a $y$. Popis povrchu pomocí bodů vysvětluje obrázek 3.




**Obrázek 3.** Schéma řezu povrchem. Modré a žluté body odrážejí světlo, oranžové jsou uvnitř ledu. Modré body jsou uvedeny ve výškové mapě, žluté body je třeba dopočítat. Při průchodu tímto povrchem je bod s červeným šrafem projit dvakrát, přičemž pokaždé má jiné sousední body a jiné body, které do něj odráží světlo.

S povrchem souvisí ještě další potřebný údaj pro výpočty simulace, a to je normála k povrchu v daném bodě. Normála je přímka kolmá na tečnu, která prochází daným bodem. O tečně je známo, že její směrnice je rovna hodnotě derivace funkce v bodě dotyku. Pro program bude nutné znát normálu jako vektor kolmý na povrch, který směřuje od povrchu do prostoru, kde je vzduch. Zde se opět projevuje výhoda a nutnost ukládání bodů vysvětlena v předchozí části, protože bod na obrázku 3 značený šrafem mající dvě nesouvislé plochy na povrchu bude muset mít i dvě různé normály pro svůj popis. Je třeba zajistit, že lokální maxima a minima budou mít derivaci rovnou 0 a tedy vodorovné tečny. Oproti vzorci je třeba počítat i se situací, že tečna v bodě bude svislá.

V následující části je třeba pro každý bod $x$ povrchu určit ty body, ze kterých na daný bod dopadá odražené světlo. Tyto body jsou označované jako body viditelné z daného bodu. Viditelné body musejí splňovat několik podmínek. Předně pro vytvoření i následné přijmutí odrazu musejí být k sobě přivrácené. Pokud je paprsek světla definovaný jako vektor **p** směřující z viditelného bodu $x_1$ do $x$, výsledek skalárního součinu vektoru **n** kolmého k povrchu v bodě $x$ a vektoru **p** musí být menší, než 0. Jen tak dopadne paprsek na bod z vnějšku, nikoliv z vnitřku ledovce. Pravidlo platí obdobně i pro bod $x_1$, jelikož v tomto případě vektor **p** míří od bodu, skalární součin musí být větší, než 0. I pokud jsou body k sobě přivrácené, může se mezi nimi nacházet překážka, která bude paprsek blokovat. Byl vytvořen jednoduchý systém predikce, zda je bod viditelný, skrytý, nebo je viditelnost nejistá. Body v řezu, u nichž je zkoumaná viditelnost, jsou procházeny ve směrech od bodu $x$, zprvu je předpokládána jejich viditelnost a testuje se, zda jsou k sobě přivrácené. V případě, že nejsou, není jistá viditelnost následujícího bodu, a kromě přivrácenosti se bude testovat i zda paprsek vyslaný z jednoho bodu dorazí do druhého, nebo narazí na překážku. Bod vyšší než $x$, kde povrch ve směru od bodu $x$ klesá, značí překážku, která bude stínit minimálně všechny body nižší než výška tohoto bodu. Body vyšší bude poté také potřeba testovat vysíláním paprsků. Pokud je při vysílání paprsků nalezen viditelný bod, předpokládá se, že následující bod bude rovněž viditelný, protože se neočekává omezení viditelnosti shora třeba stropem dutiny, kterou nelze výškovou mapou reprezentovat. Ve chvíli, kdy jsou všechny body v řezu rozdělené na viditelné a neviditelné, je pro daný bod vypočítána rychlost klesání povrchu, po určení rychlosti všech bodů je proveden krok simulace. Výsledek je vygenerován opět výškovou mapou, která je převedena do 3D objektu a renderována v programu Blender.

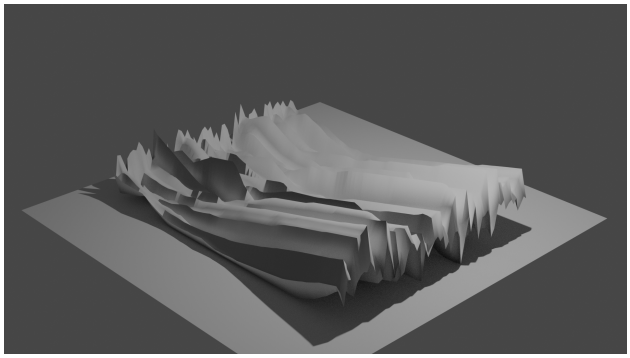


**Obrázek 4.** Vstup pro simulaci, plocha s drobnými prohlubněmi generovanými šumem pro odstartování tvorby.

Na základě vstupu zobrazeného na obrázku 4, což je plocha s jemnými propady vygenerovanými pomocí šumu, produkuje tento model výsledek zachycený na obrázku 5. Zde je vidět hlavní problém využití jednorozměrného modelu v plošném prostředí. Jednotlivé řezy jsou na sobě nezávislé a proto se sousední řezy mohou lišit do takové míry, že model netvoří realistický povrch.

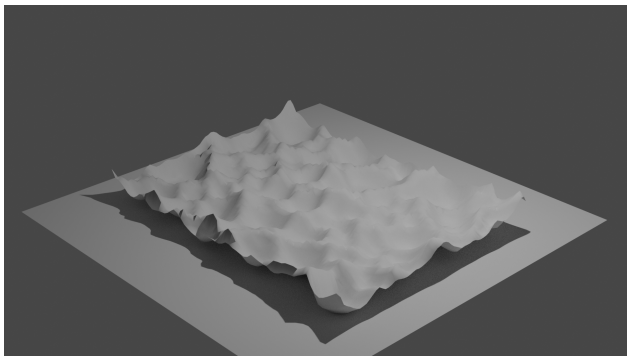
## 5. Navržený plošný model

Vzhledem k neuspokojivým výsledkům jednorozměrného modelu v ploše bylo rozhodnuto vylepšit původní



**Obrázek 5.** Výsledek simulace po 24 dnech podle původního modelu. Mezi nezávislými řádky vznikají obrovské rozdíly.

model prováděním výpočtů změn výšek i ve sloupcích mapy, jelikož samotné řádky mezi sebou nemají závislosti, což vede k nepřesnosti simulace a výsledný terén netvoří realistické struktury. Tím, že jsou v jednom kroku provedeny průchody všech řádků a následně i všech sloupců, vzniká jednak závislost mezi body mimo jeden řádek a zároveň se zvětšuje obor bodů plochy, které jsou na viditelnost zkoumané. Tento přístup stále není přesný reálnému prostoru, kde by měly být na viditelnost zkoumány úplně všechny body, proto by ale muselo dojít ke kompletní změně navrženého a implementovaného modelu.



**Obrázek 6.** Výsledek simulace po přidání průchodu sloupci. Mnohem reálnější tvary terénu.

Obrázek 6 zachycuje výsledek modelu po průchodu řádků i sloupců z počátečního stavu na obrázku 4, tedy stejného, jaký měl výsledek na obrázku 5. I tato zjednodušená varianta zohlednění bodů mimo vodorovné řezy tvoří značně realističtější povrch pole kajícníků. Tato úprava přitom znamenala pouze menší zásah do implementačního modelu, protože k její realizaci bylo možné využít funkce již implementované pro jednorozměrný model.

Časová náročnost výpočtu obou modelů závisí zejména na velikosti vstupní plochy, tedy počtu pixelů (bodů) ve výškové mapě. Pro soubor o rozměrech $200 \times 100px$ trvá výpočet 24 kroků simulace, přičemž dt=1 hodina, zhruba 62 sekund pro jednorozměrný model, vylepšený plošný model na stejný výpočet potřebuje 105 sekund. To může znamenat problém pro větší plochy, nicméně je zde poměrně velký potenciál v paralelizaci výpočtů, protože rovnoběžné řezy jsou na sobě nezávislé a lze tedy jejich výpočty provádět současně na více jádrech. Paralelizace kódu v místech výpočtu výšky bodů v řezu byla realizována s pomocí OpenMP. Paralelizovaný výpočet kroku při stejných parametrech trvá na čtyřjádrovém procesoru použitém k testování 35 sekund.

## Závěr

Cílem práce bylo vytvořit implementační model vzniku ledovcových struktur nazývaných „kajícníci“. Nejprve byly shrnuty známé informace o tomto jevu, dosavadní výzkumy jejich vzniku. Taktéž byl popsán jednorozměrný matematický model tohoto chování spočívající ve výpočtech změn výšky bodů podle viditelných bodů v řezu terénu, který vytvořila M. D. Betterton. Tento model byl implementován v jazyce C++ a bylo testováno jeho chování. Podle předpokladů tento model není dostačující pro simulaci kajícníků v ploše. Následně byl proto tento model rozšířen o výpočet výšky bodu ve dvou na sebe kolmých řezech, který vykazuje oproti původnímu modelu věrohodnější tvary výsledných struktur.

## Poděkování

Chtěl bych poděkovat svému vedoucímu práce doc. Ing. Martinu Čadíkovi, Ph.D. za vedení, pomoc a rady v průběhu práce na tomto projektu.

## Literatura

[1] Catherine Brahic. Spiky glaciers are slower to melt. https://www.newscientist.com/article/dn11309-spiky-glaciers-are-slower-to-melt

[2] Paul Scott Anderson. Europa may have towering ice spikes on its surface. https://earthsky.org/space/jupiters-moon-europa-penitentes-ice-spike

[3] Louis Lliboitry. *The Origin of Penitents*. Journal of Glaciology, s. 2:331–338, 1954.

[4] Charles Darwin. *The Voyage of the Beagle*. 1839.

[5] V. Bergeron, Ch. Berger, and M. D. Betterton. *Controlled Irradiative Formation of Penitentes*. 2006.

[6] Meredith D. Betterton. *Formation of Structure in Snowfields:Penitentes, Suncups, and Dirt Cones*. Physical Review E, vol 63, 2001.

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

# Pokročilá evaluace privátnosti na sociálních sítích

Filip Januš*

**Abstrakt**
V dnešní době stále přetrvává trend přesunu mezilidské komunikace do online prostředí. A to díky sociálním sítím a službám jimi poskytovanými. S tímto faktem souvisí i rostoucí počet uživatelů sociálních sítí. Mnoho uživatelů ovšem nevnímá rizika spojená s přítomností v internetovém prostředí. Tato práce se zaměřuje na analýzu bezpečnostních nastavení uživatelských účtů sociálních sítí a následné vyhodnocení tohoto nastavení. Cílem práce je vytvořit nástroj poskytující možnost vyhodnotit bezpečnostní nastavení uživatelského účtu na sociální síti případně doporučit vhodnější nastavení s ohledem na soukromí uživatele. Aby bylo možné dosáhnout těchto cílů, je potřebné použít vhodný model provádějící vyčíslení skóre privátnosti. Výstupem práce bude návrh a implementace nástroje provádějící analýzu, vyhodnocení a doporučení, jak vylepšit své nastavení soukromí na sociální síti. Což by mělo pomoci uživateli omezit množství uniklých citlivých informací.

**Klíčová slova:** Sociální síť — Soukromí — Bezpečnost — Skóre soukromí — Nastavení soukromí

**Přiložené materiály:** *N/A*

*xjanus08@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

## 1. Úvod

V dnešní době stále populárnějších sociálních sítí, se snadno stává, že bezpečnost soukromích informací na těchto sítích je až druhořadý nebo vůbec neřešený problém. Řada uživatelů má malé nebo žádné povědomí o možnostech nastavení a ochrany osobních údajů na sociálních sítích. Taktéž sociální sítě neposkytují uživatelům zpětnou vazbu na jejich nastavení a nakládání se svým soukromím. Nezbytnost řešit tyto problémy a poukazovat na ně dokazuje několik publikací [1, 2] nebo nedávno zveřejněný snímek "V síti".

Výzkumy ukazují, že díky klasifikačním technikám lze efektivně odhalit citlivé informace uživatelského profilu sociální sítě na základě ne mnoha informací, v kombinaci se znalostí příslušnosti do různých skupin nebo v kombinaci se znalostí o účasti v různých aktivitách na sociálních sítích.

Hlavním cílem práce je analyzovat možná nastavení soukromí na sociálních sítích. Tato nastavení vyčíslit pomocí modelů, aby bylo možné vyhodnotit získané informace o nastavení soukromí uživatele. Dále dát uživateli zpětnou vazbu na základě jeho nastavení a doporučit mu nápravná opatření. Přičemž informace budou získávány dvěma způsoby. Interním, při kterém informace o nastavení budou staženy přímo z přihlášeného účtu a nebo externí, při kterém budou informace o uživateli získány z veřejně dostupných zdrojů (vždy pro konkrétní sociální síť).

## 2. Existující práce

Již několik desítek prací bylo zaměřeno na oblast bezpečnosti na sociálních sítích. Tyto práce se tématem zaobíraly z několika různých pohledů. V [1] se autoři zabývají dolováním citlivých dat z publikovaných informací na sociálních sítích, definují zde anonymitu a popisují různé přístupy dolování dat a zabývají se možnými riziky vyzrazení citlivých informací. Jedna z prvních prací zaměřených na internetové sociální sítě [3] si kladla za cíl změřit úroveň soukromí, cílem měl být nástroj, který každému uživateli řekne, jakého skóre soukromí dosahuje a popřípadě doporučí nastavení. Je třeba upozornit, že autoři chtěli evaluovat skóre soukromí skupině uživatelů na základě pravděpodobnostních modelů.

Wang a jeho kolegové [4] se zaměřovali na vyčíslování soukromí mezi dvěma uživateli, dle nastavených atributů, prezentovali již dříve navrhované modely a následně je dále vylepšovány, tak aby lépe

reflektovaly uživatelská nastavení.

V publikaci [5] se autoři věnovali vývoji aplikace zobrazující procentuální míru ohrožení soukromí uživatele sociální sítě. Tato aplikace získává data o uživateli z API sociální sítě Facebook. Autoři se zde rozhodli vyčíslovat ohrožení privátnosti na základě informací o vztazích s ostatními uživateli a na základě dostupných informacích o přátelích. Taktéž Becker a Chen [6] vyvíjeli nástroj na detekci potencionálních úniků soukromých informací, podobně jako v [5] pracovali s informacemi o vazbách mezi uživateli a na základě nich se snažili odhalovat soukromé atributy profilů. Z výsledků vyplývá, že při testování nástroje na 93 účastnících se podařilo odhalit necelých 60% soukromých informací. Všechny popsané přístupy se snaží na základě získaných informací odhadovat aktuální nastavení a následně doporučovat nastavení.

V článku [2] autoři popisují možná bezpečnostní rizika sociálních sítí, několik vybraných popisují detailněji, nejvíce prostoru je věnováno aplikacím nazývaným *trojan applications*. Účelem těchto aplikací je především poskytnout jejich provozovateli informace z profilů uživatelů. Na závěr je zde navrhnuto také několik protiopatření. Naopak v práci [7] autoři zkoumají uživatelské povědomí o rizicích spojených se sociálními sítěmi. Dále se zde objevuje téma, kde jsou popsána některá úskalí nastavení zabezpečení a možné rozdíly mezi požadovaným a reálným nastavením.

## 3. Motivace

Cílem této práce je prostudovat možná nastavení soukromí a zabezpečení na různých sociálních sítích. Informace o nastavení vhodným způsobem agregovat a na základě navržených technik varovat uživatele před přílišnou důvěrou sociálním sítím potažmo internetu. Výsledkem by se měl stát nástroj, kterému uživatel svěří své přihlašovací údaje k sociálním sítím, ten provede kontrolu nastavení z pohledu přihlášeného uživatele a dále zkontroluje viditelné údaje z pohledu nepřihlášeného uživatele. Na základě těchto informací se pomocí matematických modelů provede výpočet míry ohrožení soukromí, dle dosaženého výsledku bude uživateli sděleno jak si stojí a popřípadě může být doporučeno, která nastavení upravit.

Zatímco výše popisované práce pracovaly vždy s informacemi poskytovanými prostřednictvím API sociálních sítí nebo s informacemi veřejně dostupnými (členství ve skupinách, různé aktivity, spojení s přáteli), aplikace navrhovaná v této práci, pracuje s informacemi získanými přímo z nastavení profilu, díky čemuž lze dosahovat podstatně přesnějších a detailnějších výsledků než v předchozích pracích. Tato skutečnost je dána především skutečností, kdy API sociálních sítí neposkytují informace o nastavení zabezpečení a soukromí účtu.

Další předností navrhované práce je také podpora několika sociálních sítí, zatím co předešlé práce se zaměřovaly pouze na jednu specifickou síť.

## 4. Sociální sítě

Sociální síť lze popsat jako internetovou službu umožňující svým uživatelům vytvářet vlastní profily, sdílet informace, videa, fotografie, komunikovat, provozovat chat a mnoho dalších aktivit [8, 9]. Existuje celá řada sociálních sítí, které lze dělit dle obsahu, který uživatelé sdílí. Dělení lze provádět také podle lokality, některé sociální sítě jsou specifické jen pro svůj region, například v Číně existuje řada sociálních sítí, které jsou téměř neznámé v Evropě.

Z matematického pohledu lze sociální síť považovat za graf, kde vrcholy jsou reprezentovány entitami a hrany vztahy mezi nimi [4]. Existují vztahy orientované či neorientované stejně jako v grafu hrany. U neorientovaných vztahů buď to vazba existuje či nikoli. U orientovaných se na vztahu podílí pouze jeden uživatel. Příkladem může být situace, kdy entita A zná entitu B, ale B nezná A. Většina sociálních sítí funguje na modelu přátelství, kdy vztah entit je obousměrný. Tj. entita A zná B a B zná A.

Důležitým pojmem v kontextu vztahů na sociálních sítích je stupeň separace [4]. Pojem úzce souvisí s viditelností atributů entity. Definuje se jako *funkce h* určující počet kroků mezi entitou $A_i$ a $A_j$.

$$d_{ij} = h(A_i, A_j) \tag{1}$$

V závislosti na stupni separace se v modelech založených na přátelství rozlišují 3 skupiny přátel, tj. přátelé (hodnota *funkce h* rovna 1), přátelé přátel (hodnota *funkce h* rovna 2) a ostatní neboli veřejní, u kterých nabývá *funkce h* hodnoty 3 a více. Do poslední skupiny se řadí i ty entity, které mezi sebou nemají žádnou vazbu.

Z pohledu zabezpečení soukromých informací uživatele nabízí sociální sítě možnosti ovlivnit zveřejňované informace. Spektrum možných nastavení se různí napříč sociálními sítěmi. Od desítek možných kombinací nastavení až po malé jednotky. Například sociální síť Facebook umožňuje uživateli konfigurovat sedmnáct různých nastavení. Naproti tomu sociální síť Tumblr disponuje pouze 3 nastaveními. Všechny sociální sítě poskytují uživateli možnosti kontrolovat své aktivity na síti, také dovolují uživateli stáhnout veškerá data, které uživatel síti poskytl. A to včetně chatových konverzací a záznamů všech akcí na sociální

síti. Mezi akce lze zařadit příspěvky označené jako *To se mi líbí* v případě Facebooku. Žádná z analyzovaných sítí ovšem neposkytuje jakoukoli formu upozornění, zda zveřejnění určitého atributu má či nemá dopad na soukromí uživatele.

## 5. Privátnost a její vyčíslení

V průběhu této kapitoly bude popsána metrika privacy score. Dále několik modelů navržených autory v pracích [4], [10] a [3], které na základě údajů o nastavení účtů dokáží tuto metriku vyčíslit.

### 5.1 Privacy score

Aby bylo možné vhodným způsobem zpracovávat úroveň soukromí je potřeba mechanismus, který umožní tuto úroveň vyčíslit. Podobné mechanismy již spolehlivě fungují v různých odvětvích komerčního sektoru. Z toho důvodu byla jedna z metod převzata [3]. Metoda vychází z již fungujících technik pro určování skore entity. Tyto techniky se používají například v bankovnictví, kde se určuje bonita nebo důvěryhodnost klienta na základě jeho vlastností. Tato metoda se v kontextu soukromí na sociálních sítích nazývá Privacy score. Indikuje potenciální nebezpečí pro soukromí entity. Platí, že čím vyšší privacy score tím výšší riziko.

Evaluace privátnosti/soukromí není zcela triviální disciplínou. Již definice privátnosti může být problematická neboť je subjektivní a různí lidé mohou chápat privátnost odlišně. Také váha jednotlivých vlastností soukromí může být napříč populací různá. Například jeden člověk považuje telefonní číslo za velmi citlivou informaci, zatím co jiný může považovat tuto informaci za naprosto nepodstatnou.

Ovšem z druhé strany, ač se mohou uživateli zdát některé z vlastností nepodstatné, opak může být pravdou. Toto dokazují například publikace [10, 2], kde jsou jednoznačně definovány některé atributy, které mají značný dopad na soukromí uživatele bez ohledu na subjektivní vnímání.

Na základě těchto znalostí vznikla řada přístupů a modelů pro evaluaci privátnosti a výpočet privacy score.

### 5.2 Virtuální atribut

Jak bylo popsáno výše existují vlastnosti/atributy, které mají dopad na soukromí, i když se mohou zdát nepodstatné. V některých případech se jedná o tzv. virtuální nebo také kompozitní atribut, který se skládá z několika jiných atributů. Kombinace několika zdánlivě nepodstatných atributů může vést k odhalení podstatné části soukromí uživatele. Například v práci [10] se poukazuje na skutečnost, že 87% obyvatel Ameriky lze identifikovat na základě poštovního směrovacího čísla, pohlaví a data narození. Ačkoliv každý z těchto atributů samostatně pro soukromí nepředstavuje značné riziko dohromady mohou vést k unikátní identifikaci jedince.

### 5.3 Popis entity

Aby bylo možné jednoduše pracovat s jednotlivými položkami entity při evaluaci privátnosti v rámci OSN (Online social network), používá se u většiny modelů popis založený na maticích tzv. Odpovědní matice [3].

Přínos tohoto zápisu spočívá v jednotném přístupu k popisování vlastností entity v OSN napříč různými výpočetními modely. Pro evaluaci konkrétní hodnoty atributu konkrétní entity bude v rámci práce používán následující zápis: R(i,j)=x, kde R je odpovědní matice, i je uživatel (řádek matice), j je konkrétní vlastnost (sloupec matice) a x je příslušná hodnota z odpovědní matice. Příkladem může být interpretace: uživatel i je ochotný vlastnost j sdílet s ohledem na x, kde x udává míru ochoty tuto informaci sdílet. Například při použití dvoustavového nastavení 0 označuje neochotu sdílet tuto informaci, na druhé straně hodnota 1 udává ochotu informaci sdílet.

### 5.4 Model citlivosti a viditelnosti

Prvním zástupcem modelů evaluace je základní model citlivosti a viditelnosti [3] využívaný i dalšími pokročilejšími modely. Základními složkami pro výpočet privacy score tímto modelem jsou:

- citlivost nebo-li privátnost vlastnosti **i** je označována jako $\beta_i$. Problematikou nastavení vah jednotlivých atributů se zabývá práce [10], z které vychází tabulka 1, kde jsou zobrazeny váhy vybraných atributů. Pro tento výpočetní model platí, že se zvyšující se citlivostí vlastnosti **i** roste také privacy score entity.
- viditelnost vlastnosti **i** entity **j** se značí V(i,j), kde funkce V je definována pomocí stupně separace. Viz definice níže. Platí, že s rostoucím počtem uživatelů se kterými je informace o vlastnosti sdílena, roste i privacy score entity.

$$V(i,j) = \left\{ \begin{array}{ll} 0, & \text{pokud i vidí pouze sám} \\ 1, & \text{pokud i vidí přátelé} \\ 2, & \text{pokud i vidí přátelé přátel} \\ 3, & \text{pokud i vidí všichni} \end{array} \right\} \quad (2)$$

Na základě výše definovaných vlastností se privacy score definuje jako monotonně rostoucí funkce dvou parametrů, citlivosti (privátnosti) a viditelnosti informací o entitě. Příkladem poukazující na důležitost

| | Váha atributu[%] |
|---|---|
| Jméno | 15 |
| Vzdělání | 15 |
| Stav | 25 |
| Rodinní příslušníci | 25 |
| Pohlaví | 25 |
| Město | 45 |
| Stát | 45 |
| Fotky | 45 |
| List přátel | 60 |
| E-mail | 65 |
| Domovské město | 65 |
| Navštívená místa | 65 |
| Datum narození | 65 |
| Telefonní číslo | 70 |
| Aktuální pozice | 80 |
| Rodné číslo | 90 |

**Tabulka 1.** Tabulka citlivostí atributů (Zdroj:[10])

parametru citlivosti může být scénář, kdy entita **j**, v tomto případě uživatel sdílí dvě osobní informace, telefonní číslo x a vzdělání y. Situace R(x,j) = 1 && R(y,j) = 0 je mnohem nebezpečnější z pohledu citlivosti sdílených informací než R(x,j) = 0 && R(y,j) = 1. V tomto případě i když velká skupina lidí bude znát vzdělání uživatele **j** není to stejné jako kdyby stejná skupina lidí znala jeho telefonní číslo.

Privacy score entity j je vypočítáno na základě následujícího vztahu:

$$PR(j) = \sum_{i=1}^{n} PR(i,j) = \sum_{i=1}^{n} \beta_i * V(i,j) \qquad (3)$$

### 5.5 PIDX (Privacy Index)

Dalším z modelů vyčíslující soukromí entity resp. privacy score je PIDX [4]. Měří úroveň publicity jedné entity vzhledem k jiné. Model PIDX pracuje se třemi metrikami: známé atributy, jejich citlivost a viditelnost. Na základě kombinací těchto metrik se určí míra ohrožení soukromí entity. Dle zvolené kombinace a kombinačního přístupu k metrikám se rozlišují tři typy: w-PIDX váhovaný PIDX, m-PIDX (maximum PIDX) a c-PIDX (composite PIDX).

Aby byla brána v potaz citlivost atributu, je každému atributu přidělen PIF (privacy impact factor). PIF je numerická hodnota mezi 0 a 1, kde 1 znamená maximální citlivost atributu. Pro výpočet PIF se použije vzorec 4, kde **i** značí konkrétní atribut a $W_{max}$ maximální hodnotu citlivosti, konkrétní hodnoty lze nalézt v dříve prezentované tabulce vah 1.

$$PIF(i) = \frac{W(i)}{W_{max}} \qquad (4)$$

Všechny varianty PIDX pracují s viditelností atributů resp. se stupněm separace a s výše definovaným PIF. Jednotlivé metody se liší pouze postupem výpočtu.

| | Název nastavení | Nastavení | Viditelnost | Váha[%] |
|---|---|---|---|---|
| 1 | Web & App Aktivita | Zapnuto | 1 | 65 |
| 2 | Historie polohy | Pozastaveno | 0 | 65 |
| 3 | YouTube historie | Zapnuto | 1 | 60 |
| 4 | Kontakty z interakcí | Zapnuto | 1 | 60 |
| 5 | Kontakty ze zařízení | Pozastaveno | 0 | 60 |

**Tabulka 2.** Příklad nastavení účtu Google

Index privátnosti PIDX je definován jako míra vyzrazení soukromí entity $A_j$ směrem k entitě $A_i$. V rámci této práce bude vždy zkoumán jeden konkrétní účet sociální sítě vůči okolí, tím pádem lze zápis funkcí modelu mírně zjednodušit oproti definicím uvedeným v [10]. Funkce PIDX nabývá hodnot z intervalu $< 0, 100 >$. Vysoká hodnota PIDX znamená vysoké prozrazení soukromých informací entity.

Nechť existuje množina $S = \{s_1, s_2, ..., s_n\}$ obsahující PIF váhy pro jednotlivé atributy a vektor $V = (v_1, v_2, ..., v_n)$ obsahující hodnoty viditelnosti, které odpovídají jednotlivým atributům entity. Jak bylo zmíněno výše existují 3 různé varianty indexu privátnosti PIDX:

1. w-PIDX vyčísluje privátnost vztahem

$$w - PIDX(V,S) = \frac{\sum_{j=1}^{n} V(j)s_j}{\sum_{j=1}^{n} s_j} \qquad (5)$$

2. m-PIDX měří maximální možné odhalení privátnosti entity $A_j$ směrem k $A_i$

$$m - PIDX(V,S) = \max(V(1)s_1, .., V(n)s_n) \qquad (6)$$

kde funkce max vrací ze zadaných hodnot tu maximální.

3. c-PIDX nebo-li kompozitní PIDX, jak napovídá název tato metoda v sobě kombinuje dva předchozí přístupy w-PIDX a m-PIDX, což může být zapsáno jako:

$$c - PIDX(V,S) = m - PIDX(V,S) + \qquad (7)$$

$$(100 - m - PIDX(V,S)) \cdot \frac{w - PIDX(V,S)}{100}$$

Zatímco, w-PIDX reflektuje inkrementální změny atributu, m-PIDX se hodí spíše pro hodnocení aktuálního soukromí, těchto dvou vlastností využívá poslední zástupce c-PIDX a snaží se kombinovat výhody obou přístupů [4].

Příkladem použití může být aplikace modelu C-PIDX na výchozím nastavení Google účtu, které je zobrazeno v tabulce 2. Pro výpočet privacy score je zapotřebí množina $V_i$, která je reprezentována sloupcem

viditelnost a množina vah $S_i$. Váhy se vypočtou dle vztahu 4. Například pro nastavení *Historie polohy*:

$$PIF(2) = \frac{60}{65} \tag{8}$$

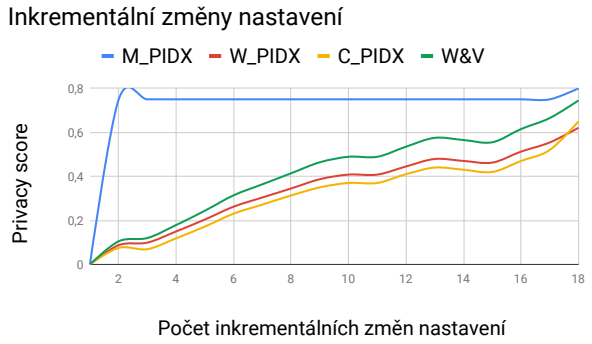
Dále se již jen dosadí známé hodnoty do rovnic 5, 6, 7, čímž se získá výsledné privacy score modelem C-PIDX.

## 6. Výběr modelu

Aby bylo možné vybrat co nejvhodnější model pro vyčíslení privátního skóre, bylo provedeno testování a vyhodnocení chování modelu v různých situacích.

### 6.1 Inkrementální změny

V rámci prvního přístupu je sledováno chování modelu při inkrementálních změnách atributů. Inkrementální změnou atributů je myšleno postupné přidávání resp. zveřejňování jednotlivých atributů. Jinými slovy: test $T_1$ zveřejňuje informace jednoho atributu, $T_2$ informace dvou atributů až $T_n$. Počet testů v rámci diskutovaného přístupu je roven počtu nastavovaných atributů. Tyto atributy byly označeny $T_1$ až $T_{17}$, společně s příslušnými váhami jsou zobrazeny v tabulce 3. Zdrojem tabulky jsou bezpečnostní nastavení sítě Facebook. Protože v rámci testování vlastností modelů je význam jednotlivých položek nastavení irelevantní, byly názvy pominuty. Testování tohoto přístupu bylo postupně provedeno na všech čtyřech prezentovaných modelech a výsledky byly vyneseny do grafu. Díky tomu, že modely pracují v různých intervalech, bylo nutné hodnoty před vynášením do grafu normalizovat.




**Obrázek 1.** Výsledek testování inkrementálních změn

Tento postup byl již použit pro modely třídy PIDX v práci [4]. V rámci této práce byl výsledek ověřen a přidán jeden další model. Na základě naměřených hodnot byly potvrzeny dříve publikované závěry a to, že model W-PIDX dobře reflektuje inkrementální změny, ale má jisté potíže s reflektováním aktuálně

| Položka | váha[%] | Položka | váha[%] |
|---|---|---|---|
| T1 | 25 | T10 | 80 |
| T2 | 60 | T11 | 80 |
| T3 | 50 | T12 | 20 |
| T4 | 50 | T13 | 30 |
| T5 | 0 | T14 | 40 |
| T6 | -10 | T15 | 80 |
| T7 | -10 | T16 | 25 |
| T8 | 80 | T17 | 15 |
| T9 | 20 | | |

**Tabulka 3.** Testovací data vycházející z výchozího nastavení sítě Facebook

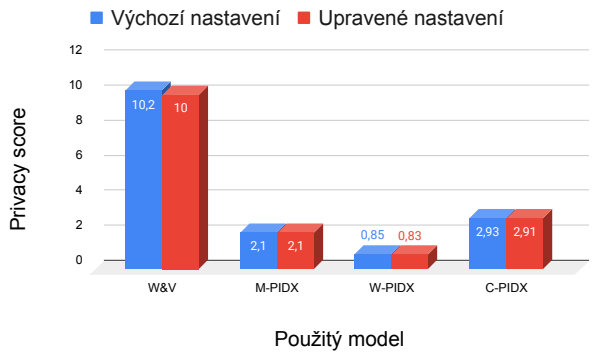
zveřejněného atributu. Dále bylo vypozorováno, že W-PIDX má tendenci vyhlazovat skokové změny. Naproti tomu model M-PIDX dobře odráží skokové změny, ale neumí pracovat s inkrementálními změnami atributů. Což je vidět na obrázku 1, kdy většina výstupních hodnot modelu je stejná až na skokové změny. Výhody obou těchto modelů by měl kombinovat model C-PIDX. Dobře reflektuje inkrementální změny a také reflektuje změny aktuálního atributu. Tento fakt lze vidět na obrázku 1 a to zejména v okolí třetího a sedmnáctého testu.

Posledním testovaným modelem je model váhy a viditelnosti zobrazený zeleně na obrázku 1. Model se chová velice podobně jako W-PIDX a taktéž má problém se skokovými změnami.

### 6.2 Reflektování nastavení zlepšující skóre

V možnostech nastavení některých sociálních sítí se objevují nastavení, která jdou protichůdným směrem oproti klasickým nastavením atributů. Za klasické nastavení atributu se považuje situace, kdy se nastavuje, zda atribut sdílet či nikoli, tedy při sdílení s okolím má různě velký negativní dopad na soukromí. Naopak tato protichůdná nastavení by měla aktivací soukromí uživatele chránit. Příkladem takového nastavení (tab. 3 položka $T_7$) je v síti Facebook: *Chcete kontrolovat příspěvky, ve kterých vás někdo označil, než se budou moct zobrazit na vaší timeline?*. Nastavení umožňuje uživateli ovlivňovat a kontrolovat co se mu zobrazí na profilu, resp. provádět "cenzuru" svého profilu. V rámci práce jsou tato nastavení považována za přínosná a mohou mírně pozitivně ovlivnit soukromí. Aby bylo docíleno pozitivního přínosu nastavení, je jeho váha nastavena na zápornou hodnotu.

Podstatným kriteriem při výběru vhodného modelu pro nástroj je schopnost reflektovat výše popsané nastavení. Pro testování tohoto scénáře byly nejprve aplikovány všechny prezentované modely na výchozí nastavení sítě Facebook, kde jsou nastavení $T_6$ a $T_7$ z tabulky 3 vypnuty. Poté bylo přidáno nastavení: *Chcete kontrolovat příspěvky, ve kterých vás někdo*




**Obrázek 2.** Výsledek testování nastavení s pozitivním efektem

*označil, než se budou moct zobrazit na vaší timeline?* ( $T_7$ ). Nad takto upravenou konfigurací nastavení byly opět provedeny výpočty všech podporovaných modelů. Výsledky jsou graficky znázorněny na obrázku 2.

Lze si zde všimnout, že model M-PIDX pravděpodobně nesplňuje požadavky na reflektování popisovaných nastavení, jelikož model v obou případech dojde ke stejné hodnotě privátního skóre. U ostatních modelů lze vidět mírné zlepšení (snížení privátního skóre) po aplikování výše diskutovaných nastavení.

### 6.3 Vyhodnocení

Jako výchozí model byl pro tuto práci zvolen C-PIDX. Jelikož inkrementální testování potvrdilo předešlé výsledky publikované v [4] a také ukázalo, že ani model váhy a viditelnosti neposkytuje lepší výsledky z pohledu reflektování skokových změn.

Při hodnocení modelů na základě nastavení, která by měla pozitivně ovlivňovat privátní skóre, se vyloučil model M-PIDX a ostatní modely se chovaly velice podobně.

Byl také brán zřetel na závěry autorů modelů PIDX, kteří uvádí v [4], že C-PIDX nejlépe z navrhovaných modelů reflektuje kompozitní atributy.

## 7. Návrh řešení

Navrhovaný systém by se měl skládat ze tří hlavních částí, z extraktoru, evaluatoru a modulu vyhodnocení. Vstupním bodem do nástroje bude extraktor, v závislosti na jeho výstupu bude pracovat evaluator, což bude stěžejní komponenta celého nástroje, neboť bude na základě dostupných informací a za pomoci dříve vybraného modelu provádět vyčíslování skóre soukromí. Poslední v řetězci bude modul vyhodnocení prezentující výsledky uživateli. Kompletní schéma systému je zobrazeno na obr. 3.

Nástroj bude podporovat dva základní režimy: Interní a externí. V interním režimu aplikace provede přihlášení na sociální síť, zde vyhledá a extrahuje informace o nastavení soukromí, které jsou následně vyhodnoceny a prezentovány uživateli. Externí přístup zjišťuje informace o uživateli z pohledu třetí osoby a na základě zjištěných informací se dedukuje aplikované nastavení účtu.

Prezentace výsledku spočívá v zařazení uživatele do skupiny s obdobným nastavením soukromí účtu. Uživateli bude také poskytnuta možnost nechat si poradit, které položky nastavení upravit za účelem zlepšení zabezpečení soukromí.

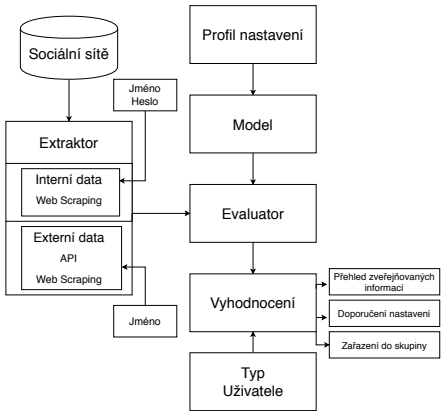
Cílové řešení by měla představovat aplikace, které uživatel poskytne přihlašovací údaje k účtu sociální sítě. Aplikace se přihlásí, zjistí aktuální nastavení zabezpečení a soukromí. Tyto informace vyhodnotí a sdělí uživateli procentuální míru ohrožení jeho citlivých informací. V případě nespokojenosti uživatele s výsledkem mu bude nabídnuta pomoc ve formě jednoduché nápovědy. Tato nápověda by měla uživateli sdělit, které položky nastavení představují největší hrozbu pro jeho soukromí. A uživatel se již sám bude moci rozhodnout, zda nastavení upraví či nikoli.

## 8. Implementace a testování

Navrhovaný nástroj byl prozatím implementovaný jako konzolová aplikace s ohledem na plánované rozšíření o grafické rozhraní.

Aplikace aktuálně podporuje funkcionalitu, která byla popsána v 7, přičemž se používá vybraný model C-PIDX. Momentálně je plně podporována sociální síť Facebook. Tj. na základě přihlašovacích údajů poskytnutých uživatelem, provede extrakci informací z jeho účtu, evaluuje nastavení, sdělí uživateli míru ohrožení, případně doporučí, která nastavení by šla upravit tak, aby bylo soukromí uživatele více chráněno.

Pro implementaci byl zvolen jazyk Python jelikož disponuje širokou škálou modulů umožňujících práci s webovými stránkami. Taktéž podporuje framework




**Obrázek 3.** Navrhovaná architektura systému

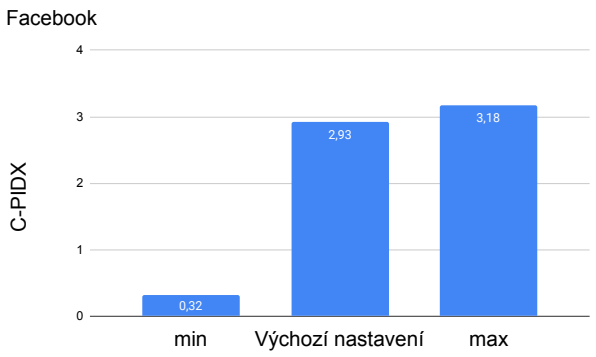
Selenium, což je nástroj dovolující automatizovaně procházet webu skrze webový prohlížeč. Tato technologie byla použita z důvodu potřeby překonat aplikační firewally sociálních sítí.

Dosavadní testování probíhalo pouze na testovacích účtech. Neboť pro ověření funkcionality nebylo zapotřebí pracovat s reálnými daty. Tato část testování se zaměřovala především na komponentu extraktoru, která zajišťuje vstupní data pro práci nástroje, tj. stažení a parsování dat. Testování proběhlo na dvou testovacích účtech pro každou podporovanou sociální síť. Na jednom účtu byla ponechána výchozí konfigurace. Příkladem tohoto vstupu je tabulka 3. Na účtu druhém byly náhodné položky nastavení pozměněny. Nakonec byla data získaná nástrojem manuálně porovnávána s daty na sociální síti. Jelikož bylo nezbytné při použití frameworku Selenium pracovat s prodlevou mezi stažením a kompletním načtením stránky včetně provedení všech scriptů. Bylo testování prováděno opakovaně. Při několika bězích testů bylo zjištěno, že tato prodleva může způsobovat potíže při hledání HTML elementů stránky. Pro zamezení tomuto problému byl přidán mechanismus, kdy při detekci tohoto chování je přidána prodleva dvě sekundy a poslední chybná akce se opakuje s takto nastavenou prodlevou. V případě opětovného neúspěchu se tento mechanismus opakuje znovu. Hodnota dvě sekundy byla zvolena na základě pozorování během testování.

Testování proběhlo pro sociální sítě Facebook, Twitter, LinkedIn a platformu Google. Během testování byl odhalen pouze problém popsaný výše i s jeho případným řešením.

Dále bylo provedeno za pomoci dříve vybraného modelu experimentální měření privátnosti pro výchozí nastavení sítě Facebook. Také byly měřeny extrémní situace nastavení, tj. situace kdy uživatel sdílí veškeré informace v co nejširším okruhu ostatních účastníků a kdy uživatel omezí sdílení informací na co nejmenší množství podle možností sociální sítě.

Výsledky experimentálního měření v síti Facebook jsou zobrazeny na obrázku 4. Lze zde vidět velký skok mezi minimálním sdílením informací a výchozím nastavením a dále již menší nárůst mezi výchozím nastavením a maximálním sdílením. Což odráží realitu, neboť ve výchozí konfiguraci Facebook zakazuje pouze sdílení polohy a několik méně podstatných položek, které nemají přílišný dopad na soukromí, ale položky s velkým dopadem jako *Kdo vás může vyhledat pomocí telefonního čísla, které jste zadali?* jsou povoleny.




**Obrázek 4.** Naměřené hodnoty na síti Facebook při výchozí konfiguraci a při mezních konfiguracích

## 9. Shrnutí a pokračování práce

Práce se zabývá soukromím na internetu resp. na sociálních sítích, což je velmi diskutované téma v dnešní době. V rámci práce byl vytvořen nástroj/aplikace, která pomůže běžnému uživateli zorientovat se v poměrně novém prostředí. Taktéž může být tento nástroj považovaný za varovný prostředek pro některé uživatele, kteří si ne zcela uvědomují skrytá nebezpečí internetu. Aplikace umí sbírat informace přímo z nastavení uživatele, což je hlavní předností této práce. Díky této funkcionalitě se pracuje s nejpřesnějšími daty a je poskytováno přesné zhodnocení soukromí a tím pádem i přesná doporučení, jak zlepšit nastavení zabezpečení privátních informací účtu.

V rámci práce byly analyzovány jednotlivé sociální sítě a jejich nastavení. Hlavním cílem práce je tato nastavení vyčíslit, tak aby bylo možné použít nějakou míru a orientační stupnici, dle které si bude uživatel moci upravit svá nastavení. Popřípadě uživateli doporučit, které položky nastavení upravit.

Aktuálním výsledkem je konzolová aplikace použitelná pro měření privátnosti v sociální síti Facebook s možností poskytnout nápovědu, jakým způsobem upravit nastavení. Pro jiné sociální sítě zatím není podporována nápověda.

Další pokračování práce bude spočívat v rozšíření kompletní funkcionality pro další sociální sítě (Twitter, Google, LinkedIn) a poté se bude zabývat především testováním, zejména uživatelským testováním. Předpokládá se, že do testování bude zahrnuto dvacet uživatelů se svým nastavením soukromí na různých sociálních sítích. Výsledkem by mělo být pravděpodobnostní rozložení skóre soukromí. Dále budou určeny extrémy skóre privátnosti pro další sociální sítě. Na základě těchto testů by se měli dle skóre soukromí rozdělit uživatelé do několika skupin, do kterých bude následně prováděna klasifikace.

## 10. Poděkování

Tímto bych chtěl poděkovat vedoucímu své práce Mgr. Kamilu Malinkovi Ph.D. za odborné vedení a cenné rady.

## Literatura

[1] Elena Zheleva, Evimaria Terzi, and Lise Getoor. Privacy in social networks. In *Synthesis Lectures on Data Mining and Knowledge Discovery*, volume 3, pages 1–85, 2012-03-31.

[2] Fredrik Erlandsson, Martin Boldt, and Henric Johnson. Privacy threats related to user profiling in online social networks. In *2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Confernece on Social Computing*, pages 838–842. IEEE, 2012.

[3] Kun Liu and Evimaria Terzi. A framework for computing the privacy scores of users in online social networks. In *ACM Transactions on Knowledge Discovery from Data*, volume 5, pages 1–30, 2010-12-01.

[4] Yong Wang, Raj Kumar Nepali, and Jason Nikolai. Social network privacy measurement and simulation. In *2014 International Conference on Computing, Networking and Communications (ICNC)*, pages 802–806. IEEE, 2014.

[5] Nilothpal Talukder, Mourad Ouzzani, Ahmed K. Elmagarmid, Hazem Elmeleegy, and Mohamed Yakout. Privometer. In *2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010)*, pages 266–269. IEEE, 2010.

[6] Justin Lee Becker and Hao Chen. Measuring privacy risk in online social networks. 2009. Dostupné z https://web.cs.ucdavis.edu/~hchen/paper/w2sp2009.pdf.

[7] Michelle Madejski, Maritza Johnson, and Steven Bellovin. The failure of online social network privacy settings. 01 2011. Dostupné z https://www.researchgate.net/publication/229046080_The_Failure_of_Online_Social_Network_Privacy_Settings.

[8] Joon S. Park, Kevin A. Kwiat, Charles A. Kamhoua, Jonathan White, and Sookyung Kim. Trusted online social network (osn) services with optimal data management. volume 42, pages 116–136, 2014.

[9] Česká terminologická databáze knihovnictví a informační vědy (tdkiv), 2003.

[10] Yong Wang and Raj Kumar Nepali. Privacy measurement for social network actor model. In *2013 International Conference on Social Computing*, pages 659–664. IEEE, 2013.

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

# Static Deadlock Detection in Frama-C

Tomáš Dacík*

**Abstract**
Frama-C is a platform for static analysis of source codes written in the C language. It provides a wide range of analysers usually based on EVA – Frama-C's value analysis plugin. Despite some attempts to support analysis of multi-threaded code have been done in Frama-C, the whole platform is currently limited to analysis of sequential code only. In this paper, we present *Deadlock*, a new plugin of Frama-C focused on deadlock detection. Together with the core algorithm of deadlock detection, we present a technique our analyser uses to handle multi-threaded code partially as a sequential one, which allows us to improve the precision of our analysis by using existing plugins of Frama-C. In our experimental evaluation, we show that our tool is able to handle real-world C code with a high precision.
**Keywords:** Static Analysis — Deadlock Detection — Frama-C

**Supplementary Material:** Deadlock repository

*xdacik00@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

## 1. Introduction

In a majority of concurrent programs, some kind of synchronisation is necessary to guarantee consistency of data. However, incorrectly used synchronisation mechanisms may lead to another class of concurrency issues, like, for example, *deadlocks*. In this work, we focus on deadlocks caused by incorrect usage of *locks*, low-level synchronisation primitives, often used in the C language. In this particular case, a deadlock is defined as a situation where each thread from some set is holding a lock and waiting for another lock, that is held by (possibly the same) thread from the set. An example of a simple deadlock is given in Listing 1.

Our implementation of deadlock detection is inspired by the tool RacerX [1] and the deadlock analysis implemented in the CPROVER framework [2]. While RacerX is explicitly designed to handle large code bases and therefore does not do any pointer analysis and resigns on soundness, CPROVER tries to be sound, which leads to its slow running times as well as a lot of false positives. Our goal is to design an analyser that combines both presented solutions, i.e., can use the existing pointer analysis available in Frama-C, but with the stress put on detection of likely deadlocks rather than soundness.

**Listing 1.** A simple C program with a deadlock between threads *thread1* and *thread2*

```
void f() {
    pthread_mutex_lock(&mutex1);
    if (...)
        pthread_mutex_unlock(&mutex1);
}

void g() {
    pthread_mutex_lock(&mutex1);
    pthread_mutex_lock(&mutex2);
}

void *thread1 (void *v) {
    pthread_mutex_lock(&mutex2);
    f();
}

void *thread2 (void *v) {
    g();
}
```

Another deadlock detection approach for low-level C programs is implemented in the L2D2 (Low-Level Deadlock Detector) plugin [3] of Facebook Infer. It uses a completely different method – an incremental and compositional analysis based on analysing each function without its calling context. While this approach promises to be more scalable, it can also in principle produce more false alarms.

We also note that there exist many more static deadlock analysers, but many of them target higher-level languages (such as Java or C++) or are much more heavyweight. A discussion of such tools is beyond the scope of this paper.

## 2. Frama-C

Frama-C is an open-source platform for static analysis of source codes written in the C programming language. Frama-C has a modular, plugin-based architecture. Out of the existing plugins, the most heavily used is the EVA (Evolved Value Analysis) plugin [4], which computes an over-approximation of sets of possible values of variables at each program point. Its results can be used for proving the absence of generic errors or assertions written in a specialised assertion language. They also serve as the input for other plugins implementing, for example, program slicing, various code optimisations, or test-input generation. There are also plugins for deductive verification, which are, however not relevant in the context of this paper.

Only sequential code can be analysed by the current version of EVA and consequently by all plugins based on it. As we have already mentioned in the introduction, some attempts to implement analysis of concurrent code have been done in Frama-C, but they were rather experimental and are no longer under active development. Examples of such attempts include the Mthread plugin [5] focused on data race detection, whose source code is unfortunately proprietary, and Conc2Seq [6] for translating concurrent code and its specification into sequential code simulating the original code and checking the given specification. This process is limited only for a subset of the C language.

## 3. Thread Analysis

Our analysis runs in two phases. In the first phase, we compute possible initial states of different threads (including information about which thread can be started at all). In the second phase, we perform a lockset analysis in which we analyse each thread as a sequential program assuming that it is started from the computed initial state. Here, note that we use the term *thread* as an abstraction representing all threads (instances that could be created during execution of a program) with the same entry point (and hence the same control).

In this section, we concentrate on the first phase of the analysis, namely, the computation of which threads can be created and with which initial states in terms of possible values of global variables and values of arguments passed to threads. The main idea is to use a fixpoint algorithm that runs as long as new threads are discovered. Each iteration of this fixpoint computation employing a nested fixpoint computation that iterates over so-far known threads, analyses them through EVA, and propagate information between them through thread creation statements only. This way, the possibility of creating new threads may be discovered. These threads will then be analysed in another iteration of the outer loop. Note that this approach under-approximates the real behaviour of the threads since no thread interleaving is considered. This is a design decision which we have done for the sake of efficiency of our analysis. While the analysis can indeed under-approximate the real behaviour, in the second phase, we are mainly interested in the parameters of lock/unlock functions, i.e., identifiers of locks, which are usually not that much influenced by thread interleaving in practice.

**Algorithm 1:** Computation of initial states of threads

**Input :** *create_stmts* ... statements where threads can be created

1 **function** *build_graph(threads)*
2 $G = \text{empty_graph}()$
3 **foreach** $t \in threads, s \in create_stmts$ **do**
4 set_active_thread($t$)
5 **if** $is_reachable_by_thread(s,t)$ **then**
6 $children = \text{get_threads}(s)$
7 **foreach** $child \in children$ **do**
8 $G.\text{add_edge}(t,s,child)$
9 **end**
10 **end**
11 **end**
12 **return** $G$
13
14 **function** *analyse_threads()*
15 $i = 0$
16 $G^0 = \text{build_graph}(\{main\})$
17 **do**
18 $i = i + 1$
19 $\widehat{G}^i = \text{compute_fixpoint}(G^{i-1})$
20 $threads = \widehat{G}^i.\text{get_nodes}()$
21 $G^i = \text{build_graph}(threads)$
22 **while** $G^i \neq \widehat{G}^i$
23 **return** $G^i$

Our method of computing initial states is formalised in Algorithm 1. The function `build_graph` is used to construct a graph encoding which thread can create which other threads through which thread-create statements based on the current approximation of the

possible initial states of the threads. The function `set_active_thread` (line 4) is implemented as a wrapper over EVA and used to set its context according the so-far computed initial states of the given thread. For each create statement that is found reachable by EVA from the initial state of the thread being examined, we use EVA to find threads it can create and add corresponding edges to the graph.

The function `analyse_threads` first builds a graph based only on the initial state of the main thread, containing every thread that can be created from the main. Once new initial states are computed, new graphs are iteratively computed on line 21. To update initial states of the threads, we propagate states of their parents in the create statements (line 19). To handle programs with nested or even cyclic dependencies between threads, we compute a fixpoint of a function propagating states over the graph. The fixpoint computation over the graph is implemented using the OCamlgraph library[1]. For programs where threads are created in the main thread only, one iteration of the loop between lines 17 and 22 suffices. However, for more complex programs where the computation of the initial states leads to discovering new threads or dependencies, more iterations of the loop are necessary – we loop until the computed graphs stop changing.

We illustrate the algorithm on the program from Listing 2. It starts with the set $create_stmt$ containing $stmt6$ and $stmt16$. First, we compute $G^0$ using the `build_graph` function. Since only $stmt16$ is reachable from the main thread, a graph with the single edge $main \xrightarrow{stmt16} thread1$ will be returned. The fixpoint computation over this graph is trivial – the state of $main$ at $stmt16$ is propagated as the initial state of $thread1$. Afterwards, we check whether a new thread can be discovered based on new initial states. We find that $thread2$ can be created from $thread1$ and add the corresponding edge $thread1 \xrightarrow{stmt6} thread2$ to the graph. The initial state of $thread2$ is computed analogously. Since there is no other thread, we return the initial states computed as follows (thread arguments are ignored):

$$thread1 : \{i \mapsto \{0\}\},\ thread2 : \{i \mapsto \{1\}\}$$

Note that the incrementation on line 17 is not reflected in the initial states of $thread1$ and $thread2$ because it is done after the thread creation.

[1] http://ocamlgraph.lri.fr/index.en.html

**Listing 2.** A program with nested threads (interfaces of thread-create functions are simplified)

```
1  int i = 0;
2
3  void *thread1 (void *v) {
4      i++;
5      pthread_t t;
6      pthread_create(&t, thread2);
7      return NULL;
8  }
9
10 void *thread2 (void *v) {
11     return NULL;
12 }
13
14 int main() {
15     pthread_t t;
16     pthread_create(&t, thread1);
17     i++;
18     return 0;
19 }
```

## 4. Lockset Analysis

The key part of our analyser is a lockset analysis inspired by the tool RacerX [1]. In RacerX, however, no pointer analysis is used, and so we had to extend its methods for this purpose. The term *lockset* refers to a set of locks that a thread holds at a particular program point. The result of the lockset analysis is the set of possible locksets for each program statement. Based on this information, we can construct a lock-order graph (further referred to simply as a lockgraph). An edge $a \to b$ in the lockgraph indicates that some thread tries to acquire lock $b$ while already holding lock $a$.

Our lockset analysis is performed for each thread detected in the previous phase by a depth-first traversal of its control flow graph. The traversal is implemented as path-insensitive, i.e., all conditions are resolved non-deterministically. The analysis is started with the empty lockset, which is modified by the locking and unlocking operations according to the transfer function defined as follows ($[\![p]\!]$ denotes the set of possible values of the variable $p$):

$$t_{stmt}(ls) = \begin{cases} \{ls \cup \{l\} \mid l \in [\![p]\!]\} & \text{if } stmt \text{ is lock}(p) \\ \{ls \setminus \{l\} \mid l \in [\![p]\!]\} & \text{if } stmt \text{ is unlock}(p) \\ \{ls\} & \text{otherwise} \end{cases}$$

In the transfer function, the evaluation of the parameters (identifiers of locks) ignores the calling context in order to facilitate usage of summaries computed as described in Section 4.1. After applying the transfer function, the analysis is forked for each pair consisting of a successor statement and possible lockset.

Let us consider the following pseudocode as an example:

```
f() {            // entry lockset = {}
    lock(p);     // [[p]] = {m1,m2}
    unlock(p);   // [[p]] = {m1,m2}
}
```

Applying the transfer function on the first statement results into the set of locksets $\{\{m1\},\{m2\}\}$. The rest of the function is then analysed separately for $\{m1\}$ and $\{m2\}$. In both cases, after applying the transfer function on the second statement, we assume that both $m1$ as well as $m2$ can be unlocked despite the fact that one of them was not locked. In other words, we over-approximate the real behaviour considering all combinations of locking and unlocking in such a case. The exit set of locksets of $f$ will then be $\{\{m1\},\{m2\}\}$.

### 4.1 Function Summaries

Function summaries are an efficient way to speed up interprocedural analysis. In our analysis, function summaries are represented by a mapping from pairs (*function*, *entry lockset*) to a set of exit locksets. The interpretation is the following: if the *function* is called with the *entry lockset*, the result is the union of the sets of locksets at each of its exit points. For example, the analysis of the function $f$ from Listing 1 called from line 14 with the entry lockset containing `mutex2` will produce the following summary:

$$\{(f,\{mutex2\}) \mapsto \{\{mutex1,mutex2\},\{mutex2\}\}\}$$

Since the evaluation of the locks used is done regardless of the calling context, $f$ will produce the same result at every other call site with the given entry lockset. In theory, each function could be analysed with each possible lockset, which means up to $2^n$ times where $n$ is the number of locks used in the program. However, functions in real programs usually release all locks they acquired, and if the pointer analysis is not too imprecise, only small locksets are created. This implies that the majority of functions are not analysed many times. Our experiments described in Section 5 show that for a subset of programs where we detected some locking operations, functions are on average analysed 1.89 times only.

### 4.2 Lockgraph Construction

When updating the locksets, whenever a lock $l$ is added to a nonempty lockset $ls$, a set of edges is added to the lockgraph. The set is computed as $ls \times \{l\}$. To track information of the origin of the edge, each edge is labelled by a set of traces. These traces are created by concatenating call stacks that lead to locking both of the locks as described in Section 4.5. The final step is to check whether there are cycles in the resulting graph, denoting possible deadlocks.

In this step, so-called *self-deadlocks*, i.e., deadlocks caused by a single thread on a single lock are ignored by default because they could lead to many false positives.

### 4.3 Context Sensitivity

The context insensitive evaluation of locking parameters may cause significant imprecision when wrappers of locking functions are used. An example of such a situation is given in Listing 3. Without context-sensitivity, the evaluation of the variable `m` on line 2 will always be the set of all mutexes used in the program. As a result, the analyser will assume that function `lock_wrapper` can lock any mutex. Then, on line 7, besides the real dependency *mutex1* $\rightarrow$ *mutex2*, the dependency *mutex2* $\rightarrow$ *mutex1* will also be created. Generally, such a situation results in a graph containing all possible edges.

For that reason, we allow such wrapper functions to be analysed in a different way. Namely, during the analysis of such functions, the call stack is taken into account when evaluating variables. A disadvantage is that we can no longer use summaries as described in Section 4.1 for such functions. A list of wrapper functions can be provided by the user of the analysis, but we also try to detect them automatically. To identify them, we check parameters of all functions, and if any of them is either a type representing a lock or a structure containing (possibly recursively) a lock, we mark the function as *context-sensitive*.

**Listing 3.** An example of a lock wrapper

```
1 void lock_wrapper(pthread_mutex_t *m) {
2       pthread_mutex_lock(m);
3 }
4
5 void *thread (void *v) {
6       lock_wrapper(&mutex1);
7       lock_wrapper(&mutex2);
8 }
```

### 4.4 Concurrency Checking

To reduce false positives, we check if all edges involved in a detected cycle are concurrent. That is not the case when, for example, both edges of the cycle were created in the main thread or generally in a thread that is not created multiple times. Two threads can also be non-concurrent if the first one is always joined before the second one is created. Checking the first

condition is simple; to decide the second one, we use a graph traversing algorithm that checks whether the first thread is always joined before the second one is created (this excludes deadlocks between threads that can never run simultaneously). Another situation that we currently do not take into account, results from using of the so-called *gatelocks*. This situation happens when both dependencies are created in a critical section protected by a common lock and therefore they cannot be reached simultaneously during the execution of the program.

Even though a cyclic dependency does not lead to a deadlock, such a situation still can be considered as a violation of a lock discipline, which may introduce a deadlock in the future, and is therefore reported as a warning of a lower severity.

### 4.5 Deadlock Reporting

To be useful in practice, the analyser should be able to provide the user with information that helps him/her to understand the reported issue. Since we do forward traversal and analyse each program path separately, we can easily report a trace of each dependency involved in a cycle and hence a potential deadlock. A trace of the dependency is created by concatenating call stacks of points where the involved locks were acquired. To make the report more succinct, the common prefix is reported only once. A deadlock report for the program in Listing 1 could then look as follows:

```
==== Lockgraph: ====
mutex1 -> mutex2
mutex2 -> mutex1
==== Results: ====
Deadlock between threads thread1 and thread2:

Trace of dependency (mutex2 -> mutex1):
In thread thread1:
    Lock of mutex2 (simple_deadlock.c:13)
    Call of f (simple_deadlock.c:14)
    Lock of mutex1 (simple_deadlock.c:2)

Trace of dependency (mutex1 -> mutex2):
In thread thread2:
    Call of g (simple_deadlock.c:18)
      Lock of mutex1 (simple_deadlock.c:8)
      Lock of mutex2 (simple_deadlock.c:9)
```

### 4.6 A Heuristic Avoiding EVA

When analysing complex programs using Frama-C and EVA, one usually needs to tune their input parameters to achieve both precision and a reasonable running time. After reporting some classes of alarms, EVA will consider the rest of the code unreachable, and the user must first solve the issue (either by fixing the code, changing parameters of Frama-C/EVA, or providing models for external functions). To provide a fully-automated alternative, we implemented a so-far experimental method that completely avoids using EVA and uses purely syntactic information to identify locks and threads.

The workflow of the analyser remains the same, only the implementation of queries to EVA in the wrapper over it differs. Instead of the value analysis, we use functions of the Frama-C API to extract information which variables are contained in expressions. This is sufficient when only references to global variables are used. If this is not the case, which happens, e.g., for locks that are members of structures frequently passed among functions, the method can lead to both under- and over-approximation. In the case of threads, we also need to find their entry point functions. If this is not possible, we assume that every function with a POSIX threads signature (*void * f (void *)*) can be an entry point of a given thread. For other queries to EVA (mainly related to computation of initial states of threads), top values of the abstract domains corresponding to any possible value are returned.

## 5. Experiments

We have evaluated our analyser on the benchmark originally used in [2][2]. The benchmark contains 993 programs that are considered to be deadlock-free and 8 programs with deadlocks, which were introduced by the authors of the benchmark. All programs are from the Debian GNU/Linux distribution and use the POSIX thread API. Out of the benchmarks, we could unfortunately use a subset only. A huge fraction of the benchmarks was rejected by Frama-C due to type errors (probably caused by a preprocessing done for the CPROVER tool). Moreover, some of the test cases contain locking operations in non-reachable code only. We compare results our tool achieved on the rest of them with those obtained by CPROVER and L2D2.

The experiments with our tool were conducted on a machine with 2.5GHz Intel Core i5-7300HQ processor and 16 GB RAM, running Ubuntu 18.04. To overcome problems with parametrisation described in Section 4.6, we tried to analyse each program using several combinations of parameters to suppress some errors reported by EVA that stop the analysis. However, this leads in some cases to a slow running time and to timeouts before our deadlock analysis even started.

**Programs with deadlocks**. When using value analysis, our tool detected deadlocks in all 8 cases that actually contain a deadlock. Both L2D2 and

[2] http://www.cprover.org/deadlock-detection

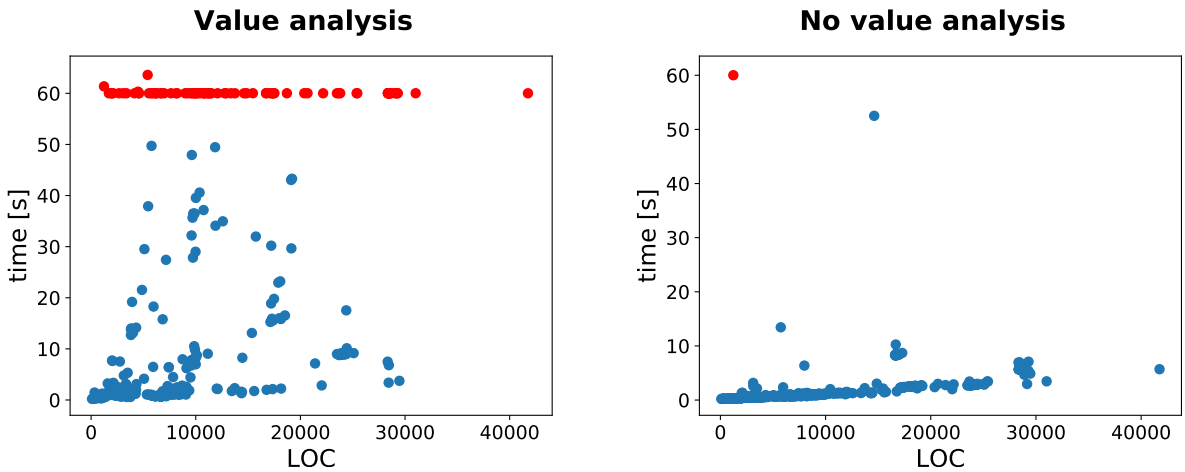



**Figure 1.** The time needed for the analysis (timeouts after 60 seconds are marked by the red colour)

**Table 1.** Experimental results on 278 deadlock-free test cases that Deadlock can handle with value analysis

| | correct | false positives | no result |
|---|---|---|---|
| Deadlock | 181 | 9 | 88 |
| L2D2 | 259 | 11 | 8 |
| CPROVER | 82 | 40 | 156 |

**Table 2.** Experimental results on 393 deadlock-free test cases that Deadlock can handle without value analysis

| | correct | false positives | no result |
|---|---|---|---|
| Deadlock | 357 | 35 | 1 |
| L2D2 | 359 | 25 | 9 |
| CPROVER | 114 | 45 | 234 |

CPROVER manage to detect them too. Our lightweight version missed one deadlock in a program that uses lock wrappers. This is due to this approach, unlike the solution described in Section 4.3, will see a single lock represented by the formal parameter of the lock wrapper function only, and hence it will not create any locking edge.

**Deadlock-free programs**. Tables 1 and 2 presents results that our tool with and without using EVA, respectively, achieved on deadlock-free programs that Frama-C could handle and their comparison with results of CPROVER and L2D2. The different numbers of test cases considered in the two tables are caused by the fact that an incorrect parametrisation can lead to considering some locking or thread-creating operations to be unreachable as described at the beginning of this section. The column *no result* includes cases where (a) our tool hit a timeout, (b) CPROVER timeouted or ran out of memory, and (c) L2D2 hit a compilation error.

Figure 1 shows how the running time of our tool grows with the number of lines of code of the programs being analysed when used with and without EVA, respectively. The left part of the graph devoted to the analysis with EVA shows the importance of choosing the right values of parameters of Frama-C and EVA: programs are either analysed quickly (often close to cases when no value analysis is done) or the analysis times out. Note that during the evaluation of CPROVER much higher limits were used: a timeout of 30 minutes and 24 GB of memory.

Further, to verify basic correctness of methods presented throughout the paper, we also prepared a set of crafted programs. These programs are available in the project repository[3].

## 6. Conclusions

We presented a design of *Deadlock*, a new Frama-C plugin for deadlock detection. The experiments show that it is able to handle real-world C code. However, in some cases, we are limited by the pointer analysis of Frama-C that we use for lock representation. Further work could concentrate on improving precision of method used when value analysis is too demanding.

We are currently also working on a data race detector using a part of our deadlock analysis. Our lockset analysis can be used for checking whether memory accesses to shared variables are protected by locks. In order to do this, some parametrisation needs to be done, because in contrast with deadlocks we need to be more conservative when adding locks to a lockset – an invalid lock generated at the beginning of the analysis would "hide" all following possible races. Differentiating between may- and must- locksets is a possible way to achieve this.

[3] https://github.com/TDacik/Deadlock/tree/master/tests

## Acknowledgements

I would like to thank my supervisor Prof. Tomáš Vojnar for his help. The work was supported by the H2020 ECSEL project Arrowhead Tools.

## References

[1] Dawson Engler and Ken Ashcraft. RacerX: Effective, static detection of race conditions and deadlocks. In *Proc. of SOSP'03*.

[2] Daniel Kroening, Daniel Poetzl, Peter Schrammel, and Björn Wachter. Sound static deadlock analysis for C/pthreads. In *Proc. of ASE'16*.

[3] Vladimír Marcin. Statická analýza v nástroji Facebook Infer zaměřená na detekci uváznutí. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2019.

[4] Sandrine Blazy, D. Bühler, and B. Yakobowski. Structuring abstract interpreters through state and value abstractions. In *18th International Conference on Verification Model Checking and Abstract Interpretation (VMCAI 2017)*.

[5] Boris Yakobowski and Richard Bonichon. Frama-C's Mthread plug-in manual. https://frama-c.com/download/frama-c-mthread-manual.pdf.

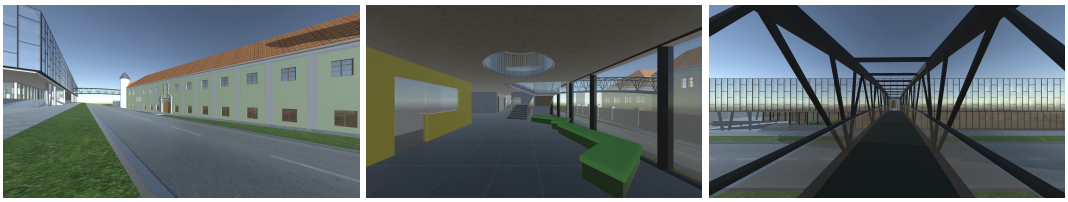
[6] A. Blanchard, N. Kosmatov, M. Lemerre, and F. Loulergue. Conc2Seq: A Frama-C plugin for verification of parallel compositions of c programs. In *2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM)*.

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

# Virtual Tour of FIT for Oculus Quest

Michal Janů*



**Abstract**
The main goal of this project is to make an application for the Oculus Quest VR headset called FIT_VR. This application has several features that allow the user to do more than just walk freely around the FIT BUT areal, such as Navigation and Instant travel. The Navigation feature is used to find the shortest route to the desired office or lecture room and instant travel allows to choose a starting location.

**Keywords:** VR tour of FIT — VR FIT — FIT tour on Oculus Quest — VR Oculus quest — Unity based VR application

**Supplementary Material:** Demonstration Video

*xjanum03@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

## 1. Introduction

The focus of this application is to help students and any other users get oriented on the school grounds and help them find a route to the room they need to go to. Another goal is just to let users explore the areal from the comfort of home and familiarize virtual reality to anyone by showing that VR [1] is not just used for gaming but that it can be used for something useful.

There already is an existing method to look around the school and that is with use of 360° photos [2] that are accessible from any device but 360° photos do not allow the user to go everywhere, unlike FIT_VR[1].

The whole VR application is based on the Unity3D engine[2] that provides the basic building elements. Oculus provides an SDK for Unity3D that enables easier work with their VR headset and controllers. All models (More in Section 6) appearing in FIT_VR[1] are done in Blender[3].

Everything in the application is made from scratch, e.g. movement, navigation (Section 3), all interactive objects (Section 4) such as menus and doors. This allows to make everything exactly as needed, for example, the default movement in VR is really uncomfortable for some people so the application allows the user to set up (Section 5) the movement as they prefer it to be. In order to make FIT_VR even more user-friendly, the application provides controller hints (Section 5) that help the user with the use of controllers.

## 2. Oculus Quest

The headset (Figure 1) is completely wireless thanks to the mobile CPU that is integrated inside the headset and to four cameras that use positional tracking that means no sensor stations so the VR can be set up practically anywhere. With the headset also come two wireless controllers [3].

## 3. Navigation

For navigation, is used Unity3D's NavMesh [4] which is used for spatial queries, like pathfinding [5] and walkability tests. The NavMesh creates Mesh that

[1] Application for a virtual tour of BUT FIT
[2] https://unity.com
[3] https://blender.org

**Figure 1.** Oculus Quest headset and controllers

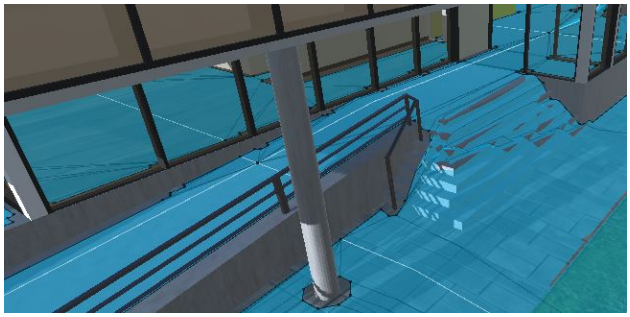


**Figure 2.** Blue texture on the ground is the generated mesh.

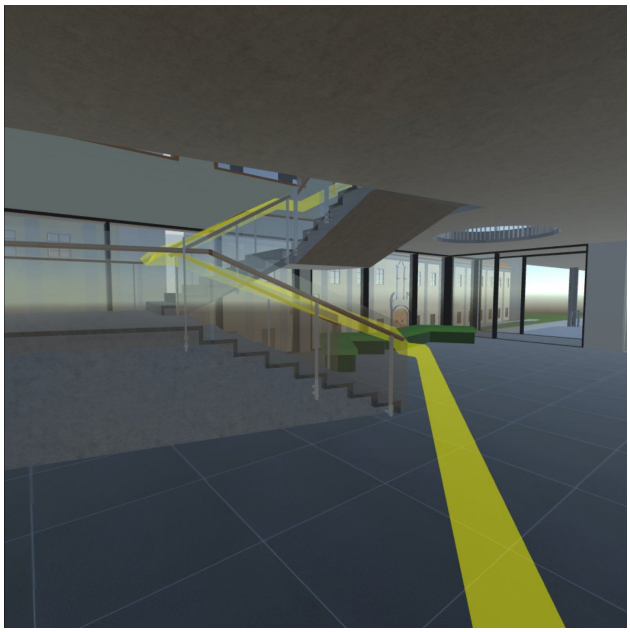


**Figure 3.** Visual representation of the navigation. The yellow path shows the shortest path calculated by NavMesh.

represents the walkable area – see Figure 2. It uses colliders to determine if the path is walkable or not. NavMesh is used by NavMeshAgent to verify the existence of a path between the Agent (user) and the destination (object).

With an object present on the mesh selected as the destination, the NavMesh finds the shortest route through the mesh which is then returned as array of coordinates. To visualise the path all coordinate points are connected by a line-renderer, which highlights the route – Figure 3. The Navigation is extended for re-routing if there is a shorter route existing.

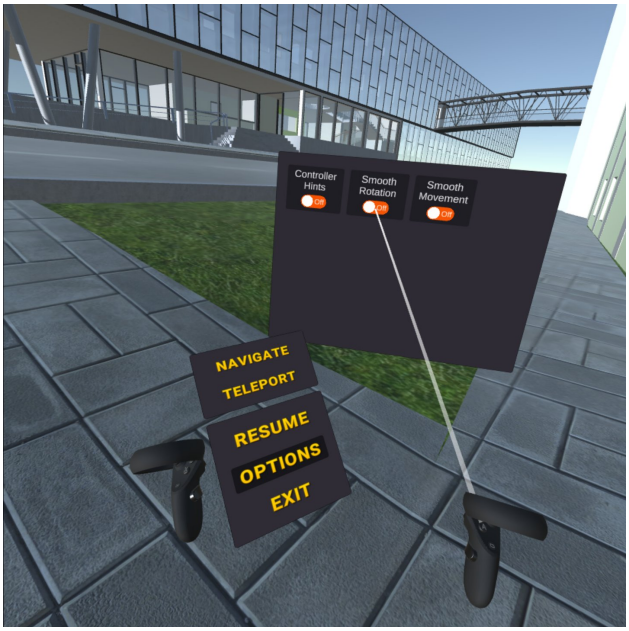



**Figure 4.** In the picture, you can see the user interface of the menu. With opened options tab where the user can set up the movement and hints. The white line is representing where the cursor is being directed.

## 4. Interactive Environment

All interactive objects in the application can be divided into two categories. One category of interactive objects are physically interactive objects such as floors, walls and other objects that have a rigid body (More in Section 4.1) or colliders (Section 4.1). The other category of user interactive objects are widgets shown only for user interaction, for example menus and door handles.

### 4.1 Physical Interactive Objects

Physical interactive objects are all objects that have a visible model (except menus etc.) in terms of FIT_VR. For example, walls and floors have colliders that restrict the user from falling through the floor or walk through walls and doors. The user has control over the object with a camera that can move on any surface and has a rigid-body attribute that is restricted by colliders.

- **Rigid-body** It's component used on the user object, that applies Unity3D's physics engine.
- **Colliders** Defines the physical collision of selected object.

### 4.2 User-Centered Interactive Objects

User-centered interactive objects are objects such as menus and door handles, where the user uses controllers to interact with them. For menus, a prefab provided by Oculus is used, whitch uses ray-casting from the controller in the direction of its normal and

**Figure 5.** Upper image shows visualisation of ray-cast hitting door handle and the image below when the ray-cast is not hitting the handle.

then checking if the ray hits any UI element in its path and visualizes the ray with line-renderer (More in Section 3). For door handles, is used a similar script that uses ray-casting from controllers that evaluates if the door handle is hit and then a button is pressed for the door to be open.

- **Prefab** Prefab is an object created as a reusable asset. Such object (prefab) stores all scripts and components applied to it.

## 5. Options

Using VR can cause motion sickness to some people. That's why the application allows configuring movement so that the user feels most comfortable. Users can configure whether walking and turning are smooth or not. There is an option to turn on/off controller hints as well.

## 6. Models

All models are made in Blender[3] and referenced by building plans provided by Lukáš Duránik and references for details of models thanks to 360° photos made by Anna Popková. User has access to all rooms

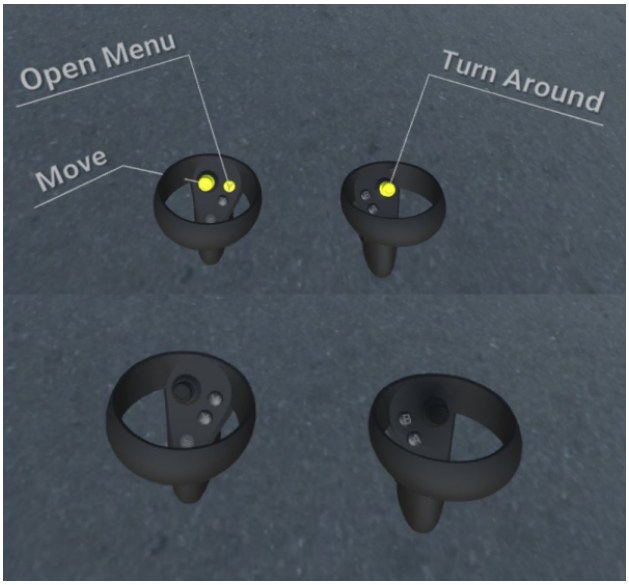



**Figure 6.** Visualisation of when the controller hints are enabled and disabled.

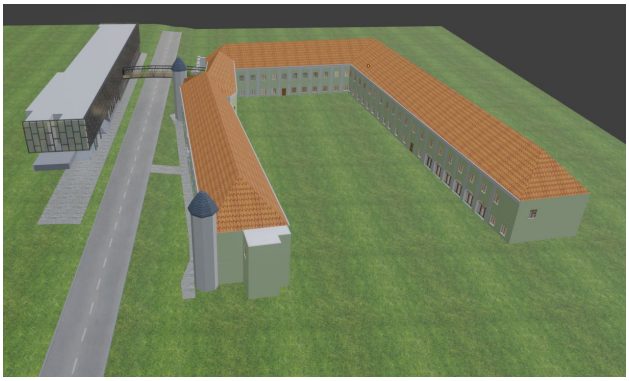


**Figure 7.** All currently user accessible buildings such as buildings A,B,C,L and bridge connecting building A and L.

and corridors that normal student has usually access to. Models for controllers are provided by Oculus but they had to be modified in the way where all buttons and joysticks could be highlighted for controller hints.

## 7. Conclusions

Through this application, anyone can explore FIT BUT without having to leave the room. If one is trying to find the right office or lecture room, he or she can use this application to find out the right way.

This application can become much more than it already is. In the future, the user could walk towards any room, for example, the library and opens the UI window that contains the information about the room, such as the current schedule. The model itself could be combined with 360° photos to show a more realistic view. Thanks to the Oculus Quest's mobile CPU that runs android it could be modified to run on any device, not just the VR ones.

## Acknowledgements

I would like to thank my supervisor prof. Adam Herout for his help, and Lukáš Duránik for providing building plans used as a reference for modeling.

## References

[1] Grigore C Burdea and Philippe Coiffet. *Virtual reality technology*. John Wiley & Sons, 2003.

[2] Yan-Fai Chan, Man-Hong Fok, Chi-Wing Fu, Pheng-Ann Heng, and Tien-Tsin Wong. A panoramic-based walkthrough system using real photos. In *Proceedings. Seventh Pacific Conference on Computer Graphics and Applications (Cat. No. PR00293)*, pages 231–240. IEEE, 1999.

[3] VR Oculus. Oculus.[online] available at: https://www.oculus.com/blog/introducing-oculus-quest-our-first-6dof-all-in-one-vr-system-launching-spring-2019, 2018.

[4] Unity Technologies. Navmesh. https://docs.unity3d.com/ScriptReference/AI.NavMesh.html.

[5] Zhang He, Minyong Shi, and Chunfang Li. Research and application of path-finding algorithm based on unity 3d. In *2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)*, pages 1–4. IEEE, 2016.

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

# Speech Enhancement with Cycle-Consistent Neural Networks

Bc. Pavol Karlík*

**Abstract**
Speech enhancement aims to improve speech intelligibility and overall perceptual quality of speech by using various algorithms. Neural networks (NNs) have become a standard approach for solving such problems. NNs are usually trained by comparing the network output to the target sample. In our work, we incorporate cycle consistency constraint during the training period to improve the network robustness — we add another NN to the process. The second NN performs an opposite task — its goal is to introduce noise to clean speech recording. The networks are trained in a cycle, each taking the output of the other network as an input. Cycle-consistency, among other things, causes the network to see a much larger variety of noisy data, which improves the network's robustness. We perform experiments on both paired and unpaired data, which is enabled by adding adversarial training to the training. The DNN models are evaluated by using an automatic speech recognition system. The speech enhancement models trained and evaluated in this work are based on a recent publication. Our results have shown that adding cycle-consistency improves the models' performance significantly.

**Keywords:** Speech Enhancement — Deep Learning — Cycle-Consistency

**Supplementary Material:** *N/A*

*xkarli05@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

## 1. Introduction

Automatic speech recognition (ASR) is a widely used technology that allows transcription of a spoken speech utterance into a corresponding sequence of words. The field has been intensively researched in the past decades, with significant advances being made through the years. These improvements led to a surge in usage of intelligent human-machine speech communication systems, such as virtual speech assistants or interactive voice response systems.

Despite significant advances in this area, there are still certain factors that limit the performance of such systems. Most notably, reverberation and ambient noise drastically reduce speech quality of speech signal. There are many speech enhancement (SE) and ASR techniques to detect and combat the effects of noise and reverberation [1, 2, 3].

Current state-of-the-art speech enhancement methods primarily employ artificial neural networks (ANNs) [4]. These networks are trained using datasets that contain pairs of clean spoken utterances and the same spoken utterances with incorporated noise and reverberation — *paired* data. However, paired sets for training the network are not always available. *Generative adversarial network* (GAN) [5] is a framework that enables neural networks to train on *unpaired* data. In GANs, two neural networks are pitted against each other, each attempting to reach its objective, which is 'adversary' to the other network. Generative adversarial network essentially models the distribution of a given dataset.

One of the modifications of the aforementioned framework, *CycleGAN* [6], uses *cycle-consistency* for unpaired data training to further improve the architecture. CycleGAN as a whole is described in Section 3. In [6], it was demonstrated that enforcing cycle-consistency constraint significantly improves the

model robustness in image-to-image translation[1]. Cycle-consistency can be achieved by introducing an additional neural network to the framework. The network performs a dual task — in our case, it attempts to produce noisy speech signal given clean speech signal as input. Both networks are then used in conjunction to be consistent with each other. Therefore, we can run a corrupted sample through denoising network, and then use its output — an enhanced speech signal — as input to the other network, producing *reconstructed* corrupted sample. The constraint is therefore enforced by adding the reconstruction loss function to the main objective function.

The potential of CycleGAN has mostly been explored in the image processing field. The aim of this work is to evaluate the effect of cycle-consistency in the speech enhancement domain, for both paired and unpaired data. This work is based on a research paper recently published by Meng et al. (2018) [7] which proposes a framework inspired by [6] for a speech enhancement task. We implement and evaluate neural network models using the constraint during the training period for various architectures. First, using paired data, we train a standard NN without the constraint, which will serve as a baseline. Then, we train a network which inserts noise into a clean speech signal. We use that network and the baseline to further train the model with cycle-consistency. Lastly, we use the same dataset as if it contained no pairs to train a GAN with cycle-consistency constraint.

We perform experiments using automatic speech recognition (ASR) system on the CHiME-3 dataset [8]. We use evaluation set for evaluating the models with ASR system and training set for training the models. In addition, we re-train the acoustic model (AM) with data enhanced with our models and perform another set of experiments using the ASR system with re-trained AM.

Section 2 explains the problem of noise and reverberation in the speech recognition field. In Section 3, we briefly discuss the process of training standard neural networks and GANs. Additionally, we describe cycle-consistency constraint. In Section 4, we describe experiments performed in this work. Section 5, showcases our results. The article concludes with Section 6, in which we briefly summarize the results we have achieved and present possible improvements.

[1]In the case of mapping an input image to a specific output image, paired data is rarely available. For example, transforming a photo to a painting in the style of Van Gogh. There are no existing photo-Van Gogh painting pairs. However, an unpaired collection of the artist's paintings and photos can be used to train such model.

## 2. Speech Enhancement and Noise Reduction

Despite widespread use of ASR technologies in various systems, there is a large number of challenges that such systems need to handle to be applicable. When a speech signal is captured by a microphone, the picked up signal can get corrupted, causing loss of quality and intelligibility. Such an altered speech signal might then be erroneously processed by the ASR system. One of the fields that pursues this problem is *speech enhancement*.

This Section describes the impact of noise on speech recognition and briefly overviews current speech enhancement approaches.

### 2.1 Noise and Reverberation

Noise in a speech signal generally represents an unwanted modification that a signal may be subjected to when being captured or processed. According to the spectral distribution, the noises can be grouped into two categories - *stationary noise*, which keeps constant spectral distribution over time and *non-stationary noise*, which is more difficult to suppress because its statistics change over time.

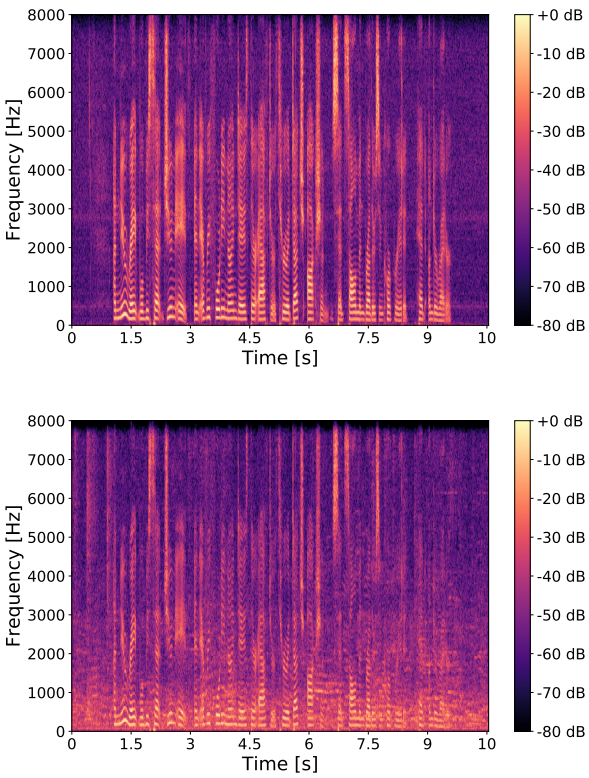



**Figure 1.** Comparison of clean and corrupted speech signal spectrogram.

Besides non-stationary noise, *reverberation* in a corrupted speech signals has a substantial impact on speech quality, as well [3]. Reverberation is a superpo-

sition of several time-shifted and attenuated versions of the clean signal. The signals differ in delay and amplitude, which makes the transition between phonemes in the signal less distinct. The presence of noise and reverberation is much longer than the Short Time Fourier Transform (STFT) analysis window size. This causes these artifacts to smear across several frames, as shown in Figure 1.

### 2.2 Speech Enhancement Methods

Generally, speech enhancement techniques modify the signal in the frequency domain. These techniques only modify the magnitude of the STFT spectrum [9], which can then be used to reconstruct the signal along with the original phase.

Common speech enhancement methods include *spectral substraction* [10], which uses noise spectrum estimated during non-speech period for denoising and *linear filter-based methods* [3], which enhance the signal in the STFT or time domain.

#### Neural network-based methods

These approaches vastly outperform standard speech enhancement techniques and their usage is currently considered a standard [2, 4]. Both high-level features and raw speech can be used as an input and output of the network.

While the ASR performance in difficult noisy and reverberant conditions has significantly improved over the past years [2, 11], there still are certain areas of focus where such systems perform poorly. It has been observed that, when ASR systems are given a challenging environment with distant noisy and overlapping conversational speech, the system performance suffers significantly [12].

## 3. Neural Networks and Cycle Consistency

The main purpose of a neural network is to map input $X$ to another output $\tilde{Y}$, formally written as

$$F : X \longrightarrow \tilde{Y} \approx Y, \tag{1}$$

where the network $F$ attempts to produce an output that is similar to a reference sample $Y$ with respect to the cost function. We can indirectly improve the robustness of $F$ by enhancing the training process with a constraint, called *cycle-consistency*.

### 3.1 Cycle-Consistent Neural Network

Cycle-consistency is a technique initially used in machine translation and visual tracking, that can be applied to enforce additional constraints within the training framework [13, 14]. For example, when translating a sentence from language $A$ to language $B$, the machine should be able to transform the translated sentence back to the original sentence in language $A$. This form of cycle-consistency is called *forward-backward consistency*.

Cycle-consistency can be achieved by introducing an additional neural network to the framework. The network serves as an inverse mapping function

$$G : Y \longrightarrow \tilde{X} \approx X, \tag{2}$$

where $G$ is the neural network performing a *dual task* — attempting to produce $X$ when given $Y$ as an input. Both networks are then used in conjunction to be consistent with each other. The *forward cycle-consistency* objective aims to accurately reconstruct $X$, and can be defined as

$$X \longrightarrow F(X) \longrightarrow G(F(X)) \approx X. \tag{3}$$

Similarly, we can define *backward cycle-consistency*, where the goal is to reconstruct $Y$, as follows:

$$Y \longrightarrow G(Y) \longrightarrow F(G(Y)) \approx Y. \tag{4}$$

The constraints are enforced by adding the cost functions for (3) and (4) to the main objective function. A large advantage of this constraint is that it has no computational performance impact on use of the resulting model. Since network $F$ produces the wanted output, network $G$ is not used at beyond the training process. The full objective function is defined as follows:

$$\begin{aligned} L_{CSE} = \lambda_1 L(F) + \lambda_2 L(G) + \\ \lambda_3 L(F,G) + \lambda_4 L(G,F), \end{aligned} \tag{5}$$

where $L(F)$, $L(G)$, $L(F,G)$ and $L(G,F)$ is a cost function of $F$, $G$, a forward and a backward cycle, respectively, with $\lambda$s being weight coefficients. The training framework is shown in Figure 3.

Enforcing forward-backward consistency can improve the robustness of speech enhancement models [7]. In this work, which is based on [7], we implement the cycle-consistency framework for speech enhancement. Besides a standard neural network framework, we use the constraint together with *generative adversarial network* (GAN).

### 3.2 Generative Adversarial Networks

The goal during the training of a standard neural network is to produce an output that resembles a certain target output — the label. It is a *supervised* approach

to learning. However, neural networks require a significant amount of training data and having a labeled sample for each training input can be a costly task. The main advantage of *generative adversarial networks* (GANs) [5] is that it can be trained with data that does not contain input-target pairs (paired data). In GANs, two neural networks are pitted against each other, each attempting to reach its objective, which is 'adversary' to the other network. These models learn to model the probability distribution of data that resembles a given training set.

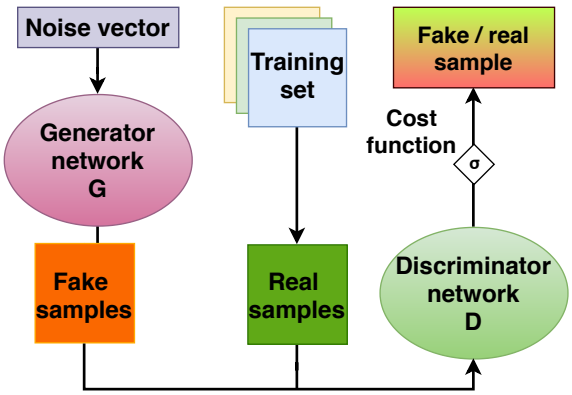



**Figure 2.** A structure of generative adversarial network.

Generative adversarial network consists of *two* neural networks, as seen in Figure 2. The generator, $G$, produces samples in the target domain, $Y$, given the generated noise (e.g., uniform noise) $Z$:

$$G : Z \longrightarrow Y. \tag{6}$$

Its adversary, $D$ attempts to recognize whether its inputs have been drawn from the training set or not. The output of the discriminator is defined as

$$D : X \longrightarrow < 0, 1 > . \tag{7}$$

Generative adversarial networks are primarily used for image vision problems. In this work, we attempt to couple GANs with cycle-consistency constraint to solve speech enhancement.

#### 3.2.1 CycleGAN

CycleGAN [6] is a GAN framework that uses a cycle-consistency loss to enable training without the need for paired data. It was originally proposed for image-to-image translation problems.

The goal of CycleGAN is to learn a mapping from the source domain to the target domain and vice versa. The framework consists of *four* neural networks in total — two generator-discriminator pairs. Forward and backward cycle-consistency losses are added to the cost function.

Additionally, the full cost function is extended with *identity mapping loss*. The generator networks are kept close to the identity mappings by the following constraints:

$$X \longrightarrow G(X) \approx X, \tag{8}$$

$$Y \longrightarrow F(Y) \approx Y. \tag{9}$$

The full objective function of CycleGAN is defined as

$$\begin{aligned} L_{CycleGAN} = \lambda_1 L(F,G) + \lambda_2 L(G,F) - \lambda_3 L_D(D_F) \\ -\lambda_4 L_D(D_G) + \lambda_5 L_I(F) + \lambda_6 L_I(G), \end{aligned} \tag{10}$$

where $L(F,G) + L(G,F)$ is a forward-backward cycle-consistency loss, $L_D(D_F)$, $L_D(D_G)$ are discriminator losses, and $L_I(F)$, $L_I(G)$ are $F$ and $G$ identity losses, respectively, with $\lambda$s being weight coefficients. The whole CycleGAN architecture used in this work is shown in Figure 4.

Unlike standard GANs, the generator networks in CycleGAN do not take a sample from random noise as input. Instead, the input is a specific piece of information, such as noisy speech utterance [15].

Recently, experiments using CycleGAN for single-channel speech enhancement problem have been conducted [7]. In this work, we apply the same architecture to evaluate performance for speech enhancement task.

## 4. Experiments

For training and evaluation, we use the CHiME-3 dataset [8]. The dataset incorporates Wall Street Journal (WSJ) corpus sentences spoken in challenging noisy environments, specifically in café (CAF), street junction (STR), public transport (BUS) and pedestrian areas (PED). The *real* data consists of 6-channel recordings of sentences spoken live in the environments. The *simulated* data was constructed by mixing clean utterances into environment background recordings. The training set consists of 9137 pairs of clean and simulated noisy training utterances. For testing, we use real noisy speech utterances from the development test set. We use recordings from the 5th channel, as was done in [7].

We evaluate three models in total — noisy-to-clean mapping network, which will serve as a baseline, a network with cycle-consistency, and a generative adversarial network with cycle-consistency. These models were proposed in [7]. For testing, only the noisy-to-clean mapping network portion of the models is used to produce enhanced speech utterances.

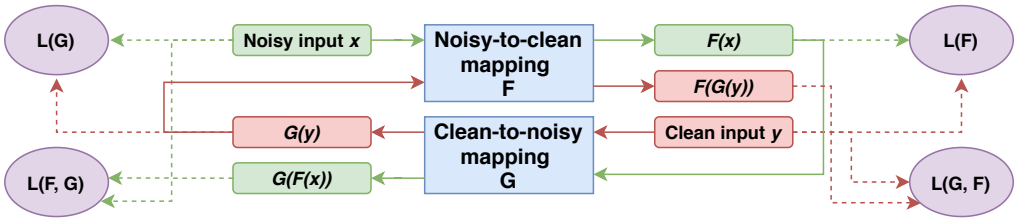



**Figure 3.** The architecture of cycle-consistency training framework for speech enhancement (CSE). Based on [7].

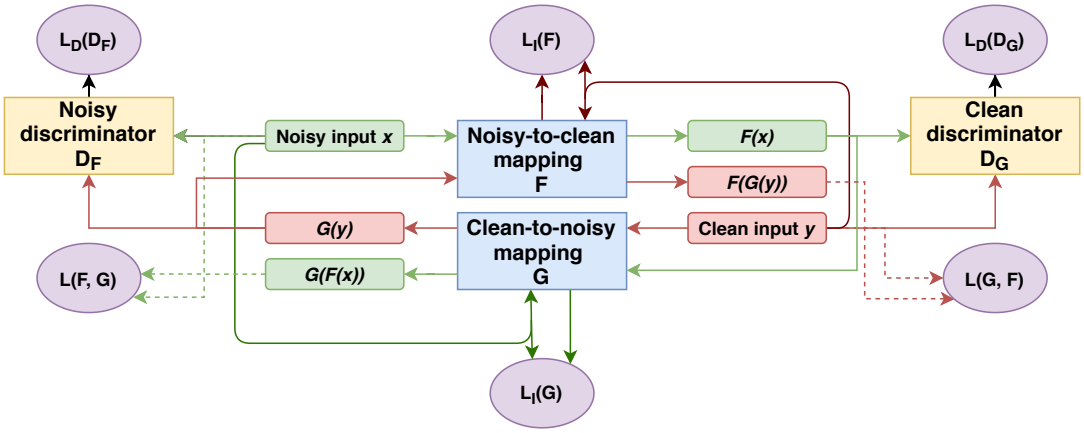



**Figure 4.** The architecture of adversarial training framework with cycle-consistency for speech enhancement (ACSE). Based on [7].

Training and evaluation was done using Sun Grid Engine[2] job scheduling system. The evaluation is performed by using a provided ASR system. ASR scripts and acoustic model re-training scripts were provided for this work. Training framework and the implemented neural network architectures are the author's own work.

## 4.1 Models Trained With Paired Data

This subsection describes models trained with *paired* data — each noisy utterance in the set has a corresponding clean utterance. We describe the training process of the baseline model and models with cycle-consistency constraints.

### 4.1.1 Baseline

Using standard supervised training, we first train a neural network for suppressing noise, $F$. The network input consists of log Mel-filterbank (MFB) features appended with first and second-order delta features, forming an 87-dimensional vector. The output is a 29-dimensional MFB without delta features. The network consists of two Long-Short Term Memory [16] layers followed by a linear layer. Each LSTM layer has 512 units. The input features were globally normal and mean variance normalized before being fed into the network.

[2]Sun Grid Engine - http://www.fit.vutbr.cz/CVT/cluster/SGE-UsersGuide.pdf

We heavily tuned network parameters in order to achieve satisfying results. In LSTM layers, forget gate biases are initialized to 1 (otherwise 0) [17]. The weights were initialized using Xavier normal distribution [18]. For optimization, we use AdamW algorithm [19]. The learning rate is set at $9 \cdot 10^{-4}$ and batch size is set at 48. The weight decay of AdamW is set at $1 \cdot 10^{-4}$. We find that using recurrent dropout in the first LSTM layer slightly lowers the model performance. We use Mean Squared Error (MSE) as a cost function.

This baseline setup slightly deviates from [7], in which the network was optimized by using stochastic gradient descent (SGD) optimizer. No weight initialization techniques nor any other training parameters were mentioned in the reference paper.

### 4.1.2 Forward and Backward Cycle-Consistency

We train a neural network, $G$, that inserts noise into clean speech utterance. The input and the output feature dimensions are 29 and 87, respectively. The learning rate is set at $8 \cdot 10^{-4}$. Other parameters and a cost function are the same as specified in 4.1.1.

Then, we use the pre-trained networks, $F$ and $G$, and jointly train them using cycle-consistency loss. We train the model with forward cycle-consistency and a model with both forward and backward cycle-consistency. When computing cycle-consistency loss, the input of one network is normalized before being fed to the other network. We set the learning rate at

$4 \cdot 10^{-4}$. The batch size is set at 24. The $\lambda$ loss function coefficients are the same as in [7].

While using a forward cycle alone has shown to degrade the model performance, having both forward and backward cycle constraints has shown to further improve the model's robustness. The training process of the best model converges in 7 epochs. We assume that, by pre-training $F$ and $G$ with carefully tuned hyperparameters, the networks adjust the weights to a relatively proper state rather quickly.

### 4.2 Models Trained With Unpaired Data

We use the same training set that contains noisy-clean sample pairs. However, the dataset is used as if it contained no related pairs. In practice, we take a batch of random noisy samples, a different batch of random clean samples, and work with these during the training iteration.

For generator networks, we use the same architecture as $F$ and $G$. The discriminator networks consist of two fully-connected hidden layers. Each hidden layer has 512 units. The output layer has 1 unit. The discriminators, $D_F$ and $D_G$, take 87-dimensional inputs (appended with delta features) and 29-dimensional inputs, respectively. $D_F$ and $D_G$ evaluate the probability of the input belonging to the noisy and clean set, respectively. We use AdamW optimizer for both generator and discriminator training.

Generally, GANs are difficult to train as a whole, as they can be very sensitive to changing hyperparameters. A large amount of minor training process adjustments was proposed [20, 21] that can significantly improve convergence and prevent common pitfalls, such as mode collapse [20].

Before beginning adversarial training, the generator networks need to be initialized in order to learn an underlying structure. Otherwise, the model would have trouble converging. From our experiments, these techniques were important to make the adversarial training converge:

- **initialization of generators** - The initialization is done by pre-training the generators as identity mapping functions — the target sample is the same as the input sample, but without normalization. The training hyperparameters for noisy-to-clean and clean-to-noisy generator networks are the same as of $F$ and $G$, respectively. The initialization procedure in [7] may differ, as the initialization details were not mentioned.
- **buffer of generated samples** - As suggested by Shrivastava et al. [21], we update the discriminators by using a history of generated utterances rather than the ones produced by the latest generators. We store two sample buffers of size 72 that keep previously generated noisy and clean samples. The original Cycle-GAN uses a history of 50 samples [6]. It is not specified whether the framework in [7] uses such technique.
- **one-sided label smoothing** - We modify the cross-entropy cost functions of the discriminators by employing one-sided label smoothing. Label smoothing is a regularization technique that prevents the discriminators from predicting the labels too confidently during training, which can result in poor generalization.

Using pre-trained generators, we perform adversarial training. The learning rate and weight decay are set at $1 \cdot 10^{-6}$. During each iteration, the discriminator networks are trained before the generator networks. The $\lambda$ loss function coefficients are the same as in [7].

## 5. Results

In this Section, we show the performance of our best-performing models for each category. We discuss the re-training of an acoustic model portion of the ASR system and discuss its impact on the system performance. We compare our approach to the reference publication. The results show that cycle-consistency plays a vital role in improving the model's robustness.

### 5.1 Models Trained With Paired Data

Due to carefully optimizing training hyperparameters and using proper weight and bias initialization methods, the baseline model alone reduces the ASR word error rate (WER) by 17.40% as opposed to no enhancement.

| Model | Noise Environment | | | | | |
|---|---|---|---|---|---|---|
| | BUS | PED | CAF | STR | Avg. | RWERR |
| None | 41.27 | 17.48 | 27.09 | 24.97 | 27.70 | - |
| Baseline | 28.91 | 16.36 | 27.58 | 18.68 | 22.88 | 17.40 |
| CSE-FW | 29.41 | 15.68 | 26.68 | 18.82 | 22.65 | 18.23 |
| CSE | **28.35** | **15.40** | **25.24** | **18.57** | **21.89** | **20.97** |

**Table 1.** The ASR WER (%) performance of real noisy test data in CHiME-3 enhanced by different models. Relative WER reductions (%) are shown in the last column. BUS, PED, CAF, STR refer to 4 different recording environments.

Further training the model with using only a forward cycle-consistency (CSE-FW) slightly boosts the model performance, up to 18.23% relative WER reduction (RWERR). The model with both cycle-consistency

constraints has shown to perform better, increasing the RWERR to 20.97%.

### 5.2 Models Trained With Unpaired Data

As shown in Table 2, we have reached 12.71% relative WER improvement over noisy data with our variation of CycleGAN (named ACSE), which is only slightly worse than the baseline from Table 1, which was trained with paired data.

| Model | Noise Environment | | | | | |
|---|---|---|---|---|---|---|
| | BUS | PED | CAF | STR | Avg. | RWERR |
| None | 41.27 | 17.48 | 27.09 | 24.97 | 27.70 | - |
| ACSE | **32.91** | **16.27** | **26.39** | **21.16** | **24.18** | **12.71** |

**Table 2.** The ASR WER (%) performance of real noisy test data in CHiME-3 enhanced by different models. Relative WER reductions (%) are shown in the last column.

### 5.3 Re-training the Acoustic Model

The authors of [7] performed acoustic model re-training in order to improve ASR performance. The acoustic model portion of the ASR is trained by taking speech recordings and their text transcriptions, from which a statistical representation of the sounds that make up each word is created. The ASR system provided for this work, trained using Kaldi[3] has DNN-HMM acoustic model, which we re-trained on speech utterances enhanced by our models. The dataset used for re-training the acoustic model is the same set that was used for training the neural networks.

In the publication, the re-training was performed on data enhanced from the adversarial model (ACSE), but not others. We have re-trained the acoustic model not only on ACSE, but also on baseline and CSE, as well.

| Acoustic Model | Architecture | | | |
|---|---|---|---|---|
| | Baseline | CSE | ACSE | ACSE ([7]) |
| Clean | 22.88 | 21.89 | 24.18 | 29.44 |
| Re-trained | 19.16 | 18.42 | **14.72** | 18.20 |

**Table 3.** Comparison of ASR WER (%) performances of speech enhancement models evaluated with clean and re-trained ASR acoustic model.

As seen in Table 3, re-training an acoustic model significantly boosts the performance, causing total relative WER reduction up to 46.86% for ACSE. Surprisingly, acoustic model re-trained using ACSE-enhanced speech samples shows biggest performance improvement. Similar improvements can be seen in ACSE ([7]). CSE and baseline only slightly improve the ASR performance. The possible reason for this is that while ACSE performs worse on a test set, it can generalize to unseen data better, and thus be a more appropriate candidate for re-training the acoustic model.

[3] Kaldi ASR - https://kaldi-asr.org/

### 5.4 Summary

To summarize our work, we present a table with relative WER reductions (RWERR) to overview our achieved results. The table depicts relative WER improvements over noisy data. The WERs of noisy data in our work and the publication are 27.70% and 29.44%, respectively.

| Model origin | Architecture | | | |
|---|---|---|---|---|
| | Baseline | CSE | ACSE | ACSE (re-trained AM) |
| Our work | **17.40** | **20.97** | **12.71** | **46.86** |
| Publication | 12.33 | 19.60 | 6.9 | 38.17 |

**Table 4.** Comparison of relative WER improvement (%) of models over noisy data.

Table 4 shows that by carefully picking hyperparameters and using various NN training enhancements, our models have performed significantly better compared to [7]. The table shows relative word error rate reduction over results obtained from data without any form of enhancement. While the models in [7] were evaluated using a different ASR system, the relative WER improvement is shown over WER of noisy data from the publication, which was 29.44%, whereas the noisy data WER in our work was 27.70%.

## 6. Conclusions

The goal of this work was to apply cycle-consistency constraint during the training process to improve the performance of speech enhancement models. The constraint only alters the process of training the target neural network, and the second neural network is not used in the evaluation/application phase.

We evaluated multiple models, whose goal was to enhance speech utterances using only a single channel. We trained models with paired data to extend standard NN with the constraint and unpaired data using a slight modification of the CycleGAN architecture.

Our results have shown that the cycle-consistency constraint significantly improved the performance of the models. Training with paired data, CSE has reached a relative WER reduction of 20.97% when compared to noisy data, while [7] achieves 19.60% RWERR. Our

Cycle-GAN variant, ACSE, achieved 12.71% RWERR on unpaired data, which is a significant improvement compared when to [7]'s 6.69 % RWERR. The models were used to re-train the acoustic model, which was then used to re-evaluate the ASR WER of those models. The baseline, CSE, and ACSE has reached 30.83%, 33.50% and 46.86% relative WER improvement over noisy data, respectively.

For future work, training features in the time domain, as opposed to the frequency domain, can be considered [22, 23], as certain information can be lost when transforming speech into higher-level features. Temporal convolutional neural networks [24] have recently been used with great success for speaker separation [25], but have not yet been much explored for speech enhancement.

## Acknowledgements

I would like to express my sincere gratitude and thanks to my supervisor, Ing. Kateřina Žmolíková for her valuable advice, support, and guidance.

## References

[1] Keisuke Kinoshita, Marc Delcroix, Sharon Gannot, Emanuël AP Habets, Reinhold Haeb-Umbach, Walter Kellermann, Volker Leutnant, Roland Maas, Tomohiro Nakatani, Bhiksha Raj, et al. A summary of the reverb challenge: state-of-the-art and remaining challenges in reverberant speech processing research. *EURASIP Journal on Advances in Signal Processing*, 2016(1):7, 2016.

[2] Shinji Watanabe, Marc Delcroix, Florian Metze, and John R Hershey. *New era for robust speech recognition: exploiting deep learning*. Springer, 2017.

[3] Takuya Yoshioka, Armin Sehr, Marc Delcroix, Keisuke Kinoshita, Roland Maas, Tomohiro Nakatani, and Walter Kellermann. Making machines understand us in reverberant rooms: Robustness against reverberation for automatic speech recognition. *IEEE Signal Processing Magazine*, 29(6):114–126, 2012.

[4] Zixing Zhang, Jürgen Geiger, Jouni Pohjalainen, Amr El-Desoky Mousa, Wenyu Jin, and Björn Schuller. Deep learning for environmentally robust speech recognition: An overview of recent developments. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 9(5):49, 2018.

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[6] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

[7] Zhong Meng, Jinyu Li, Yifan Gong, et al. Cycle-consistent speech enhancement. *arXiv preprint arXiv:1809.02253*, 2018.

[8] Jon Barker, Ricard Marxer, Emmanuel Vincent, and Shinji Watanabe. The third 'chime'speech separation and recognition challenge: Dataset, task and baselines. In *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 504–511. IEEE, 2015.

[9] Jont B Allen and Lawrence R Rabiner. A unified approach to short-time fourier analysis and synthesis. *Proceedings of the IEEE*, 65(11):1558–1564, 1977.

[10] Steven Boll. Suppression of acoustic noise in speech using spectral subtraction. *IEEE Transactions on acoustics, speech, and signal processing*, 27(2):113–120, 1979.

[11] Wayne Xiong, Jasha Droppo, Xuedong Huang, Frank Seide, Mike Seltzer, Andreas Stolcke, Dong Yu, and Geoffrey Zweig. Achieving human parity in conversational speech recognition. *arXiv preprint arXiv:1610.05256*, 2016.

[12] Jon Barker, Shinji Watanabe, Emmanuel Vincent, and Jan Trmal. The fifth'chime'speech separation and recognition challenge: Dataset, task and baselines. *arXiv preprint arXiv:1803.10609*, 2018.

[13] Di He, Yingce Xia, Tao Qin, Liwei Wang, Nenghai Yu, Tie-Yan Liu, and Wei-Ying Ma. Dual learning for machine translation. In *Advances in Neural Information Processing Systems*, pages 820–828, 2016.

[14] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Forward-backward error: Automatic detection of tracking failures. In *2010 20th International Conference on Pattern Recognition*, pages 2756–2759. IEEE, 2010.

[15] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[16] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[17] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *International Conference on Machine Learning*, pages 2342–2350, 2015.

[18] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[19] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[20] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242, 2016.

[21] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Joshua Susskind, Wenda Wang, and Russell Webb. Learning from simulated and unsupervised images through adversarial training. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2107–2116, 2017.

[22] Ashutosh Pandey and Deliang Wang. A new framework for supervised speech enhancement in the time domain. In *Interspeech*, pages 1136–1140, 2018.

[23] Ashutosh Pandey and DeLiang Wang. Tcnn: Temporal convolutional neural network for real-time speech enhancement in the time domain. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6875–6879. IEEE, 2019.

[24] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[25] Yi Luo and Nima Mesgarani. Conv-tasnet: Surpassing ideal time–frequency magnitude masking for speech separation. *IEEE/ACM transactions on audio, speech, and language processing*, 27(8):1256–1266, 2019.

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

# Best possible speech recognizer on your own data

Tomáš Sýkora*

**Abstract**
Many state-of-the-art results in different machine learning areas are presented on day-to-day basis. By adjusting these systems to perform perfectly on a specific subset of the general data, huge improvements may be achieved in their resulting accuracy. Usage of domain adaptation in automatic speech recognition can bring us to production level models capable of transcribing difficult and noisy customer conversations way more accurately than the general models trained on all kinds of language and speech data. In this work I present 12.7% word error rate improvement in our speech recognition task over the general domain speech recognizer from Google. The improvement was achieved by both very precise annotation and preparation of domain data and by combining state-of-the-art architectures and algorithms. The described system was successfully integrated into a production environment of the Parrot transcription company founded by, among other team members, current and former faculty students, which drastically increased performance of the human transcribers.

**Keywords:** automatic speech recognition — domain adaptation — conversational speech

**Supplementary Material:** *N/A*

*xsykor25@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

## 1. Introduction

Automatic speech recognition (ASR) has achieved many improvements over the last decade, similarly to other machine learning areas. New algorithms and more computational power allowed researchers to feed the systems with much more data than before, which markedly boosted results and accuracies of such systems. As a result, machine learning products can be now applied in various day-to-day scenarios where they automate the previously used manual processes and save both time and money of companies using them.

Automatic speech recognition is a system capable of transcribing speech from audio signal form to text. In Parrot, a court reporting transcription company, we integrate automatic speech recognition as a helper tool for our human transcribers. After obtaining a transcription request from a customer and generating the automatic transcription, the human transcribers fix errors in the ASR output using a specialised editor tool which is way more efficient than transcribing the whole recordings from scratch manually. To improve experience and efficiency of the human transcribers even more, we decided to build a more accurate automatic speech recognition system. As until now we've been using a general speech recognition, I show in this paper that a domain specific system can outperform the general one in terms of word error rate (WER).

The main reason for this being possible is no need of transcribing all kinds of target domains, acoustic setups and specific language dialects. Instead, we focus only on scenarios that occur commonly in our use case. Although this way our model performs worse in general, it performs way better on our target data where it knows the data well as it was trained on similar ones.

The goal of this work was to achieve higher transcription accuracy on our legal based audio conversations than the one obtained with the general domain model. This was accomplished by very precise data selection, annotation and preparation followed by the state-of-the-art language and acoustic models. Precise manual transcriptions cleaning, robust text normalization of unlabeled data, various audio augmentations, speaker adaptations, recurrent neural network language models and time-delayed neural architectures resulted in an ASR system with more than 12.7% improvement in WER in comparison to a general ASR model developed by Google[1].

## 2. Automatic speech recognition systems

To solve the large vocabulary continuous speech recognition (LVCSR) [1] problem, ASR systems stand on complex architectures composed of several subsystems. Specifically, acoustic and languages models combined together in a decoding network as depicted in Figure 1.

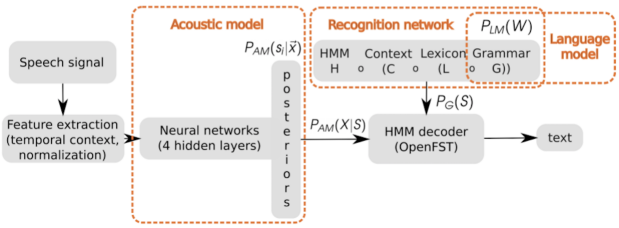



**Figure 1.** *Components of an automatic speech recognition system transcribing a speech signal to text (figure obtained from [2]).*

Acoustic models (AM) are trained to match every word in a speech signal to a sequence of acoustic units called phonemes. To do this, Hidden Markov models (HMM) [3] are one of the state-of-the-art approaches to solve this sub-problem. To simulate the human perception of speech where context plays an important role, language models (LM) predict probabilities of word sequences in a given language. The acoustic and language model combination can be expressed in a form of the Bayesian rule shown in Equation 1, where $P(X|W)$ is modeled by an acoustic model (probability of the acoustic observation $X$ given the word sequence $W$), while a language model estimates and models the second part of the rule $P(W)$ (apriori probability of a word sequence $W$). Assuming $P(X)$ is the same for all possible transcriptions, we do not have to consider it and only the numerator part of the rule remains to be realised by an ASR system.

$$W^* = \arg\max_w P(W|X) = argmax_w \frac{P(X|W)P(W)}{P(X)} \quad (1)$$

### Acoustic models

Current state-of-the-art approaches in HMM states representation use deep neural networks (DNN) [4] which replaced the previously used Gaussian Mixture Models (GMMs). However, GMMs are still found useful as a preprocessing step before the neural net training. As mentioned in [5], they help to force align phonemes in the individual speech utterances before sending them into an actual neural network based model.

The accuracy of a DNN during the training is approximated by an objective function. A basic objective function commonly used for a 1-of-$K$ classification problem is the multi-class cross-entropy (CE). However, later studies proved that replacing the cross-entropy objective function with sequence-discriminative criteria (e.g. maximum mutual information (MMI)) improves the overall results in the automatic speech recognition task as it is a sequence classification problem [6]. Further on, usage of lattice-free version of the MMI objective function (LF-MMI) decreases the computational costs by omitting the need of precomputation of the lattices for all possible word sequences (only the reference word sequence lattices remain) and avoiding the initial training with a CE model to create a precise weights initialisation [5].

Due to dynamic nature of speech ASR systems have to model temporal relationships between acoustic events, while at the same time providing for invariance under translation in time. To fight this, the time-delay neural network (TDNN) architecture [7] takes a window of both past and future feature frames to recognise a single phoneme. Additionally, as shown in Figure 2, later studies showed that not all connections between all frames are necessary, which significantly improves performance and lowers the amount of computations needed during training [8].

To further improve the prediction and generalization abilities of acoustic models, different speaker adaptations may be applied. i-vectors, low-dimensional features capable of characterizing speakers, are often

[1] https://cloud.google.com/speech-to-text/

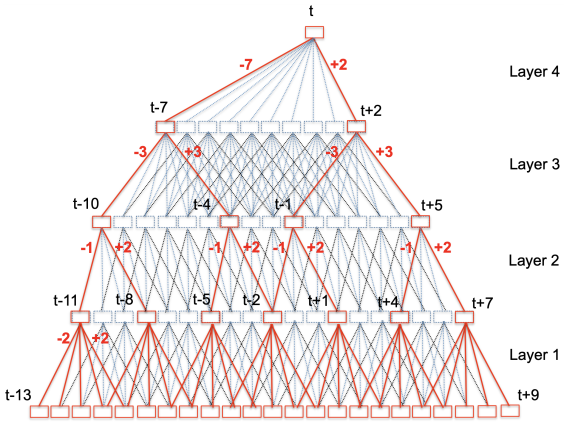



**Figure 2.** *Pruned time-delay neural network as presented in [8] computes with sub-sampling (red) instead of using all connections (blue+red).*

taken as additional input to an acoustic model besides the main (*mfcc*) features [9].

### Language models

Language models (LM) are applied to estimate word sequences apriori probabilities in a given spoken language (e.g. English). The most widely used statistical language model is the *n*-gram LM. Its goal is to identify occurrence counts of sets of *n*, *n*-1, ..., 1 words in a large text corpus. The *n*-gram model is then used to predict *n*-th word given a *n*-1 long word sequence. Additionally, techniques like Kneser-Ney smoothing [10] were introduced to improve the original *n*-gram algorithm. If multiple different domain text corpuses are available, building separate domain-specific language models for each corpus individually and applying an interpolation technique to combine them into a single language model optimised for the target use case showed better performance in comparison to training a single LM on a big mixed-text corpus [11].

To further increase the language modeling capacities, recurrent neural network (RNN) models come in place. Although RNN language models (RNNLMs) show better results in comparison to the *n*-gram models, because of the RNNLM theoretically infinite history input lengths, it is technically impossible to compile them into a static decoding graph. Thus RNNLMs are usually not directly used in the decoding. Instead, lattice rescoring is a common approach to take the advantage of the recurrent models in the decoding process [12]. After a word lattice is generated from the 1st-pass decoding, it is then rescored with an RNNLM.

As a conjunction of acoustic and language modelling, modelling inter-word silence probabilities proved to consistently improve the overall accuracy of ASR systems [13]. As an example, zero-silence is less likely to follow the word *White* in "Gandalf the White said," than in "The White House said.".

### Kaldi - speech recognition toolkit

Kaldi[2], introduced in [14], is a free open-source project consisting of many utilities for different parts of ASR pipelines from feature extraction, through GMM/DNN based HHMs training and speaker adaptation techniques, to decoding graphs and lattice rescoring algorithms implementations. Besides the individual components, algorithms and utilities, Kaldi also provides multiple prearranged pipelines called recipes containing state-of-the-art setups on different generally known speech datasets. Kaldi toolkit utilities were mostly used in this project.

## 3. Conversational speech dataset preparation

The data used to train both language and acoustic models had to perfectly represent the target domain. The accuracy of the data annotation had to meet high expectations. The whole training dataset for the acoustic model was composed of two subsets - the real data from the company customers (29 hours) and a dataset of the U.S. Supreme Court public hearings transcriptions[3] (the whole dataset contains 7000 hours but only a 500h subset was used in this project so far). Our target domain consisted of different audio quality types:

- Formal depositions - indoor environment, quiet room, mostly clear speech, high-quality microphone based on a table in front of the speakers.
- Witness statements - indoor environment, less formal environment, more spontaneous and sentimental discussion, middle quality microphones.
- Different less frequently represented scenarios - prison phone calls, police bodycams, street conversations etc.

### Manual annotation

Acoustic data annotation was performed in two steps. First, the whole recordings (5-400 minutes long) were transcribed in a way that suits the customer - with numerical characters, abbreviations, punctuation etc. Correct speaker labels were assigned to individual speech parts which segmented the audio into smaller speaker segments. In the second step, the speaker segments were automatically assigned with start/end timestamps using the *gentle*[4] force alignment tool. The timestamps

[2] https://kaldi-asr.org/
[3] https://www.supremecourt.gov/oral_arguments
[4] https://github.com/lowerquality/gentle

were thereafter manually corrected by human annotators in a proprietary designed annotation tool. Besides that, the human annotators were instructed to normalize the original transcriptions to a fully verbatim form (e.g. year 1950 replaced with 19 50 if pronounced as "nineteen fifty" etc.).

### Language model text corpuses

To train the language models (both *n*-gram and RNNLM), different text corpuses were experimented with. Firstly, the company transcriptions were used as one text corpus (17k sentences). Secondly, the whole Supreme Court hearings dataset transcriptions created the second text corpus (4M sentences). The rest consisted of general language text corpus (42M sentences subset of Google Billion Word Benchmark [15]), a set of legal deposition transcriptions collected on the Internet (3M of conversational data) and a set of publicly available court decisions[5] (20M sentences of non-conversational data).

### Automatic data preprocessing

As only the company transcriptions were manually normalized and transformed to the fully verbatim form, automatic normalization had to take place for all texts available. The normalization replaced all numerical values and special characters like dollars and percents with their alphabetical equivalents in an appropriate way (*'70s* as *seventies*, *$100,000* as *one hundred thousand dollars* etc.). The letter case was processed only at the sentence beginnings, the rest of words remained with the original case; each word at a sentence beginning was decided to remain capital or was lowercased with help of a pretrained named entity recognition tool *spacy*[6]. All punctuation was removed except in special cases like *hm-mm* or *uh-huh* and single quotes (*don't*). Abbreviations were split into single letters and several other minor preprocessing steps were performed.

Input text:

*MMy name Bond, james bond. A CIA agent. Ian Flaming introduced me in 1953 in a Casino Royale novel, later filmed in 2006 for $10 million.*

Text after manual correction and partial normalization:

*My name is Bond, James Bond. A CIA agent. Ian Flaming introduced me in 19 53 in a Casino Royale novel, later filmed in 2006 for $10 million.*

Text after automatic normalization:

*my name is Bond James Bond a C I A agent Ian Flaming introduced me in nineteen fifty three in a Casino Royale novel later filmed in two thousand six for ten million dollars*

Due to the LF-MMI objective function sensitivity to incorrect transcripts, the Supreme Court acoustic data were cleaned from the utterances with supposedly incorrect transcriptions. The cleaning was performed using a *kaldi* script[7]. The script removed utterances which, after decoding with a biased LM (trained on the transcriptions themselves), the lattice oracle path was still far from the transcript. This way, 12% of the acoustic data was considered to be incorrect and thus removed from the dataset. The removed utterances often contained crosstalks or the audio was cut in the middle of the utterance text.

### Data augmentation

To increase the size of the company dataset and for the Supreme Court data to better match the target acoustic setup, all of the data underwent an augmentation procedure using speed perturbation and the room impulse responses dataset described in [16] (RIR noises). In the paper, 3-fold speed perturbation was used in all experiments. Additionally, improvements of both 2 and 3-fold augmentations using RIR noises in addition to speed perturbation were presented. As our company dataset size was too small at the moment of training, 3-fold speed perturbation was applied to create different speed versions of the original data. Thereafter, the amount of augmented data with RIR noises was increased from 3 to 4-fold in comparison to the original paper. Supreme Court dataset was doubled using RIR noises. No speed perturbation took place as the amount of the Supreme Court data was sufficient. All mentioned augmentation procedures were performed using appropriate *kaldi* scripts.

Additionally, all utterances were concatenated with randomly selected 30ms silence audio chunks (30ms to both start and end of an utterance) to prevent HMM first and last silence representing states from assigning speech phonemes to them, which would be the case for utterances where there were no silence frames at their beginnings and/or ends.

[5] https://case.law/bulk
[6] https://spacy.io/
[7] https://github.com/kaldi-asr/kaldi/blob/master/egs/wsj/s5/steps/cleanup/clean_and_segment_data_nnet3.sh

## 4. Experiments

Throughout the experiments, the company dataset was split into an 11.5 hour test set and an 18 hour train set, both with unique speakers. 500 hours were separated from the U.S. Supreme Court dataset to increase the train set size and another 60 hours Supreme Court subset was created as a second test set. Both company and Supreme Court sets were augmented. The company 18 hours were augmented to 220 hours using 4-fold RIR noises and 3-fold speed perturbation. The Supreme Court data were augmented from 500 hours to 990 hours using 2-fold RIR noises augmentation. The separate improvements are described below.

### Language models

Several LM experiments were performed to prove that the interpolation of several narrow domain *n*-gram LMs can outperform one large *n*-gram LM. First, four different-domain 4-gram language models were trained on the text corpuses described in Table 1. An open-source tool KenLM[8] was used to train the individual LMs and the final interpolated LM.

**Table 1.** 4-*gram language models tested on a subset of 9k sentences of the company transcriptions not seen during the training. Log-linear weights were assigned by the interpolation procedure using the KenLM tool.*

| Dataset | Size | Weight | Perplexity |
|---|---|---|---|
| General | 42M | 0.2 | 510 |
| Legal general | 23M | 0.1 | 427 |
| Supreme Court | 4M | 0.2 | 407 |
| Parrot transcriptions | 17k | 0.5 | 146 |
| Interpolated LM | - | - | 126 |

In another experiment, the same text corpuses were used differently. The out-of-domain corpus (Google Billion Words) was not used at all. The Supreme Court dataset was split into two subsets, transcriptions before and after the year 2000. Finally, the legal general corpus was split into two subset, dataset of Court Decisions (non-conversational texts) and legal speech transcriptions collected on the Internet (conversational texts). Another difference in comparison to the previous LM experiment was improved text normalization capable of correct handling of more specific use cases than the previous normalization version. To sustain the comparability of the test set, the original normalization was applied on it in the second experiment.

Two different RNNLMs for lattice rescoring were trained on the same data splits used by the *n*-gram LMs. Their contributions are shown the tables 3 and 4.

[8] https://kheafield.com/code/kenlm/

**Table 2.** 4-*gram language models tested on a subset of 9k sentences of company transcriptions not seen during the training. Log-linear weights were assigned by the interpolation procedure. The text corpuses from in Table 1 were differently divided and more robust normalization was applied on them. The robust normalization removed a big part of the Court decisions corpus.*

| Dataset | Size | Weight | Perplexity |
|---|---|---|---|
| Court decisions | 15.5M | 0.2 | 340 |
| Legal transcriptions | 2.5M | 0.55 | 176 |
| Supreme Court | 1.5M | -0.1 | 337 |
| Supreme Court from before 2000 | 2.5M | -0.1 | 340 |
| Parrot transcriptions | 17k | 0.45 | 139 |
| Interpolated LM | - | - | 123 |

### Acoustic models

The acoustic model architecture was based on the kaldi Switchboard recipe[9]. Firstly, a sequence of GMM based models was trained, each of which used phoneme level alignments from the previously trained GMM model. Starting with a monophone training on a subset of the shortest utterances (to minimize incorrect phoneme alignments), continuing with bigger models trained on bigger subsets, ending with a triphone GMM model with speaker adaptation, which thereafter aligned the utterance phonemes before being passed into the TDNN model.

**Table 3.** *Convolutional time-delay (cnn-tdnnf) models trained on the cleaned and augmented Supreme Court dataset (s.c., 500h augmented to 990h) and 18 hours of our company data. The 18 hours of Parrot data were extended by augmentations to 220h. test1 is an 11.5 hour test set of the company data, test2 is a 60 hour subset of the Supreme Court dataset, both not seen during the training.*

| | Training info: | | WER [%] | |
|---|---|---|---|---|
| **AM** | **LM** | **Dataset** | test1 | test2 |
| google | unknown | unknown | 35.0 | - |
| cnn-tdnnf | 4-gram | 990h s.c. | 38.3 | 13.8 |
| cnn-tdnnf | 4-gram + lat. rescoring | 990h s.c. | 34.9 | 15.1 |
| cnn-tdnnf | 4-gram | 18h parrot + 200h aug. + 990h s.c. | 27.63 | 13.96 |
| cnn-tdnnf | 4-gram + lat. rescoring | 18h parrot + 200h aug. + 990h s.c. | **24.21** | - |

[9] https://github.com/kaldi-asr/kaldi/tree/master/egs/swbd

Experiments with different data setups were run which is demonstrated in the tables 3 and 4.

**Table 4.** *Improved 4-gram language model from Table 2 and a new RNNLM trained on the newly normalized corpuses from the same table used with the same AM as in the Table 4. test1 is the company 11.5h test set same as the test1 from Table 3. test3 is the same set except with the improved text normalization.*

| Training info: | | | WER [%] | |
|---|---|---|---|---|
| **AM** | **LM** | **Dataset** | test1 | test3 |
| google | unknown | unknown | 35.0 | 35.1 |
| cnn-tdnnf | 4-gram (v2) | 18h parrot + 200h aug. + 990h s.c. | 24.65 | 24.42 |
| cnn-tdnnf | 4-gram (v2) + lat. rescoring | 18h parrot + 200h aug. + 990h s.c. | 22.59 | **22.32** |

The best WER was achieved by decoding with the beam size 15, lattice pruning beam 12 and rescoring lattice beam 12. Higher values resulted in around 0.01% improvement but drastically increased the computing time thus the lower beam values seem to be the most reasonable.

## 5. Conclusions

A combination of proven state-of-the-art automatic speech recognition components was presented in this work. The paper demonstrates how to prepare conversational speech data from a specific domain to achieve higher accuracy than a generic ASR model. With only 18 hours of the target domain data in combination with a public similar domain dataset of 500 audio hours, a 12.7% WER improvement was gained in comparison to a general ASR model from Google. The paper shows how the individual components and data preparation steps help with increasing the resulting accuracy of the system.

Improvements of several of the mentioned components weren't measured separately thus it's not clear how big contribution they brought. Such experiments have to be run in the future. Also, no time was spent to tune the hyper-parameters of the individual components which can become an object of the future research. Additionally, a bigger target domain dataset could boost the final results even more.

## Acknowledgements

I would like to thank my supervisor Ing. Igor Szőke Ph.D. for his help and advice in this project.

## References

[1] Steve Young. A review of large-vocabulary continuous-speech. *IEEE Signal Processing Magazine*, 13(5):45–, Sep. 1996.

[2] Karel Veselý. The project kaldi – open source speech recognition. online, 2018.

[3] Mark Stamp. A revealing introduction to hidden markov models, 2004.

[4] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, Nov 2012.

[5] Daniel Povey, Vijayaditya Peddinti, Daniel Galvez, Pegah Ghahremani, Vimal Manohar, Xingyu Na, Yiming Wang, and Sanjeev Khudanpur. Purely sequence-trained neural networks for asr based on lattice-free mmi. pages 2751–2755, 09 2016.

[6] Karel Veselý, Arnab Ghoshal, Lukáš Burget, and Daniel Povey. Sequence-discriminative training of deep neural networks. In *Proceedings of Interspeech 2013*, number 8, pages 2345–2349. International Speech Communication Association, 2013.

[7] Alex Waibel, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano, and Kevin J Lang. Phoneme recognition using time-delay neural networks. *IEEE transactions on acoustics, speech, and signal processing*, 37(3):328–339, 1989.

[8] Vijayaditya Peddinti, Daniel Povey, and Sanjeev Khudanpur. A time delay neural network architecture for efficient modeling of long temporal contexts. In *INTERSPEECH*, 2015.

[9] G. Saon, H. Soltau, D. Nahamoo, and M. Picheny. Speaker adaptation of neural network acoustic models using i-vectors. In *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, pages 55–59, Dec 2013.

[10] Reinhard Kneser and Hermann Ney. Improved backing-off for m-gram language modeling. *1995 International Conference on Acoustics, Speech, and Signal Processing*, 1:181–184 vol.1, 1995.

[11] Ernest Pusateri, Christophe Van Gysel, Rami Botros, Sameer Badaskar, Mirko Hannemann, Youssef Oualil, and Ilya Oparin. Connecting and

comparing language model interpolation techniques. *arXiv preprint arXiv:1908.09738*, 2019.

[12] Hainan Xu, Tongfei Chen, Dongji Gao, Yiming Wang, Ke Li, Nagendra Goel, Yishay Carmiel, Daniel Povey, and Sanjeev Khudanpur. A pruned rnnlm lattice-rescoring algorithm for automatic speech recognition. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5929–5933. IEEE, 2018.

[13] Guoguo Chen, Hainan Xu, Minhua Wu, Daniel Povey, and Sanjeev Khudanpur. Pronunciation and silence probability modeling for asr. In *INTERSPEECH*, 2015.

[14] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukáš Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlíček, Yanmin Qian, Petr Schwarz, Jan Silovský, Georg Stemmer, and Karel Vesel. The kaldi speech recognition toolkit. *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*, 01 2011.

[15] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling, 2013.

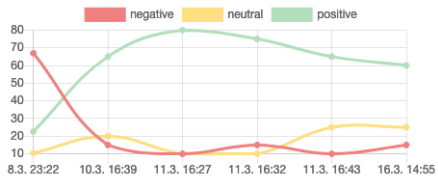
[16] Tom Ko, Vijayaditya Peddinti, Daniel Povey, Michael L Seltzer, and Sanjeev Khudanpur. A study on data augmentation of reverberant speech for robust speech recognition. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5220–5224. IEEE, 2017.

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

# Estimation of Emotions from a Text and Analyzing Tweets

Aneta Dufková*




WHAT DOES TWITTER THINK .com

**Abstract**
This paper describes the process of estimation of emotions from a text using machine learning. Negative, positive, and neutral emotions are recognized from tweets focused on various topics. The whole work is not only about machine learning, but also about natural language processing. It involves data gathering, preprocessing of obtained texts, and a lot of experimenting with both model and dataset.
The final model has been used to create a simple web application whatdoestwitterthink.com, which allows user to discover what do people on Twitter think. User can write a topic, the app downloads tweets related to this topic in real time, and analyzes them. The application also collects feedback from users to improve the classifier.

**Keywords:** sentiment analysis — tweet sentiment — estimation of emotions

**Supplementary Material:** Demonstration Video — Website

*xdufko02@fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

## 1. Introduction

Estimation of emotions, in other words sentiment analysis, is a very important problem. There are many companies which would like to know what does the public think about them or their new product [1]. Hoteliers want to know whether their customers are satisfied with their services or not [2]. It is not necessary, however, to only look for examples in business. Everyone can sometimes be curious about what do other people think about some topic, and this work aims to provide a simple way to discover it.

In more detail, the main goal of this work is to find convenient data, preprocess them and build a sentiment classifier using neural network. It can recognize positive, neutral and negative emotions, and it is built especially for classifying tweets. Pre-trained model is used in a simple web application whatdoestwitterthink.com which demonstrates practical use of sentiment analysis and can be useful for individuals and businesses.

There are many existing solutions of sentiment analysis, both academic and commercial. Some of them are able to recognize wide spectrum of emotions like anger, frustration, anxiety, or happiness. Unfortunately, academic work is usually not accessible for public and commercial solutions are almost always paid. One good exception is Sentiment140[1] from Stanford University. It has an API for classifying tweets, but it does not have a nice user interface, therefore, it is useful primarily for developers. This is the added value of this work. Its result is useful for everyone, not only developers or researchers. Furthermore, the resulting application collects feedback from users. It can be eligible for further improvements and experiments.

[1] http://www.sentiment140.com/

## 2. Background

Social media, including Twitter, are one of the reasons why popularity of sentiment analysis increases [3] and approach to sentiment analysis is evolving.

### 2.1 Machine Learning

In the past, lexicon based approach, which determines emotion by polarity of the words, had been widely used.

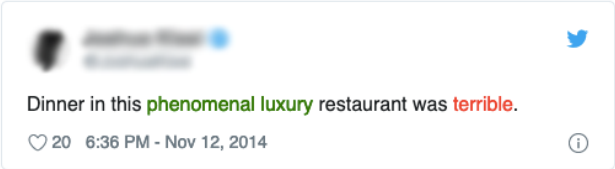



**Figure 1.** This is the example of tweet, where simple lexicon based sentiment analysis fails. There are two positive words and only one negative, though overall sentiment should be negative.

Today, it is common to use machine learning and train classifiers with features such as unigrams or bigrams [4]. Algorithms like Naive Bayes, Maximum Entropy or Support Vector Machines can be also used. This work uses neural network.

When working with text, specific case of RNN (Recurrent Neural Networks), called LSTM (Long Short Time Memory) networks, is popular [5]. LSTM networks are able to remember context. Another suitable type are CNNs (Convolutional Neural Networks). This work experiments with both of them.

### 2.2 Natural Language Processing

Text has to be preprocessed, in order to get rid of unnecessary parts (for example URLs, user mentions in tweets). Preprocessing will be discused in section 3.2. After that, a dictionary is built, where every word receives its own index. Finally, the word embeddings are created.

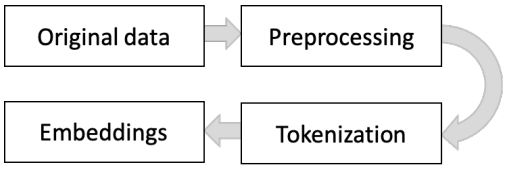



**Figure 2.** Text has to be preprocessed, transformed into tokens and embeddings.

Embeddings are able to formulate relationships between words. Even though deep learning libraries usually allow to use built-in embedding layer, it is common to use pre-trained models. One of them is Word2vec developed by Tomas Mikolov et al. [6]. Word2vec model trained on GoogleNews articles is used in this work[2].

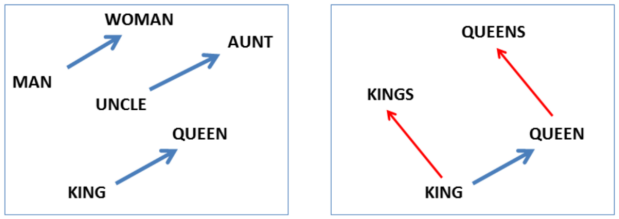



**Figure 3.** Word2vec can map relations between genders or singular and plural forms. Image taken from [6].

## 3. Data Gathering And Preprocessing

Tweets are very unique texts in many ways [7]. They are short, contain misspellings and slang.

### 3.1 Appropriate dataset

Initially, this work used data from aforemetioned project Sentiment140. It contains 1,6 milion tweets annotated as positive or negative according to occurrence of positive and negative emoticons. Number of positive and negative tweets is equal. Testing dataset, containing 500 manually annotated tweets, was also prepared within this project.

The tweets have been cleaned, which means they do not contain URLs, mentions of other users and emoticons. Missing emoticons appeared as a complication, which is demonstrated in figure 4. Moreover, there are only positive and negative tweets in training data, neutral class is missing.

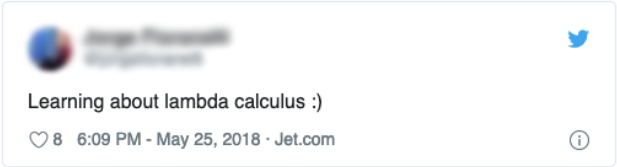



**Figure 4.** This tweet without emoticon would be considered as neutral, maybe negative for someone. But it is definitely positive with emoticon.

For that reason, additional datasets have been also used. It is quite simple to find a dataset consisting of tweets, but neutral emotion is almost always missing, although neutral class in training data is important [8].

An experimental approach was to download large dataset of tweets from Kaggle [3] without annotation and annotate them using Sentiment140 API. The results were good and most importantly, there was a neutral class.

[2] code.google.com – pre-trained Word2vec model
[3] Kaggle.com– Customer Support on Twitter

In the end, manually annotated data were added. Details about how different datasets influenced the results are listed in the section 4.

### 3.2 Data cleaning

Tweets are noisy. It is necessary to preprocess them, remove useless parts and try to normalize them. Usually, lemmatization and stemming are used to normalize words. Simply stated, stemming just chops off the end of words. Lemmatization is more powerful, therefore it is used in this work.

| Original word | Stemming | Lemmatization |
|---|---|---|
| Caring | Car | Care |
| Stripes | Strip | Strip / Stripes |

**Table 1.** Lemmatization considers the context. The problem of stemming is that more words can lead to the same form.

Common approach is to remove stop words, which do not contain any emotion (what, where, etc.). It can reduce the size of vocabulary, unfortunately, it slightly worsened the results, as it is shown in table 2.

After experimenting, the final preprocessing includes lowercase conversion, deleting user mentions, URLs, letters RT indicating retweet, some punctuation marks and transformating one or more occurence of character to two occurences. Thanks to that, "thaaaaanks" becomes "thaanks", which is something more positive than "thanks".

**Figure 5.** This is an example of preprocessing.

## 4. Implementation And Evaluation

In the beginning, LSTM newtork and 1,6 milion tweets from Sentiment140 were used for training and testing. Since only positive and negative tweets are present in training dataset, it was only binary classification. No pre-trained embeddings were used. For testing, there were two sets. The first one contained 1500 unseen tweets from original 1,6 milion. The second one contained only manually annotated tweets, also only positive and negative. Accuracy on both of these sets were almost the same – 80 % and it did not increase with any of the made experiments.

Training LSTM networks is computationally expensive, hence the experiments were limited to fewer architectures and hyper-parameters. This was the main reason why, all things considered, different type of neural network was used for another experiments. It was observed that the performance of CNN classifier is comparable to that of LSTM. Moreover, CNN models are computationally inexpensive.

CNN has an embedding layer, which uses pre-trained Word2vec model. Sequences obtained from tokenizer are its input. It maps each word of a tweet to a feature vector and outputs matrix. Then, the network consists of five convolutional layers which capture contextual information. Each of them is followed by max-pooling layer, extracting the largest values and reducing size. After passing through these layers, dense and dropout layers are used. The last dense layer uses softmax activation function, which outputs the probability value for each class.

At this stage, it turned out that it is necessary to handle the neutral emotion. One of the possible solution was not adding neutral training data and considering those sentences, whose probability for positive and negative classes is similar as neutral. This approach was not successful, as it is illustrated in table 2.

Another idea was to change the dataset, experiment with representation of classes and sample weights. Different methods of handling emoticons were used. One of them was representing emoticons as text – for example "emojistart grinning face emojiend".

| | 1 | 2 |
|---|---|---|
| **Data** | Positive and negative only, threshold for neutral set manually | Stop words removed |
| **Acc. on validation dataset** | 81 % | 78 % |
| **Acc. on manually annoted dataset** | 58 % | 59 % |

**Table 2.** These table shows two experiments with poor results. The first one involved only positive and negative training data, although there was neutral class in testing dataset. The second one used stop words removing technique.

Several datasets were used for other experiments. Finally, the best results were obtained from classifier,

| | 1 | 2 | 3 |
|---|---|---|---|
| **Dataset** | Customer Support on Twitter | Customer Support on Twitter + 1000 manually annotated | mixed + emoticons |
| **Dataset details** | | same number of positive, negative, neutral tweets | emoticons in text replaced by “emojistart emojiname emojiend” |
| **Acc. on validation dataset** | 86 % | 85 % | 80 % |
| **Acc. on manually annotated dataset** | 54 % | 62% | 53 % |

**Table 3.** This table compares different experiments. All of them used dataset mentioned in section 3.1 annotated with Sentiment140, the second and third ones added another data and used sample weights. The second one had the same number of samples from each class, the third one replaced emoticons.

which worked with dataset, whose sentiment was annotated with Sentiment140 API. Considering the fact that testing dataset is also from project Sentiment140, it is reasonable.

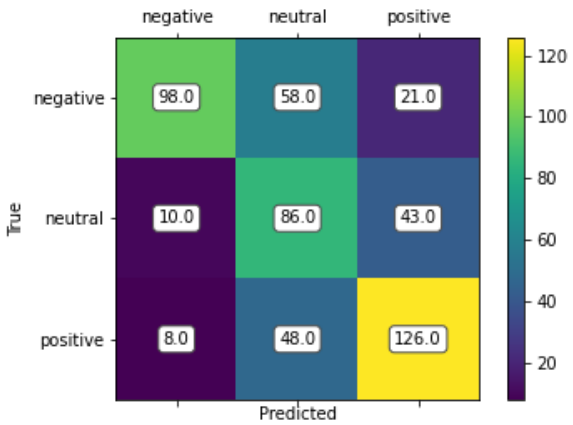



**Figure 6.** Confusion matrix (belongs to experiment 2 mentioned in table 3) shows that neutral class has the most false positives.

62 % seems not to be great, but this number alone actually does not say almost anything. It expresses how good the model is on this particular dataset from Sentiment140. The problem is, this dataset is from 2009 and it might be considered as slightly different from modern tweets. For example, modern tweets contain emoticons which did not exist 11 years ago. Therefore, further testing on different datasets is needed to improve the classifier.

Above that, the web application collects feedback from users and it is planned to use this data to improve the classifier. More about the method of collecting feedback can be found in section 5.

## 5. Web Application

Pre-trained model was used to create a web application whatdoestwitterthink.com. This application is based on client-server model.

Client is represented by a simple webpage implemented with HTML, CSS, and JavaScript. Python web server was made with Flask framework. Its task is to download data from Twitter, preprocess, and classify them. It uses pre-trained model and tokenizer.

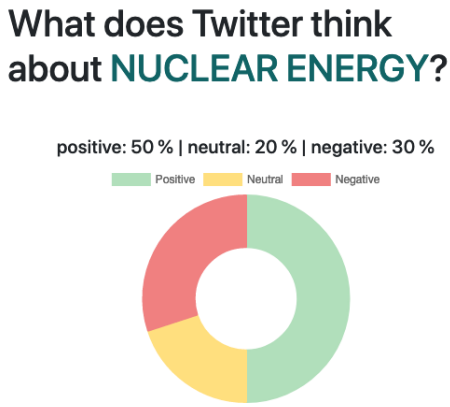



**Figure 7.** The main task of the web application is to classify how many tweets are positive, negative and neutral.

When a user writes down a topic, website sends a request to Flask server. It downloads data, classifies them, and sends a response in JSON format. Website processes this JSON and displays data as charts etc. It also shows examples of classified tweets.

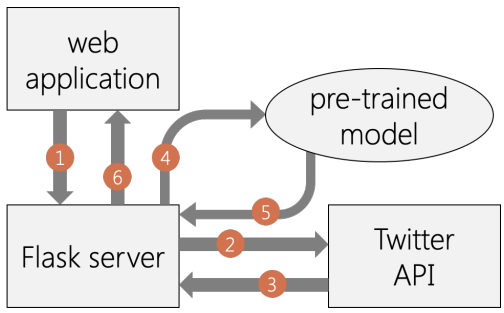



**Figure 8.** After user writes down a topic, the request to a Flask server is sent (1). Server sends request to Twitter API (2), API returns tweets in JSON format (3). Flask server uses pre-trained model to classify tweets (4, 5) and sends a response to web application in JSON format (6).

The Flask server uses SQLite database, therefore, the history of searches is saved. The app is able to display last searches and how were emotions changed during time.

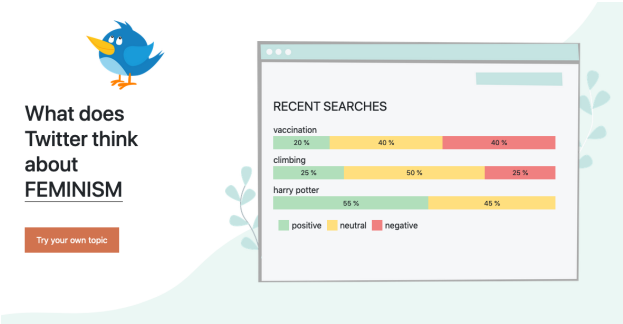



**Figure 9.** Since the web application uses SQLite[4], it is able to display recent searches.

The application shows examples of classified tweets; and users can correct them, if they are classified incorrectly. Corrections are saved in a database for further improvements of classifier.

## 6. Conclusions

This paper describes one of the possible approaches to sentiment analysis – training neural network. It also includes basic techniques of text preprocessing.

Pre-trained model is then used in web application whatdoestwitterthink.com, which allows its users to analyze sentiment from tweets and collects feedback from users. Target group of this application can be anyone, since it was made for the general public.

There are many opportunities for improvement. First of all, it is necessary to test classifier on different datasets. Additionally, the classifier is not able to cope with sarcasm, tweets containing both positive and negative emotions etc.. Moreover, there are tweets in which it seems impossible to decide whether they are positive or negative.

**Figure 10.** The user agrees (positive). But he is also angry where were American values (negative).

The main goal right now is to collect feedback from users and use it for additional improvements.

[4] SQLite.org – C-language library that implements a SQL database engine

## Acknowledgements

I would like to thank my supervisor Ing. Igor Szőke, Ph.D. for his help and assistance, Ing. Ondřej Novotný for his advices and provided data and Ing. Sebastián Poliak for many ideas what to try and how to improve my work.

## References

[1] Apoorv Agarwal, Boyi Xie, Ilia Vovsha, Owen Rambow, and Rebecca Passonneau. Sentiment analysis of twitter data. *Proceedings of the Workshop on Languages in Social Media*, 01 2011.

[2] Walter Kasper and Mihaela Vela. Sentiment analysis for hotel reviews. *Speech Technology*, 4:96–109, 11 2012.

[3] Bing Liu and Lei Zhang. *A Survey of Opinion Mining and Sentiment Analysis*, pages 415–463. Springer US, Boston, MA, 2012.

[4] Lei Zhang, Riddhiman Ghosh, Mohamed Dekhil, Meichun Hsu, and Bing Liu. Combining lexicon-based and learning-based methods for twitter sentiment analysis. 01 2011.

[5] Chunting Zhou, Chonglin Sun, Zhiyuan Liu, and Francis C. M. Lau. A C-LSTM neural network for text classification. *CoRR*, abs/1511.08630, 2015.

[6] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, Atlanta, Georgia, June 2013. Association for Computational Linguistics.

[7] Alec Go, Richa Bhayani, and Lei Huang. Twitter sentiment classification using distant supervision. *Processing*, 150, 01 2009.

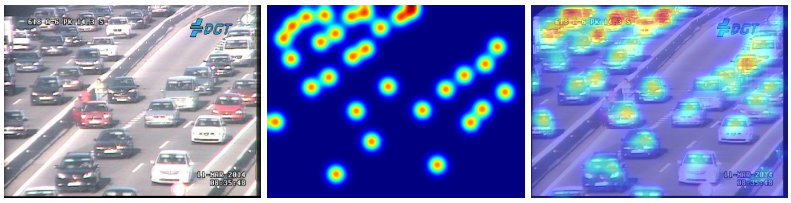
[8] Moshe Koppel and Jonathan Schler. The importance of neutral examples for learning sentiment. *Computational Intelligence*, 22(2):100–109, 2006.

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

# Counting Vehicles in Image and Video

Dominik Gabzdyl*



**Abstract**
Traffic analysis is still a challenging task. During such task there are many pitfalls to be aware of. Such as small image resolution, high number of overlapping objects, angle of camera, blurred objects due to their motion or weather conditions. This paper addresses the issue of counting vehicles instances in images and videos. Remarkable results and state-of-the-art methods are defined by convolutional neural networks. There are many approaches to address the issue of counting objects in images. One of which is counting by regression, which is the aim of this paper. I propose an architecture which is inspired by some state-of-the-art models. The proposed model improves accuracy on various datasets. For instance on the very small PUCPR+ dataset the Root Mean Square Error between expected and predicted vehicle counts was reduced from 34.46 to 8.84 vehicles (measured on the test set).

**Keywords:** vehicle counting — counting by regression — convolutional neural networks

**Supplementary Material:** *N/A*

*xgabzd00@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

## 1. Introduction

Before the rise of Artificial Intelligence, vehicle detection task was carried out via hardware solutions. Inductive loops [1] solution, for instance, is one of them. Vehicle passing through such loop would cause change in magnetic field thus signalizing its presence. However such hardware solutions are costly. Recent major rise in computer vision presented approaches with high accuracy and low implementation cost.

In this paper I address the hard problem of precisely counting objects instances within an image (or video). The correct detection of an object is severely impacted by many factors. Some of which include extremely overlapping objects, distorted image, bad weather conditions, high density of objects, etc.

Data acquired from automated traffic flow analysis can improve management of traffic. Busy highways or road intersections can be monitored for traffic congestions to see how they evolve. Public authorities in charge of the maintenance and planning of road infrastructures can make use of such information.

## 2. Related work

Counting by regression belongs to supervised learning category. In supervised learning a ground truth (expected output) must be provided. One technique of the counting by regression approach is to use heatmaps. In this technique a pair of input image and its expected output heatmap image is provided to model. The training phase of model is then basically learning mapping between ground truth images and input images.

Following the idea of Lempitsky et al. [2] the counting problem is then a process of recovering a density function as a real function of pixels of input images. Predicted heatmap patch of objects is the output from a network of this approach. Expected

heatmap patch is made by applying Gaussian blur with a reasonable standard deviation value $\sigma$ (in this work $\sigma = 15$) centred on each dot in the annotated image.

However, this will get us a full ground truth image. We need to split input images and ground truth images into patches. Note that each input and ground truth patch pair must have matching coordinates (their patch coordinates must correspond).

Gaussian blur uses a kernel whose values add up to 1. This ensures no energy is added or removed from the image after this operation. Thus, the sum of convolved dots (pixels) in the annotated image is unchanged (the vehicle count is preserved). To get the final count we simply sum up the density map.

In this section I will also discuss architectures which inspired my work. I will provide a brief summary of each. I will also mention pros and cons where possible.

### 2.1 Multi-column Deep Neural Networks for Image Classification

This network architecture [3] is inspired by vertical column-like arrays of neurons (located in the temporal cortex of human).

The model is made of vertically stacked columns to learn image features independently and in parallel. Each column consists of the same combination of convolutions, pooling layers and fully connected layers. An image gets a unique distortion/preprocessing before it is passed into a column. At the end all columns are averaged and a final output is produced.

### 2.2 Counting CNN

One of the proposals of the paper *Towards perspective-free object counting with deep learning* [4] is the Counting CNN. The model falls into counting by regression category, which is the category covered in this paper. In the training phase 800 RGB patches are randomly cropped with a fixed size from an input image and are fed to the model. Augmentation strategy is done by vertical flipping each patch making it 1,600 patches in total. They crop patches of size 72x72x3 and 18x18x1 patch is the direct output from their network. The prediction is done via dense patch extraction and the output patches are assembled together to generate the final ground truth estimation. Counting CNN was also trained on the TRANCOS dataset (see TRANCOS dataset) and reported an improved accuracy on this dataset [4].

Their architecture is rather small than large (6 convolutional layers in total). This makes it relatively fast to predict an output patch from the input one. If the number of training patches is smaller, training will be a lot faster with not so big accuracy impact.

### 2.3 Context-Aware Crowd Counting Network

Even if the Counting CNN tries to solve the perspective distortion by random patch extraction, its results indicate severe inaccuracies since the scale continuously varies across the whole image [5]. The Context-Aware Crowd Counting Network (CAN) adaptively encodes multi-level contextual information into features it produces. They perform so-called Spatial pyramid pooling [6]. Scale-aware features are computed by average-pooling subtraction from the VGG-16 [7] front-end. The front-end uses a method known as transfer learning. This means pre-trained weights are downloaded and used instead of initializing weights by some distribution, thus significantly reducing the training time. CAN model downloads pre-trained weights from the ImageNet dataset [8] and adjusts (trains) them to fit a particular dataset. In the training phase input image is divided into four regions of equal size (non-overlapping). One of these four regions is randomly chosen and passed to the net.

### 2.4 Pyramid Density-aware Attention Net for Accurate Crowd Counting

A very recent architecture (January 2020) to the time of writing my thesis [9]. PDANet continues to improve the accuracy of the CAN model.

This model also makes use of VGG-16 front-end for low-level feature extraction. Pyramid feature extraction with spatial and channel attentions are attached to the front-end to produce richer features. The model distinguishes between images with sparse and dense object instances. To distinguish between these two a classification module is used which then decides whether the image is either sparse or dense.

Within the decoder module there are two branches: dense and sparse. The model is trained in a such way that weights in the upper branch (dense) are adjusted for dense features and the lower branch (non-dense) is adjusted for sparse ones. At the end these branches are combined together to produce the final density estimation.

## 3. Proposed solution

The proposed architecture makes use of VGG-16 front-end for low-level features extraction. The CAN and PDANet use only first 10 layers of VGG-16 and so does the proposed solution. The architecture is depicted in the Fig. 1. The front-end is then connected

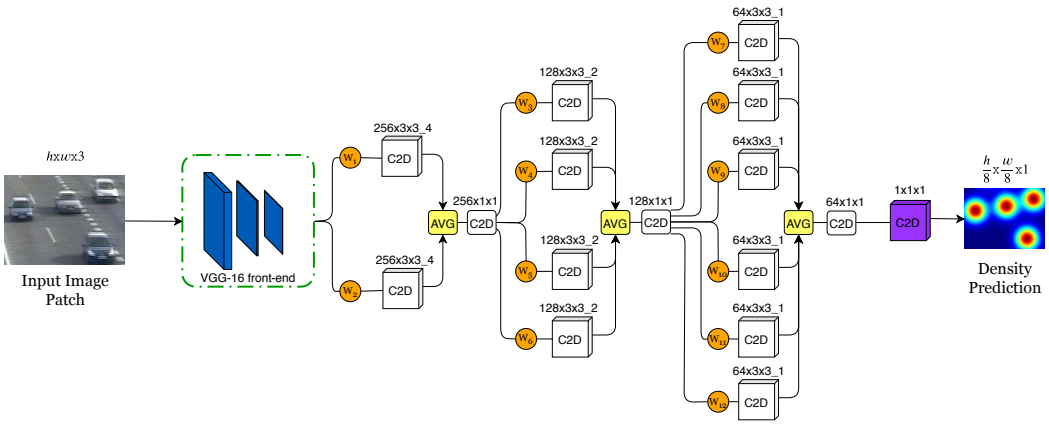



**Figure 1.** Proposed architecture. RGB patches are fed to the VGG-16 front-end network. Resulting low-level features are weighted (neurons $w_i$) and sent to dilated convolutions in parallel. Averaging followed by 1x1 convolving ends one such parallel layer. Dilated convolutions in parallel layers increase the receptive field of the network and help to understand the overall picture. The last parallel layer gets additional 1x1 convolving and produces density estimation. Input patch height and width dimensions are reduced by a factor of 8 due to two max-pooling layers in the VGG-16 front-end.

**Table 1.** Dilated convolutions setup

| Structure number | Filter depth | Dilatation rate |
|---|---|---|
| 1. | 256 | 4 |
| 2. | 128 | 2 |
| 3. | 64 | 1 |

to a tree-like structure, which is inspired by aforementioned Multi-column network.

One layer (structure) of the entire tree-like structure is composed as follows. Input to this layer gets weighted $n$-times (where $n$ is the number of parallel conv layers within the corresponding structure). Each weight $w_i$ represents one neuron, whose value is determined during the training phase. These weights use Softmax activation functions. Initially the value of each weight $w_i$ is set to an increasing multiple of $1/n$. So for example the last structure has the initial values $w_7 = 1/6$, $w_8 = 2/6$, $w_9 = 3/6$, $w_{10} = 4/6$, etc.

Each weight is connected to a dilated conv layer (C2D white box in the Fig. 1). All dilated convolutions use the same kernel size 3x3 but they differ in dilatation rate and filter depth. The setup of these conv layers is shown in the Table 1 (structure count starts from the front-end).

Dilated convolutions are then averaged. Final 1x1 convolving ends one such structure. All convolutions in each structure are followed by ReLU activation functions.

The final structure gets additional 1x1 convolution to produce a density estimation for a given input patch.

The weighting idea came from PDANet classifier, which is responsible for giving each of the two branches unique weights. Dilated convolutions increase the receptive field of the network and help to understand the overall picture instead of finer details [10].

I have implemented this architecture in Keras framework [11]. The model uses Mean Square Error (MSE) loss and Adam optimizer to compute weights.

**Training setup** Input RGB patch extraction is done via division of input image into four equal regions and randomly picking one of them. This patch gets randomly chosen combination of augmentation (zoom, horizontal flipping, gamma change, gaussian noise addition). The front-end has 2 max-pooling layers, which reduce input patch dimensions by a factor of 8. Therefore, the corresponding ground truth patch needs to be scaled down. Simply stretching or shrinking the patch could change the count (hence patch normalization is needed). In the *Towards perspective-free object counting with deep learning* paper [4] a normalization formula was proposed to solve such issue (see Equation 1).

$$\hat{D}^{P+}_{pred} = \frac{\sum_{\forall p} D^{P}{}_{pred}(p)}{\sum_{\forall p} \hat{D}^{P}_{pred}(p)} * \hat{D}^{P}_{pred} \qquad (1)$$

where:

- $\hat{D}^{P+}_{pred}$: is the rescaled density patch *with* a matching count
- $D^{P}{}_{pred}$: is the original (not rescaled) density patch
- $\hat{D}^{P}_{pred}$: is the rescaled density patch *without* a matching count
- $p$: is the pixel

Dense patch extraction is a chosen method to produce final density estimation (each patch is a quarter of an image). Since patches are overlapping the final

**Figure 2.** Sample images from specified datasets. **Top row:** TRANCOS **Bottom left:** PUCPR+ **Bottom right:** CARPK

density map must be normalized. Each position (pixel) in the final density map is an average of patches that cast a prediction in it.

## 4. Comparison with state of the art

The proposed architecture is trained and tested on the following datasets. Each dataset will be introduced and results achieved will be shown in tables. I will compare official state-of-the-art results with my model. All official state-of-the-art results are pulled from the respective dataset web page. Tables with comparisons are truncated to show only Top 5. Values in these tables depict error rates between expected and predicted vehicles counts measured on the corresponding test sets (the lower the value the more precise the prediction is).

### 4.1 TRANCOS dataset

A Spain car dataset [12]. TRANCOS is the acronym for TRaffic ANd COngestionS. As the name suggests the dataset focuses on traffic congestions. The dataset was made by capturing traffic via selected real surveillance cameras during three week period. The dataset contains more than 1,200 images and 46,700 annotated vehicles. These images are split in 3 sets as follows:

- Training - 403 images
- Validation - 420 images
- Test - 421 images

Since annotations are partial (not every vehicle is annotated), they used Region of Interest (ROI). Binary masks are provided to make visible only the annotated section of image (leaving non-ROI regions black). To train a model we are given a choice between 2 methods: either train solely on training set or to train on both training and validation sets (or as they call it - 'trainval' set).

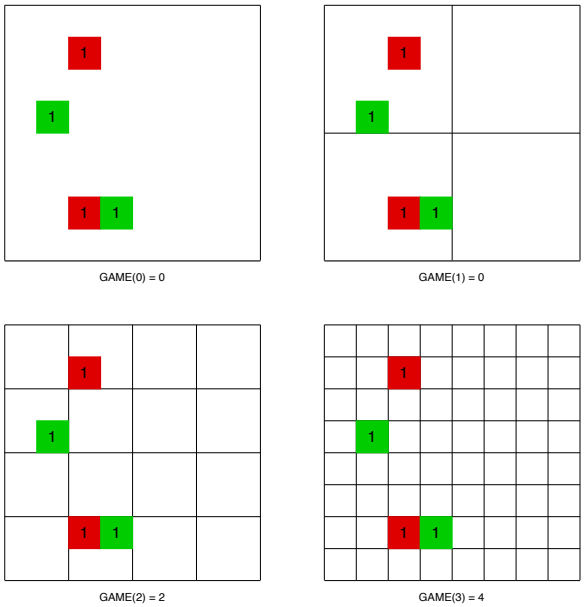



**Figure 3.** GAME(L) metric visualization

TRANCOS uses its own metric to measure network's accuracy – the $GAME(L)$ metric. $GAME(L)$ stands for Grid Average Mean Error and is required to be used when reporting results of one's model accuracy on this dataset. State-of-the-art results are shown in the Table 2. The $GAME(L)$ metric is formulated in the Equation 2.

$$GAME(L) = \frac{1}{N} \cdot \sum_{n=1}^{N} \left( \sum_{l=1}^{4^L} \left| e_n^l - gt_n^l \right| \right), \qquad (2)$$

where:

- $N$: is the total number of test images
- $L$: is the total levels number (max 4-regions exponent)
- $e_n^l$: is the estimated count within region $l$ of image $n$
- $gt_n^l$: is the ground truth count within region $l$ of image $n$

This metric takes into consideration vehicle localization error. The higher the $L$ number the more strict /precise localization error estimation is. I have made a toy example for demonstration purposes (see Fig. 3). In this figure there are red and green boxes with numbers within a grid. Red boxes represent expected vehicles within a region, whilst green ones represent vehicles prediction within a region. TRANCOS has matlab code for calculating this metric (publicly available along with the dataset on their web page [17]). I have rewritten this code from matlab to python to get the $GAME(L)$ evaluation of my model which is written in Keras [11].

**Table 2.** Evaluation of state-of-the-art methods on dataset TRANCOS. Lower number is better.

| Method | GAME(0) | GAME(1) | GAME(2) | GAME(3) |
|---|---|---|---|---|
| IbPRIA 2015 (HOG-2) [12] | 13.29 | 18.05 | 23.65 | 28.41 |
| IbPRIA 2015 (Fiaschi+RGB Norm+Filters) [12] | 17.68 | 19.97 | 23.54 | 25.84 |
| IbPRIA 2015 (Lempitsky+SIFT) [12] | 13.76 | 16.72 | 20.72 | 24.36 |
| Hydra CNN (ECCV2016) [4] | 10.99 | 13.75 | 16.69 | 19.32 |
| proposed model | **3.08** | **4.29** | **5.97** | **7.95** |

**Table 3.** Evaluation of state-of-the-art methods on dataset PUCPR+. Lower number is better.

| Method | MAE | RMSE |
|---|---|---|
| YOLO [13] | 156.00 | 200.42 |
| Faster R-CNN [14] | 111.40 | 149.35 |
| One-Look Regression [15] | 21.88 | 36.73 |
| Layout Proposal Network [16] | 22.76 | 34.46 |
| proposed model | **7.48** | **8.84** |

**Table 4.** Evaluation of state-of-the-art methods on dataset CARPK. Lower number is better.

| Method | MAE | RMSE |
|---|---|---|
| YOLO [13] | 48.89 | 57.55 |
| Faster R-CNN [14] | 47.45 | 57.39 |
| One-Look Regression [15] | 59.46 | 66.84 |
| Layout Proposal Network [16] | 23.80 | 36.79 |
| proposed model | **15.62** | **18.43** |

### 4.2 PUCPR+ dataset

This dataset is a subset of the PKLot dataset [18]. As per description the captures are taken at Pontifical Catholic University of Parana (PUCPR), located in Curitiba, Brazil. Images were taken from the 10th floor of the administration building of the PUCPR. These captures were taken during various weather conditions such as sunny, overcast and rainy.

Originally, the PUCPR dataset (note the missing '+' sign) is annotated only partly (100/331 parking spaces in a single image). Meng-Ru et al. [16] localized and completed missing annotations and called it PUCPR+. In total of only 125 images with nearly 17,000 cars, the 25 of them are meant for model testing. Such low number of images makes it a challenging task to train a network. State-of-the-art results are shown in the Table 3 (*MAE* and *RMSE* stand for Mean Absolute Error and Root Mean Square Error respectively). The dataset is available on the github [19].

### 4.3 CARPK dataset

Meng-Ru et al. proposed their own dataset and called it CARPK [16]. CARPK provides large-scale images taken from the drone point of view. The drone took images from four various parking lots. There are nearly 1,500 images with approximately 90,000 cars in total. Drone addresses the problem of a bias created by a fixed camera scene, where the point of view is constant. Images annotations are done by bounding box technique, where each vehicle gets upper left and bottom right (x, y) coordinates representing the bounding box. For the purpose of my regression approach this bounding box annotation is converted to dot annotation by taking a centre of the bounding box. State-of-the-art results are shown in the Table 4 (*MAE* and *RMSE* stand for Mean Absolute Error and Root Mean Square Error respectively). As well as the PUCPR+ dataset this one is also available on the github [19].

## 5. Conclusion

The paper presented a neural network architecture for counting vehicles instances in images. Proposed architecture is based on the VGG-16 front-end and multi-column structures setup, which experimentally improved current state-of-the-art results on datasets mentioned in the Section 4. To summarize, the contributions made in this paper are as follows.

Firstly, introduction of the new neural network architecture for counting vehicles, which is inspired by state-of-the-art architectures.

Secondly, evaluation of the architecture on vehicle datasets and comparison with current state-of-the-art architectures.

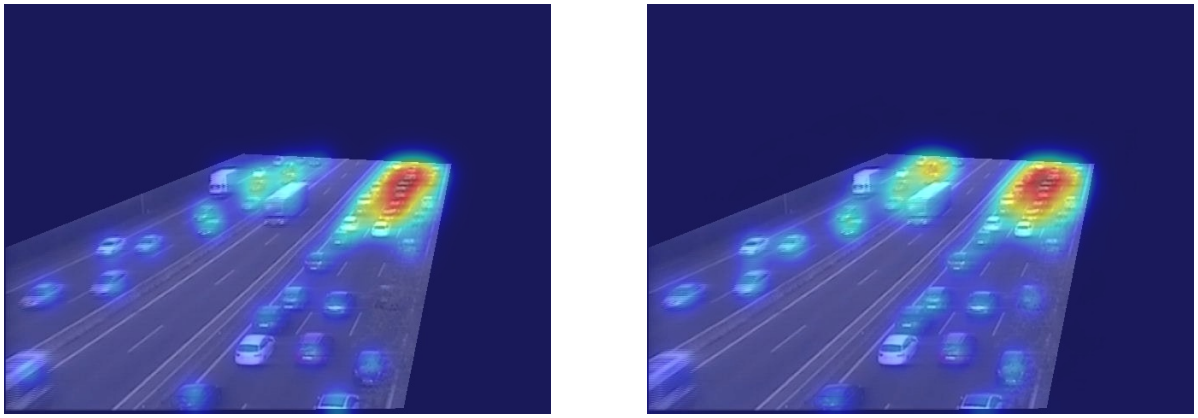
Lastly, further improvement of accuracy of state-of-the-art results on vehicle datasets. Some of the network predictions are shown in Figures: 4, 5, 6, 7. However, there is still space for improvements and possible further research in counting by regression approach.

## Acknowledgements

I would like to thank my supervisor Ing. Jakub Špaňhel for his patient guidance, encouragement and advice he has provided throughout writing this paper.

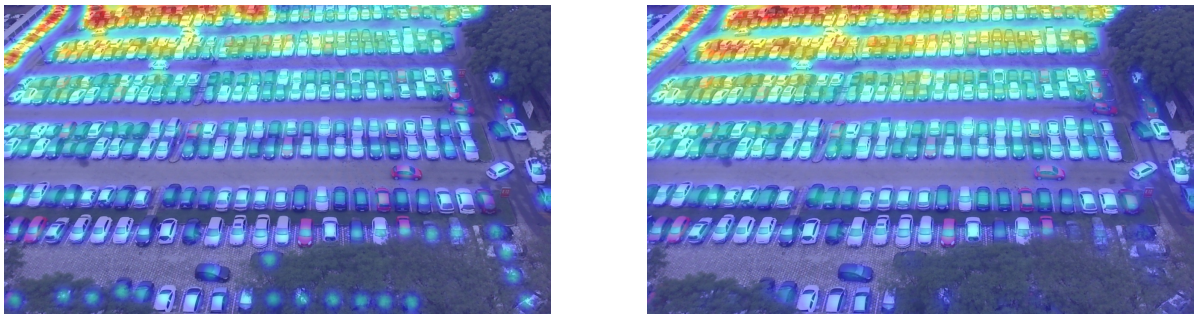
## References

[1] Lawrence A. Klein, Milton K. Mills, and David R. P. Gibson. *Traffic detector handbook*, volume 1, chapter 2. 3 edition, Oct 2006. Tech Report.



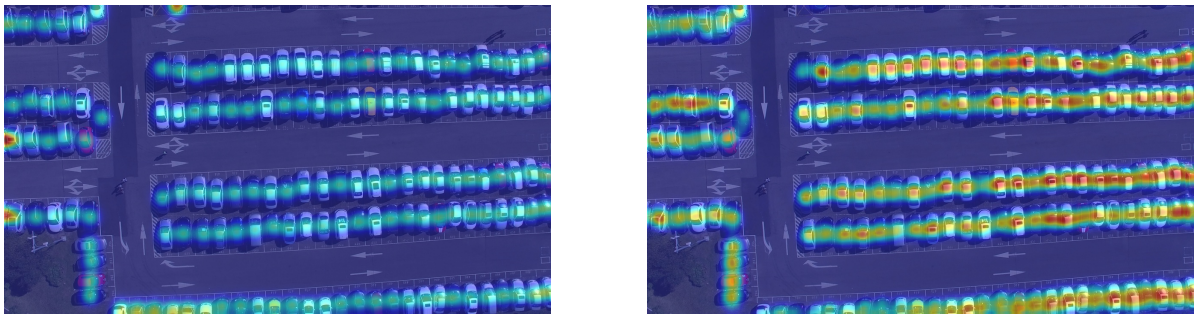
**(a)** Ground Truth: 44.03 **(b)** Prediction: 41.89

**Figure 4.** TRANCOS prediction



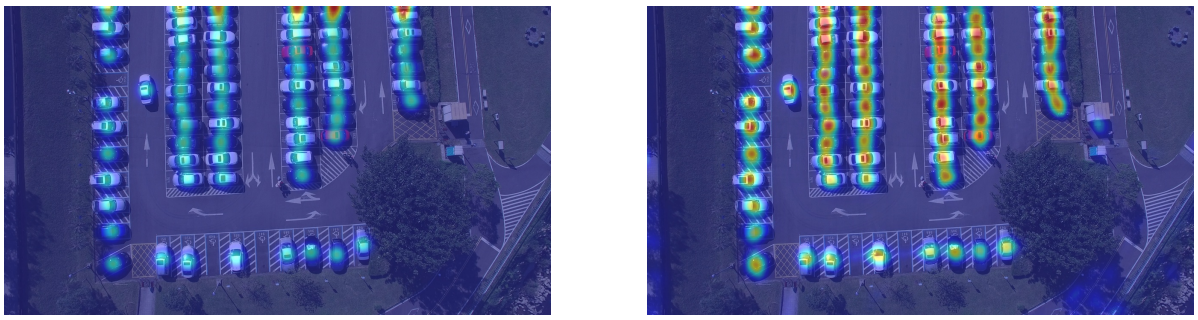
**(a)** Ground Truth: 328.77 **(b)** Prediction: 328.10

**Figure 5.** PUCPR+ prediction



**(a)** Ground Truth: 152.41 **(b)** Prediction: 143.76

**Figure 6.** CARPK prediction



**(a)** Ground Truth: 67.15 **(b)** Prediction: 63.94

**Figure 7.** CARPK prediction

[2] Victor Lempitsky and Andrew Zisserman. Learning to count objects in images. pages 1324–1332, 01 2010.

[3] Dan Cireşan, Ueli Meier, and Juergen Schmidhuber. Multi-column deep neural networks for image classification, 2012.

[4] Daniel Oñoro and Roberto López-Sastre. Towards perspective-free object counting with deep learning. volume 9911, 10 2016.

[5] Weizhe Liu, Mathieu Salzmann, and Pascal Fua. Context-aware crowd counting. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(9):1904–1916, 2015.

[7] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[8] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, June 2009.

[9] Saeed Amirgholipour, Xiangjian He, Wenjing Jia, Dadong Wang, and Lei Liu. Pdanet: Pyramid density-aware attention net for accurate crowd counting, 2020.

[10] mc.ai. *10. Introduction to Deep Learning with Computer Vision - Types of Convolutions & Atrous Convolutions*. https://mc.ai/10-introduction-to-deep-learning-with-computer-%vision-types-of-convolutions-atrous-convolutions-2. [Online; visited 12.03.2020].

[11] Keras. *Keras Documentation - Home*. https://keras.io/. [Online; visited 31.01.2020].

[12] Ricardo Guerrero-Gómez-Olmedo, Beatriz Torre-Jiménez, Roberto López-Sastre, Saturnino Maldonado-Bascón, and Daniel Oñoro. Extremely overlapping vehicle counting. 06 2015.

[13] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[14] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 91–99. Curran Associates, Inc., 2015.

[15] T. Nathan Mundhenk, Goran Konjevod, Wesam A. Sakla, and Kofi Boakye. A large contextual dataset for classification, detection and counting of cars with deep learning. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 785–800, Cham, 2016. Springer International Publishing.

[16] Meng-Ru Hsieh, Yen-Liang Lin, and Winston H. Hsu. Drone-based object counting by spatially regularized regional proposal networks. In *The IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2017.

[17] TRANCOS. *TRANCOS dataset - Download*. http://agamenon.tsc.uah.es/Personales/rlopez/data/trancos/TRANCOS_v3.tar.gz. [Online; visited 31.01.2020].

[18] Paulo Almeida, Luiz Soares de Oliveira, Alceu Jr, Eunelson Jr, and Alessandro Koerich. Pklot - a robust dataset for parking lot classification. *Expert Systems with Applications*, 42, 02 2015.

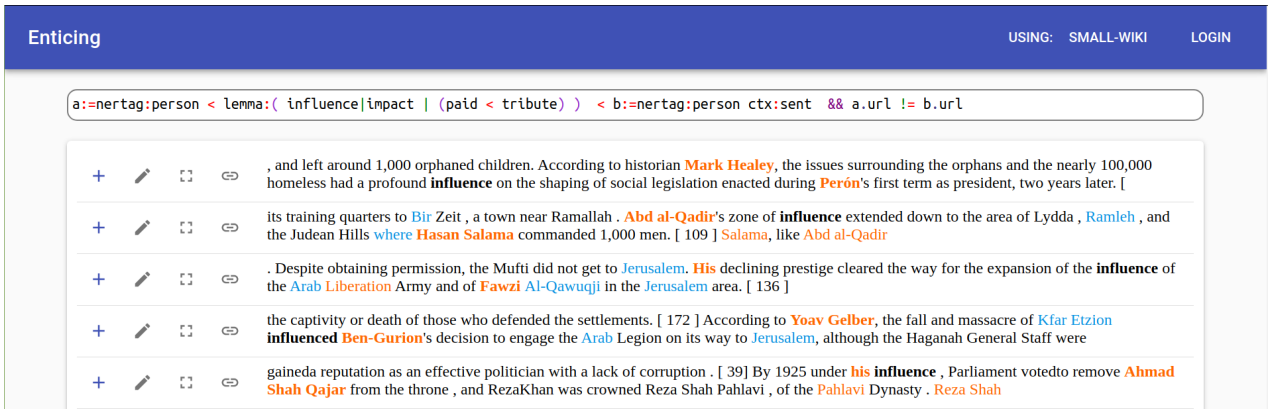
[19] Meng-Ru Hsieh, Yen-Liang Lin, and Winston H. Hsu. *CARPK, PUCPR+ datasets - Download*. https://lafi.github.io/LPN/. [Online; visited 31.01.2020].

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

# Enticing – Semantic Search Engine

David Kozák*




**Abstract**
The topic of this paper is semantic searching over big textual data. It describes the design and implementation of a search engine called Enticing that queries semantically enhanced documents efficiently and has a user friendly interface for working with the results. First, state of the art solutions along with their strengths and shortcomings are analyzed. Then a design for new search engine is presented along with a specialized query language EQL. The system consists of components for indexing and searching the documents, management server, compiler for the query language and two clients, web based and command line. The engine has been successfully designed, developed and deployed and is available via Internet. As a result of that, the possibility to use semantic searching is available to a wide audience.

**Keywords:** search engine — semantic enhancement — MG4J — compiler — indexation — searching — annotation — big data

**Supplementary Material:** Web Interface

*xkozak15@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

## 1. Introduction

The topic of this paper is semantic searching over big textual data. The Knowledge Technology Research Group (KNOT)[1] at FIT BUT has a Natural language processing (NLP) pipeline which can analyse documents written in natural languages and add additional meta information to them. Such information can be *syntactic*, such as lemma of the word or its position within sentences and paragraphs, or *semantic*, such as entities like people and places. The output of this pipeline is a big volume of textual data. It is already a great piece of work on it's own, but without the ability to query these semantically enhanced documents, their usage is limited. The goal of this project is to design and develop a search engine that would query the documents efficiently while allowing to use all the meta information in the queries. The engine will be called Enticing.

A couple of search engines with support for semantic search such as Mimir or Sketch Engine have been implemented before. One such engine has even been created internally within KNOT[2]. However, none of

[1] https://www.fit.vut.cz/research/group/knot/

[2] http://knot.fit.vutbr.cz/projects.html

those matched the requirements for the new engine. Its goal is to allow the user to query all the meta information including entities with attributes and relationships between them. At the same time, the queries should be easy to read even for non-programmers. The engines mentioned above either did not provide support for entities with attributes or were way too complex for our use case. The previous engine was an attempt to create such a system, but unfortunately, it was not written in a maintainable way, which made any further extensions or bugfixes very time consuming. That's why the decision was made to create a new engine and design it with stability and maintainability in mind.

In order to query the semantic metadata, a special query language has to be used. As a part of this project, such query language called EQL has been designed and its compiler was integrated into the search engine infrastructure.

The text is structured as follows. Basic definitions and techniques used inside search engines are defined in section 2. Semantic enhancement and semantic search are introduced in 3. Section 4 describes state of the art semantic search engines along with their strengths and weaknesses. The design of Enticing is presented in 5. Section 6 describes Enticing Query Language. Section 7 discusses testing and evaluation of the platform. In the end, a conclusion is given.

## 2. Indexing and searching inside search engines

This section describes the problem of indexing and searching in the context of search engines. First subsection defines basic terms and typical techniques used within search engines. Afterwards a search engine called MG4J is presented, as it is used internally within Enticing.

### Basic definitions

Since this project is a follow-up work of [1, 2], the terminology will be mostly identical as it was defined in [1] with some modifications and extensions.

**Index** The Oxford dictionary defines it as an alphabetical list of names, subjects, etc. with reference to the pages on which they are mentioned[3]. In [1], they defined index as a structure allowing a faster access to a certain piece of information without the need to process all the data. This definition suits our needs perfectly, therefore we will follow it.

[3] https://www.lexico.com/en/definition/index

**Inverted Index** An inverted index over a collection of documents contains, for each term of the collection, the set of documents in which the term appears and additional information such as the number of occurrences of the term within each document, and possibly their positions [3]. This data structure is used inside search engines to efficiently determine which documents contain the specified words.

**Query expansion** Modern web search engines rely on query expansion, an automatic or semi automatic mechanism that aims at rewriting the user intent (i.e., a set of keywords, maybe with additional context such as geographical location, past search history, etc.) as a structured query built upon a number of operators [4].

**Tree based evaluation of the query** Search queries can be efficiently represented as trees. The leaves represent keywords from the query and the intermediary nodes represent operators. The searching algorithm can then proceed in the bottom up way as follows. First, all the leaves of the tree are evaluated and the results are stored within them. Then their parents are evaluated, combining results from their children. The execution proceeds all the way to the root, which represents the whole query [1].

**Semantic models of searching** The semantics of the search query is given by the semantic model. The simplest one is the boolean model, where only conjuctions, disjunctions, negations and keywords are allowed. Unfortunately, this model does not provide any information regarding the fact how the document was matched by the query. MG4J uses a semantic model called Minimal Interval Semantics. It uses intervals of natural numbers that are incomparable towards inclusion. Each interval is a witness of the satisfiability of the query, and defines a region of the document that satisfies it [4].

**Snippet** If a query matched a document, the result has to be presented to the user. One of the most common forms of presenting search results are snippets. A snippet is a part of a document that matched given query, possibly extended with some additional information given by the searching algorithm.

### MG4J – Managing Gigabytes for Java

This subsection describes MG4J, a free full-text search engine for large documents written in Java [5]. It is developed under the GNU Lesser General Public

License[4] at the University degli Studi di Milano[5].

MG4J has a query language with very expressive set of operators allowing to build complex queries. These operators are implemented using new very efficient search algorithms [6]. It supports searching over multiple indexes and combining the results. On top of that, MG4J is open source, so it is possible to dive into the source code when the answers cannot be found in the documentation. Unfortunately, it has no support for entities with attributes and relationships between them. Nevertheless, the aforementioned properties make it a very suitable backend for our semantic search engine.

## 3. Semantic enhancement of natural languages

This section explains the topic of semantic enhancement. First it provides basic definitions. Then it describes how semantically enhanced documents are created within KNOT.

### Basic definitions

The key definition in semantic enhancement is **Semantic annotation**. Semantic annotations are metadata assigned to other data in order to increase their context and semantics [1]. These annotations are usually derived from unstructured content using Natural language processing and afterwards they are encoded in a structured format suitable for semantic search [7].

**Semantic search** over documents aims to find information that is not based just on the presence of words, but also on their meaning. It is gradually establishing itself as the next generation search paradigm, which can satisfy better a wider range of information needs, as compared to traditional full-text search. In the case of semantic search, what is being indexed is typically a combination of words, formal knowledge typically expressed in an ontology, and semantic annotations mentioning ontological concepts in the text [7].

### Corpora processing tools

Any search engine would be useless without large enough amount of data for searching. This subsection briefly introduces the corpora processing pipeline[6] which is used within KNOT to create semantically enhanced documents. These documents are then used in Enticing. The input of this pipeline are html pages and the output are tsv files, where each line represents a single word in the document along with its metadata. The pipeline consists of the following stages.

1. **Verticalization** – HTML pages are parsed into tsv files.
2. **Deduplication** – Filters out duplicit pages and paragraphs within them.
3. **Tagging** – Idenfies parts of speech.
4. **Parsing** – Syntax analysis.
5. **SEC** – Semantic enhancement.

## 4. State of the art solutions

This section covers three state of the art search engines which support semantic search to a various extent along with their strengths and shortcomings.

### Mimir

Mimir is a semantic search engine developed at the The University of Sheffield. It was developed as a part of the Gate infrastructure for language engineering[7]. Using index federation and cloud-based deployment, it can scale up to 150 millions documents. It also supports hybrid queries that arbitrarily mix full-text, structural, linguistic and semantic constraints [7]. It internally combines different indexing technologies. The full-text search is done using MG4J and a triple store queried using SPARQL is then used for accessing Linked Open Data resources.

Mimir is a very powerful engine, even too powerful for our use case. Most of its functionality is not necessary, therefore using it would be too heavyweight.

### Sketch Engine

Sketch Engine is a tool for analyzing how languages work. It analyses large text corpuses in order to find out what is typical and what is rare for a given language. It houses more than 500 corpora in more than 90 languages [8] and it supports searching in them. It is an interesting piece of work that can be used for studying languages, but it was not designed to be a publicly available search engine.

### Previous system at FIT BUT

One semantic search engine was developed within KNOT. It's original author was Jan Kouril[8] and it was extended by S. Panov in [1] and K. Gresova in [2]. Its architecture was the following. There were two main components, Webserver and IndexDeamon. IndexDeamon was able to index documents and query them. Webserver provided user interface for the system.

[4] https://www.gnu.org/licenses/lgpl-3.0.en.html
[5] http://www.unimi.it/
[6] http://knot.fit.vutbr.cz/corpproc/corpproc_en.html
[7] https://gate.ac.uk/
[8] https://www.fit.vut.cz/person/ikouril/

For querying the documents, a special package Query along with a query language mg4j-eql was developed in [1]. The Query package was created so that it could be used in various clients. It was later integrated into the old webserver in [2]. K. Gresova also made several changes to the webserver to make it more flexible. However, the original code was not written in a maintainable way and, as it typically happens with software products, it's quality got worse and worse over time. And because of the maintainability issues, it was very hard to extend its functionality or provide bug fixes.

## 5. Design of the platform

This section covers the design of Enticing. It is a distributed system consisting of several components. The components and the relationships between them are visualized in the Figure 1.

The system is designed to handle very large text corpuses. The components for indexing and searching are meant to be run in parallel on multiple machines, each of them working with a piece of the corpus. Since it is quite easy to split the set of documents into parts of similar length, this design allows to scale up or down very easily simply by adding or removing servers.

There is quite a lot of metadata available, but users sometimes need just a part of it. The system allows users to customize what metadata should be returned, which can reduce the size of the result only to the information that is really wanted.

Each of the components of the system will now be described in more details.

### Webserver

Webserver is the first and only component common users will interact with. It is also the only one that is meant to be publicly accessible. All requests should pass through it before being forwarded further into the system. It exposes an API that can be used by any third-party service to submit a search query. It is bundled with a single page JavaScript application that serves as a GUI of the system.

Apart from being the entry point for queries, Webserver's responsibilities also include user management and search settings management.

### ConsoleClient

Sometimes, especially for research purposes and for testing, it is useful to submit a query and collect all the results into a file or to execute a group of queries in a batch. ConsoleClient is a component that handles this use case.

### IndexServer

IndexServer is a component that handles a set of documents and exposes an API for querying them. This API should be accessible only internally, that is from the Webserver or the ConsoleClient, it is not meant to be directly reachable for the users of the system.

Internally, the set of documents is divided into collections. Each of these collections is handled concurrently. The documents are queried using MG4J.

This component is meant to be deployed multiple times on multiple machines to handle bigger text corpuses.

### IndexBuilder

This component is a command-line tool responsible for preprocessing documents using MG4J and creating indexes that are later used by IndexServers. The process of indexing is both time and resource consuming, that's why it is handled separately and not directly from the IndexServers.

## 6. EQL – Enticing Query Language

EQL is a language which can be used to query semantically enhanced documents on the Enticing platform. The queries can be as simple as a few words, but also very complex, containing logical operators, subqueries over multiple indexes or constraints further limiting the results.

### Operators

The operators can be divided into four categories.

#### Basic operators

- **Implicit and** – `A B` – If no operator is specified, **and** is chosen implicitly. That is all mentioned words have to be in the document, in any order.
- **Order** – `A < B` – A should appear before B, but they do **not** have to be next to each other.
- **Sequence** – `"A B C"` – A, B and C have to appear in this order next to each other.
- **And** – `A & B` – Both A and B have to be in the document, in any order.
- **Or** – `A | B` – At least one of A, B has to be in the document.
- **Not** – `!B` – B should not be in the document.
- **Parenthesis** – `(A | B) & C` – Parenthesis can be used to build more complex logic expressions.
- **Proximity** – `A B ~ 5` – A and B should appear at most 5 positions apart.

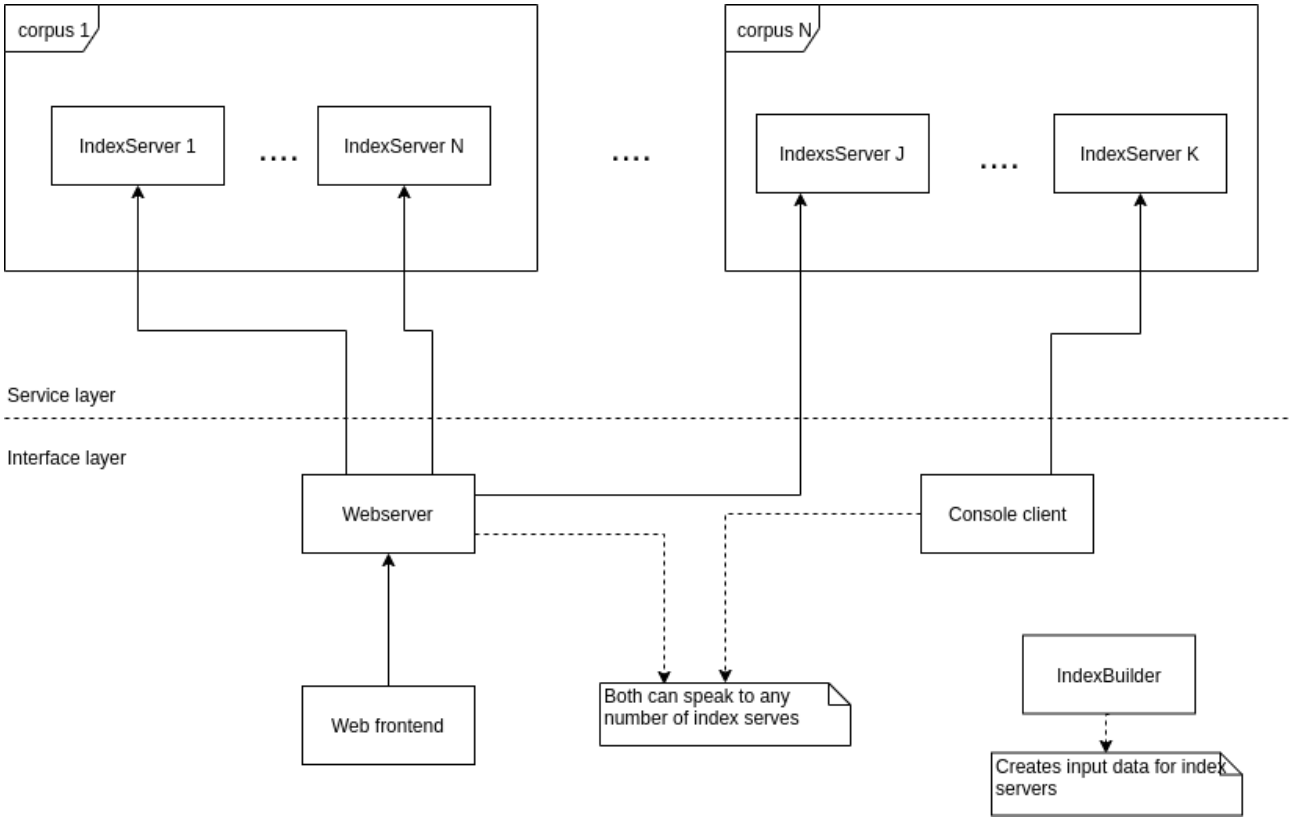



**Figure 1. Components** Components of the system

### Index operators

To work with meta information, one has to specify index for querying. That can be done using the index operator.

- **Index** – `index:A`

This query will look for documents, where value A is present at given index. For example, `lemma:work` will match any document in which any word, whose lemma is work, appears, e.g. works, working, worked, etc.

It is also possible to ask more complex queries such as `index:A|B|C`. Apart from another index operator or entity operator, a query of any form can be used.

- **Attribute** – `person.name:A`

When working with entities, it is possible to query their attributes as well. The query above will return documents with people whose name is A.

- **Align** – `index1:A ^ index2:B`

The align operator allows to express multiple requirements over one word. For example, it is possible to look for a noun, whose lemma is do. This query can be written as `pos:noun ^ lemma:do`.

### Context constraints

The default context for searching is the whole document. For more granular queries, it is possible to add the following limitations to the end of the query.

- **Paragraph** – `ctx:par` – Limits the query to one paragraph only.
- **Sentence** – `ctx:sent` – Limits the query to one sentence only.

### Global constraints

Sometimes it is necessary to specify a relationship between multiple entities that can't be expressed using the previous operators. One example might be searching for documents talking about two artists influencing each other. A simple query for that would be the following.

```
nertag:artist < lemma:influence
  < nertag:artist
```

But there is a problem with this query. It might return irrelevant snippets, because there is no requirement that the two artists should be different. This is where the global constraints come into play. The global constraint is a predicate which is separated from the query by symbols `&&`. The constraint consists of one or more equalities and inequalities connected using logical operators `and, or, not` and parenthesis, if necessary.

In order to use the global constraints, relevant parts of the query have to be identified first.

- **Assignment** – `x:=A` – Assign an identifier to a certain part of the query.

Afterwards, it is possible to write queries with global constraints.

```
a:=nertag:artist < lemma:influence
  < b:=nertag:artist
  && a.nerid != b.nerid
```

### Searching algorithms

As mentioned earlier, MG4J has no support for entities with attributes and relationships between them. It also does not have support for identifiers. Therefore, the query has to be rewritten before being passed into MG4J and the results have to be post-processed. This step computes which parts of the documents were matched by which nodes in the Abstract Syntax Tree (AST) of the query. Using this information, global constraints can be evaluated. It can also be used to provide very precise highlighting. Unfortunately the computation is both time and space consuming. It essentially requires us to re-implement the searching functionality of MG4J on our own, which is not an easy task. On the other hand, it gives us a chance to create our own semantics for the query, which can be different from the underlying technology.

Minimal-Interval Semantics used by MG4J can significantly reduce the number of returned snippets and it allows them to use very efficient searching algorithms [4]. However, it has drawbacks. Imagine a query about a place and a person. Now lets say that there is a document containing three different people and three places and the intervals between them are overlapping. MG4J would return only one interval connecting one person with one place. In Enticing, we prefer to return all three combinations that occur, even though they overlap. MG4J does not support that, but it will at least return the document for us and we can then perform the post-processing generating all possible combinations. The con of our solution is that the number of results can grow exponentially. To cope with that, we decided to limit the number of snippets per document.

## 7. Testing and evaluation

This section covers testing and evaluation of the platform. Since the project is very big, multiple different types of tests were used together to ensure its quality. Each module has its own *unit tests* verifying its functionality. Different modules are combined together into components and *integration tests* verify the communication between them.

A significant part of testing is centered around EQL and its searching functionality. They form the very core of the search engine and therefore their correctness is of the highest importance. EQL compiler has a suite of tests verifying that the translation from EQL to MG4J works as expected. Another suite of EQL tests was created to ensure that semantic errors are discovered before the query is executed. More *functional tests* were added to verify that given a query and a group of test documents, correct results are provided. These tests are done on multiple layers, from the core EQL algorithms through IndexServers all the way to the Webserver, to ensure that the results are not malformed as they are post-processed.

### Performance

Apart from the correctness itself, performance is very important. Users expect search results to be delivered fast and therefore the whole system has to be optimized to satisfy that. In order to perform any optimizations, measurements have to be taken first, to make sure that the optimizations work as expected.

The current measurements at the time of writing this paper are presented in the table 2. Please note that the system is still being developed, so the results presented here might not be aligned with the performance you are experiencing when using Enticing. We are currently working on optimizing the search performance, so hopefully the system will be even faster by the time you experiment with it. We measured the duration of querying one IndexServer, as it is the most important use case to optimize. Wanted amount of snippets was 20. Tested IndexServer instance was deployed on KNOT server *knot01.fit.vutbr.cz*. This server has Intel Xeon E5-2630 2,3 GHz processor with 15MB cache and 6 cores. Total ram size is 65536 MB. IndexServer instance was handling 10 mg4j files, which together had 2.9 GB and contained 24371 documents. We created a list of queries and then submitted them 100 times, measuring the execution time of each query. We then computed the average value, the deviation, min and max of the execution time for each query. We started with simple single word queries, then combined them together, also adding restrictions later on. Contrary to what we expected, more complex queries are not always significantly slower. The number of occurences of searched terms and their locations play a significant role as well. For example, there are only 1285 matches of the word water, but there are more than 10000 entities of type person. Therefore less documents have to be iterated when providing 20

results. You can also notice a significant slowdown in the query *water nertag:person nertag:location ctx:sent*. This happens because the context restrictions operators are currently not very optimized. They will be our first target in future optimizations.

## 8. Conclusions

The main topic of this paper was a search engine for querying semantically enhanced documents. The Knowledge Technology Research Group (KNOT)[9] at FIT BUT has a Natural language processing (NLP) pipeline which can analyse documents written in natural languages and add additional meta information to them. Such information can be *syntactic*, such as lemma of the word or their position within sentences and paragraphs, or *semantic*, such as entities like people and places. The search engine should query the documents efficiently while allowing to use all the meta information in the queries.

The first section focused on the problem of indexing and searching inside search engines. It also described MG4J[10], a search engine that is used internally in the resulting infrastructure. The following section focused on semantic enhancement of natural languages. The corpora processing pipeline[11] used within KNOT to create semantically enhanced documents was described. Afterwards the design of the new search engine called Enticing was presented. It is a distributed system, with each of its components consisting of several modules with well-defined interfaces. Each component is designed to be deployed as a separate process on its own server. Therefore the system should be able handle most of exceptions in its components without shutting down totally and at least partial results should be presented to the user whenever possible.

In order to query the semantic metadata, new query language called EQL (Enticing Query Language) was designed. This language is powerful enough to query all the entities inside semantically enhanced documents but also simple to understand, so that users from other domains can use it as well.

The system was built incrementally. First, basic prototypes of all components were developed to better understand the domain and gather more precise requirements. Afterwards, the architecture was carefully designed and all components were extended accordingly. Some parts were rewritten multiple times until their quality was sufficient. After the implementation, the system was tested and deployed on KNOT servers. The core was then further extended by adding a monitoring infrastructure and maintenance system. A follow-up work can extend the platform by adding new types of IndexServers, for example backed by a neural network, or by adding a native mobile client.

[9] https://www.fit.vut.cz/research/group/knot/

[10] http://mg4j.di.unimi.it/

[11] http://knot.fit.vutbr.cz/corpproc/corpproc_en.html

## Acknowledgements

I'd like to thank my supervisor, Ing. Jaroslav Dytrych, Ph.D., and Doc. RNDr. Pavel Smrž, Ph.D for their professional help and guidance. I also want to thank Ing. Jan Doležal for his guidance and support in the design process. Last but not least, I'd like to thank my parents for their neverending support.

## References

[1] Panov Sergey. Indexing and searching semantically annotated texts. Bachelor thesis, Brno University of Technology, Faculty of Information Technology, Brno, 2017.

[2] Grešová Katarína. Searching semantically annotated texts. Bachelor thesis, Brno University of Technology, Faculty of Information Technology, Brno, 2018.

[3] Sebastiano Vigna. Quasi-succinct indices. In *Proceedings of the 6th ACM International Conference on Web Search and Data Mining*, pages 83–92, Rome, Italy, 2013. ACM.

[4] Paolo Boldi and Sebastiano Vigna. Efficient optimally lazy algorithms for minimal-interval semantics. In *Theoretical Computer Science*, pages 8–25, Dipartimento di Informatica, Università degli Studi di Milano, Italy, 2016.

[5] Paolo Boldi and Sebastiano Vigna. Mg4j: high-performance text indexing for java. online, 2005.

[6] Paolo Boldi and Sebastiano Vigna. *Mg4j (big) The Manual*. Available at http://mg4j.di.unimi.it/man-big/manual.pdf.

[7] V. Tablan, K. Bontcheva, I. Roberts, and H. Cunningham. Mímir: an open-source semantic search framework for interactive information seeking and discovery. *Journal of Web Semantics*, 2014.

[8] Adam Kilgarriff, Pavel Rychlý, Miloš Jakubíček, et al. Sketchegine. online.

**Table 2.** Results of performance tests

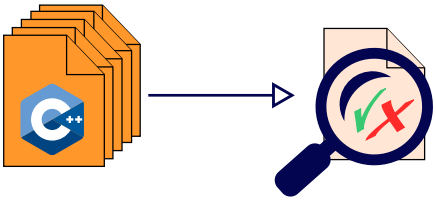
| EQL Query | Average[ms] | Deviation[ms] | min[ms] | max[ms] |
|---|---|---|---|---|
| water | 711.4 | 65843.12 | 337 | 1823 |
| nertag:person | 155.4 | 2263.76 | 104 | 421 |
| nertag:location | 177.73 | 24321.48 | 93 | 835 |
| water nertag:person | 503.0 | 35445.24 | 355 | 1259 |
| water nertag:person ctx:sent | 423.15 | 8891.11 | 321 | 771 |
| nertag:person nertag:location | 130.24 | 1919.32 | 96 | 433 |
| nertag:person nertag:location ctx:sent | 287.09 | 2490.72 | 220 | 463 |
| water nertag:person nertag:location | 453.82 | 14603.47 | 331 | 1056 |
| water nertag:person nertag:location ctx:sent | 4317.23 | 16767.16 | 4071 | 4828 |
| nertag:person nertag:person | 119.28 | 1071.40 | 96 | 403 |
| nertag:person nertag:person ctx:sent | 124.48 | 828.09 | 100 | 330 |
| a:=nertag:person b:=nertag:person | 117.31 | 699.074 | 97 | 287 |
| a:=nertag:person b:=nertag:person ctx:sent | 130.61 | 667.64 | 104 | 247 |
| a:=nertag:person b:=nertag:person && a.url != b.url | 126.33 | 801.48 | 98 | 263 |
| a:=nertag:person b:=nertag:person ctx:sent && a.url != b.url | 353.91 | 10034.46 | 250 | 801 |

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

# Detekce paralelních chyb ve víceprocesových programech

Monika Mužikovská*



**Abstrakt**
Dynamická analýza se s úspěchem využívá pro detekci chyb ve vícevláknových programech. Algoritmy, které byly za tímto účelem navrženy, jsou ale často využitelné i pro víceprocesové programy. Žádný ze známých nástrojů pro dynamickou analýzu ale monitorování procesů nepodporuje. Cílem této práce bylo rozšířit nástroj ANaConDA o analýzu a monitorování víceprocesových programů. Výsledkem je implementace rozšíření, které za vývojáře analyzátorů řeší problémy spojené s oddělenými adresovými prostory a synchronizací pomocí semaforů. Rozšíření bylo využito pro úpravu analyzátoru AtomRace pro detekci časově závislých chyb nad daty ve víceprocesových programech a použito na experimenty se studentskými projekty z předmětu Operační systémy. Výsledky experimentů ukázaly, že se nástroj ANaConDA může stát vítaným pomocníkem při implementaci nejen víceprocesových projektů.

**Klíčová slova:** ANaConDA – Dynamická analýza – Paralelní chyby – Víceprocesové programy

**Přiložené materiály:** N/A

*xmuzik05@stud.fit.vutbr.cz, *Fakulta informačních technologií, Vysoké učení technické v Brně*

## 1. Úvod

Vícevláknové a víceprocesové programy jsou náchylné na specifické chyby s nedeterministickým projevem, a proto se k jejich testování využívají jiné přístupy než pro typické sekvenční programy. Mezi úspěšné metody pro odhalování paralelních chyb patří dynamická analýza, která monitoruje běh programu, sleduje důležité události a ze získaných informací vyvozuje závěry o případných problémech. Jelikož je monitorování běhu programu implementačně velmi náročné, vznikly nástroje, které tuto funkcionalitu poskytují (např. ConTest [1], RoadRunner [2], můžeme zde zařadit i Valgrind [3] a samozřejmě nástroj ANaConDA [4]). Detekce chyb potom probíhá v analyzátorech, které často implementují speciální algoritmus zaměřený na konkrétní typy chyb. Aplikační oblast analyzátoru je tedy omezena tím, jaké typy programů jsou nástroje schopny monitorovat. V současné době se bohužel u všech existujících nástrojů jedná pouze o vícevláknové programy a paralelismus na úrovni procesů není podporován. Přitom navržené algoritmy často mezi vlákny a procesy nerozlišují a je možné je použít pro detekci chyb v obou typech programů. Pro plné využití potenciálu těchto algoritmů se tedy nabízí rozšíření již existujícího nástroje o podporu monitorování procesů.

Přirozenou volbou byl nástroj ANaConDA, který je aktivně vyvíjen zde na fakultě. ANaConDA slouží k dynamické analýze C/C++ programů a poskytuje sadu analyzátorů pro detekci časově závislých chyb nad daty (z angl. *data race*), uváznutí či porušení atomicity. Kromě monitorování běhu také nabízí sadu podpůrných funkcí a struktur pro snadnější imple-

mentaci nových analyzátorů. Podrobnosti jsou popsány v sekci 2.

### 1.1 Řešené problémy a přínos práce

Tento článek dále shrnuje, v čem se vlákna a procesy liší, a tím pádem se musí lišit i implementace analyzátorů. Hlavní rozdíl spočívá v tom, že vlákna mají společný adresový prostor, a tedy sdílí veškerá data. Všechny dynamické analyzátory implementované v nástroji ANaConDA se od tohoto faktu odrážejí. Předpokládají, že všechny identifikátory a adresy jsou sdílené mezi všemi vlákny, takže např. využívají logickou adresu jako jednoznačný identifikátor. Procesy ale mají oddělené adresové prostory a tento předpoklad je pro ně tedy chybný. Při monitorování programu tak analyzátory musí brát v potaz to, že stejná logická adresa v různých procesech může příslušet jiné fyzické adrese (ale nemusí) a ne všechna data jsou mezi procesy sdílená, a tedy ne každý přístup do paměti může způsobit chybu. Aby bylo možné rozhodovat, zda je přístup do paměti potenciálně nebezpečný či nikoli, je třeba monitorovat meziprocesovou komunikaci prostřednictvím sdílené paměti, což dosud nebylo potřeba. Také samotné analyzátory jsou paralelní programy a využívají stejnou formu paralelismu jako monitorovaný program. Současné implementace předpokládají vícevláknovost a přizpůsobují tomu mechanismy sdílení dat a synchronizaci. U víceprocesových programů je samozřejmě třeba volit jiné přístupy, a proto se i implementace analyzátoru musí přizpůsobit. Stávající analyzátory tak nelze beze změny na monitorování procesů použít a musí být vytvořeny analyzátory nové. Sice implementující stejný algoritmus pro detekci chyb, ale zohledňující problémy, ke kterým u procesů dochází. Aby úprava implementace analyzátorů za účelem monitorování procesů byla co nejmenší, byla většina problémů vyřešena v rámci této práce a implementována do jádra nástroje ANaConDA. Konkrétně tato práce přinesla:

1. API pro komunikaci procesů analyzátoru.
2. Řešení problému s identifikátory a logickými adresami.
3. Monitorování synchronizace pomocí obecných semaforů (dosud bylo poskytováno pouze pro binární zámky).
4. Rozšíření *happens-before* relace pro extrapolaci běhu programu na tyto obecné semafory.

Detaily jsou popsány v sekci 3.

Podpora pro monitorování procesů byla využita k implementaci analyzátoru AtomRace [5] pro detekci časově závislé chyby nad daty ve víceprocesových programech. S analyzátorem byly provedeny experimenty nad studentskými projekty z předmětu Operační systémy, které ukázaly, že i v projektech s plným bodovým hodnocením lze najít paralelní chybu.

## 2. Monitorování vláken

Tato sekce je věnována stručnému popisu nástroje ANaConDA a jeho funkci při dynamické analýze vícevláknových programů. Dále je uveden algoritmus AtomRace pro detekci časově závislé chyby nad daty, který je v nástroji ANaConDA implementován jako jeden z poskytovaných analyzátorů. Na závěr sekce je popsána tzv. *happens-before* relace, kterou využívají komplikovanější analyzátory pro extrapolaci běhu programu.

### 2.1 Nástroj ANaConDA

ANaConDA[1] (Adaptable Native-code Concurrency-focused Dynamic Analysis) je open-source nástroj pro dynamickou analýzu vícevláknových C/C++ programů na binární úrovni [4][6]. Skládá se ze tří hlavních částí: nástroje Pin, jádra nástroje a analyzátorů. Pin[2] [7] provádí dynamickou instrumentaci programu a monitoruje jeho běh. Při důležitých událostech informuje jádro prostředí ANaConDA, které nízkoúrovňové informace předzpracuje a v uživatelsky přívětivější podobě předává analyzátorům. Kromě toho také jádro poskytuje API, které obsahuje úložiště pro lokální data vláken (*Thread Local Storage*), načítání konfiguračních souborů, uzamykatelné objekty a další pomocné funkce.

Komunikace mezi nástrojem Pin a jádrem a také mezi jádrem a analyzátory probíhá pomocí tzv. zpětných volání (*callback*). Analyzátory pomocí speciálních funkcí registrují obslužné rutiny pro události jako získání či uvolnění zámku, vytvoření nebo ukončení vlákna, volání funkce nebo přístup do paměti, které se spustí vždy, když v monitorovaném programu daná událost nastane. Toto zpětné volání také získá doplňující informace jako identifikátor vlákna či zámku, logickou adresu paměti nebo argument a návratovou hodnotu funkce.

### 2.2 Algoritmus AtomRace

AtomRace [5] je jeden z nejjednodušších algoritmů pro detekci časově závislé chyby nad daty, který nástroj ANaConDA poskytuje. Je založen na monitorování přístupů do paměti a rozpoznání situace, kdy k této chybě dojde přesně dle její definice – tedy když dvě

[1] http://www.fit.vutbr.cz/research/groups/verifit/tools/anaconda/
[2] https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool

různá vlákna přistoupí do stejného paměťového místa a alespoň jeden z přístupů je zápis.

ANaConDA pro každou událost poskytuje dvojici zpětných volání – jedno se vykoná před danou událostí v programu (tzv. *before callback*) a druhé po dané události v programu (tzv. *after callback*). Algoritmus AtomRace zapsaný pomocí těchto zpětných volání je na ukázce 1. Vlákna využívají sdílenou mapu, kde se pro jednotlivé logické adresy ukládají informace o přístupech. Ve zpětném volání `beforeAccess()` vlákno ověřuje, zda na danou adresu nepřistoupilo i jiné vlákno[3]. Pokud ano a alespoň jeden z přístupů je zápis, dojde k detekci chyby. Ve zpětném volání `afterAccess()` se ze sdílené mapy smaže informace o přístupu.

```
SharedMap Access;

beforeAccess(thread t, address a, mode m) {
  if (Access[a] == null) {
   Access[a] = (t, m);
  }
  else if (m == write ||
  Access[a].m == write)
   RACE DETECTED
}

afterAccess(thread t, address a, mode m) {
  if (Access[a].t == t)
    Access[a] = null;
}
```

**Zdrojový kód 1.** Pseudokód algoritmu AtomRace.

Algoritmus AtomRace má oproti komplikovanějším algoritmům, které sledují synchronizaci, jednu velkou výhodu – neprodukuje falešná chybová hlášení. Proto byl také upraven pro analýzu procesů a použit pro experimenty se studentskými projekty popsanými v sekci 4.

### 2.3 *hb*-relace a vektorové hodiny

Dynamické analyzátory získané informace typicky využívají pro tzv. extrapolaci a snaží se detekovat i takovou chybu, ke které v aktuálním plánu přepínání kontextu nedošlo, ale v jiném podobném by mohlo. Za tímto účelem je často nutné sledovat synchronizaci. Vhodnými nástroji jsou tzv. *happens-before* relace (dále značena jako *hb*-relace) a vektorové hodiny (*vector clock*). V nástroji ANaConDA ji využívá např. analyzátor FastTrack [8] pro detekci časově závislé chyby nad daty nebo analyzátory pro detekci porušení kontraktu [9]. Jelikož se jedná o důležitou techniku, je vhodné ji přizpůsobit i procesům, a proto je zde stručně popsána.

*hb*-relace je částečné uspořádání a využívá se pro určování pořadí událostí v různých vláknech s ohledem na synchronizaci. Formálně je *hb*-relace $\prec_{hb}$ nejmenší tranzitivně uzavřená relace nad množinou událostí $\{e_1, \cdots, e_n\}$ stopy $\tau = e_1 \cdots e_n$. Dvě události $e_j$ a $e_k$ jsou v *hb* relaci $e_j \prec_{hb} e_k$ vždy, když platí $j < k$ a zároveň alespoň jedna z následujících podmínek:

- Obě události byly vykonány stejným vláknem.
- Událost $e_j$ uvolňuje zámek, který událost $e_k$ obdrží.
- Událost $e_j$ je vytvoření nového vlákna $u$ ve vláknu $t$ a událost $e_k$ je provedena ve vláknu $u$.
- Událost $e_j$ je provedena ve vláknu $u$ a událost $e_k$ je čekání vlákna $t$ na ukončení vlákna $u$.

Pokud mezi dvěma událostmi *hb*-relace není, jsou považovány za souběžné a mohou být zdrojem paralelních chyb.

V nástroji ANaConDA se *hb*-relace implementuje s využitím vektorových hodin navržených pro algoritmus FastTrack, které udržují informaci o logickém čase každého vlákna v programu. Formálně jsou vektorové hodiny zobrazení z množiny všech vláken $T$ v programu do množiny přirozených čísel, $VC : T \rightarrow \mathbb{N}$. Každé vlákno $t \in T$ si udržuje vlastní hodiny $VC_t$. Hodnota $VC_t(t)$ představuje aktuální logický čas vlákna $t$. Hodnota $VC_t(u)$ pro každé $u \in T, u \neq t$ představuje poslední čas, kdy se vlákno $t$ synchronizovalo s vláknem $u$. Pro každou událost $e_u$ provedenou vláknem $u$ v čase $tm \leq VC_t(u)$ tedy platí, že je v *hb*-relaci s aktuální událostí $e_t$ ve vláknu $t$: $e_u \prec_{hb} e_t$.

Při synchronizaci dvou vláken dojde k aktualizaci jejich vektorových hodin s využitím operátoru spojení, který je definován následovně:

$$VC_1 \sqcup VC_2 = \lambda t.max(VC_1(t), VC_2(t))$$

Pro účely monitorování vícevláknových programů bylo popsáno pravidlo pro aktualizaci vektorových hodin při synchronizaci pomocí zámků. Každý zámek $L$ má vlastní vektorové hodiny $VC_L$. Při události, kdy vlákno $t$ uvolní zámek $L$, dojde k aktualizaci vektorových hodin zámku: $VC'_L = VC_t$. Tím se tedy uloží informace o vektorových hodinách vlákna $t$ v době provedení události. Při události, kdy vlákno $u$ obdrží zámek $L$, dojde k aktualizaci vektorových hodin vlákna $u$: $VC'_u = VC_u \sqcup VC_L$. Tím se tedy vlákno $u$ jednak synchronizuje s vláknem $t$, ale tranzitivně také s ostatními vlákny.

[3] Ověření `Access[a].t != t` není nutné, protože stejné vlákno nemůže dvakrát přistoupit na stejnou adresu, aniž by se vykonalo i zpětné volání `afterAccess()`, ve kterém se záznam o přístupu odstraní. Pokud je tedy `Access[a] != null`, jedná se o přístup jiného vlákna.

# 3. Návrh rozšíření

V této sekci budou popsány základní rozdíly mezi vlákny a procesy a návrhy řešení jednotlivých problémů, které také byly implementovány do nástroje ANaConDA. Nejpodstatnější otázkou je samotný přístup k analýze. Analyzátory jsou totiž také paralelní programy a mají stejný počet vláken (případně procesů) jako monitorovaný program. Všechna vlákna vykonávají stejný algoritmus a sdílí mezi sebou informace, na základě kterých obvykle jedno vlákno detekuje chybu. Pro procesy se nabízelo několik možností, z nichž se jako nejzajímavější jevil stejný přístup jako pro vlákna nebo centralizovaná analýza, kterou bude provádět jeden proces a ostatní se využijí pro sběr dat. První možnost má velkou výhodu, protože umožňuje použít stávající analyzátory také pro procesy (s lehkými modifikacemi kvůli problémům popsaným dále). Proto také byla implementována.

## 3.1 Zpětná volání a identifikátory

Většinu zpětných volání poskytovaných analyzátorům lze využít jak pro vlákna, tak i pro procesy. Problém je pouze s událostmi specifickými pro vlákna, jako je jejich vytvoření, start a ukončení. Tato zpětná volání se často využívají pro inicializaci struktur, ale také se jedná o důležité synchronizační události pro *hb*-relaci. Je tedy potřeba doplnit odpovídající volání také pro procesy.

Všechna zpětná volání získávají identifikátor vlákna, které provedlo danou událost. Stejnou informaci je potřeba dodat i pro procesy. Jelikož ale nástroj Pin poskytuje funkci `PIN_GetPid()`, není nezbytně nutné dosavadní volání přepisovat a je možné si v nich identifikátor vyžádat dodatečně.

## 3.2 Lokální a globální data

Jelikož je analyzátor paralelní program a na detekci chyby se podílí všechna vlákna (resp. procesy), je nutné využívat určité principy sdílení dat. Konkrétně existují dva důvody, proč je toto sdílení nutné:

1. Sdílení získaných informací mezi jednotlivými vlákny nebo procesy ( např. vektorové hodiny jednotlivých vláken nebo sdílená mapa využívaná v algoritmu AtomRace).
2. Sdílení lokálních dat mezi jednotlivými zpětnými voláními jednoho vlákna nebo procesu.

Jelikož vlákna sdílí adresový prostor, oba problémy je možné řešit pomocí globálních proměnných. Pokud proměnná obsahuje data sdílená mezi všemi vlákny, je nutné ji chránit zámkem, aby v samotném analyzátoru nevznikla paralelní chyba. Pro ukládání lokálních dat se využívá *Thread Local Storage* (TLS), ke kterému vlákna přistupují pomocí jednoznačného identifikátoru.

Pro procesy je situace opačná. Jejich adresový prostor je oddělený, takže nepotřebují TLS a lokální data mezi jednotlivými zpětnými voláními mohou být uložena v globálních proměnných. Sdílení dat mezi procesy je ale mnohem komplikovanější a je třeba přistoupit k mechanismům meziprocesové komunikace, jako např. sdílená paměť.

### 3.2.1 API pro komunikaci procesů

Vlákna nebo procesy často sdílí informace v dynamických STL kontejnerech, jako jsou vektory, fronty nebo mapy. Jejich uložení do sdílené paměti o fixní velikosti s sebou přináší problémy spojené s detekcí nedostatku paměti, jejím zvětšením a případnou realokací. Protože se implementačně jedná o komplikovaný problém, bylo API nástroje ANaConDA obohaceno o sdílené datové struktury, které jej řeší za vývojáře analyzátorů. API analyzátorům poskytuje sdílené číselné datové typy, vektor, mapu, oboustrannou frontu, řetězec, zámek apod. spolu s metodami a operacemi, aby se jejich použití nelišilo od klasických datových typů.

Problém detekce nedostatku paměti a jejího zvětšení se podařilo vyřešit díky knihovně `boost`, která poskytuje speciální alokátor právě pro ukládání dynamických datových struktur do sdílené paměti a objekt pro správu a zvětšování sdílené paměti. Zvětšení ale může způsobit realokaci, po které je nutné, aby si všechny procesy znovu namapovaly danou sdílenou proměnnou do svého adresového prostoru. Z tohoto důvodu je přístup k dynamickým strukturám realizován přes prostředníka, který procesům poskytuje informace nutné pro rozpoznání, zda od posledního přístupu došlo ke zvětšení sdílené paměti s proměnnou.

## 3.3 Logické adresy

Oddělený adresový prostor také způsobuje problémy ve zpětných voláních pro přístupy do paměti, která jako jednoznačný identifikátor dotčeného paměťového místa využívají logickou adresu. Tento přístup je korektní ve vícevláknových programech a je využit i ve výše popsaném algoritmu AtomRace, ale pro procesy nefunguje. Pokud totiž dva různé procesy přistoupí do stejného místa ve sdílené paměti, logické adresy mohou být naprosto odlišné. Intuitivně se nabízí využít fyzickou adresu, ale ta se za běhu programu může změnit (např. realokace). Byl tedy implementován překlad z logické adresy na jednoznačný identifikátor paměťového místa, který je stejný pro všechny procesy.

Překlad logické adresy je potřebný pouze tehdy,

když monitorovaný program přistupuje do sdílené paměti. Tuto novou adresu je pak možné odvodit z identifikátoru dané sdílené paměti a posuvu v rámci ní. Algoritmus pro překlad je tedy založen na monitorování vytváření a odstraňování sdílené paměti v analyzovaném programu. Každý segment sdílené paměti je v nástroji ANaConDA reprezentován pomocí tzv. tokenu. To je lokální datová struktura, v níž si každý proces uchovává informace o jednoznačném identifikátoru sdílené paměti a rozsahu logických adres, které jí přísluší. Jeho využití pro překlad je popsáno v algoritmu 1. Pro logickou adresu je potřeba najít token, který reprezentuje danou sdílenou paměť. Poté se z identifikátoru sdílené paměti vytvoří hash[4], který je pro všechny procesy stejný. K této nové "adrese" se přičte posun v rámci paměti a výsledkem je jednoznačný identifikátor paměťového místa.

**Algoritmus 1** Algoritmus pro překlad adres.

**Vstup:** logická adresa $a$, vektor $vc$ tokenů $t$
**Výstup:** pokud $a$ identifikuje místo $m$ ve sdílené paměti, pak jednoznačný identifikátor místa $m$

1: **if** $\exists t \in vc : a \in t.range()$ **then**
2: $\quad h = t.hash()$
3: $\quad offset = a - t.base()$
4: $\quad$ return $h + offset$

Implementace překladu v nástroji ANaConDA také řeší problémy spojené se změnou velikosti sdílené paměti a rozdíly v identifikaci POSIX a System V sdílené paměti (včetně anonymní sdílené paměti). Výsledný algoritmus se stal součástí API poskytované nástrojem ANaConDA a byl s úspěchem využit v implementaci algoritmu AtomRace.

Překlad logických adres se netýká pouze zpětných volání pro přístupy do paměti, ale také identifikátorů zámků a argumentů či návratových hodnot funkcí (např. synchronizačních funkcí pro semafory, které pracují s ukazateli).

## 3.4 Synchronizace

Posledním důležitým rozdílem mezi vlákny a procesy je způsob synchronizace. Mezi nejčastěji používané mechanismy patří zámky a semafory. Oba způsoby se dají využít v obou typech paralelních programů, ale protože vlákna častěji využívají zámky, je v nástroji ANaConDA aktuálně podpora pouze pro ně. U rozšíření pro procesy se od počátku předpokládalo, že by mohlo být užitečné i pro studenty předmětu Operační systémy, kteří jako jeden z projektů programují víceprocesový program s využitím semaforů. Z tohoto důvodu byla do nástroje ANaConDA přidána podpora pro jejich monitorování.

Podpora pro monitorování semaforů spočívá v dodání zpětných volání pro odpovídající synchronizační funkce spolu s podstatnými informacemi o daném semaforu. Pro účely algoritmů využívajících *hb*-relaci je také potřeba ji definovat pro semafory a určit pravidla pro aktualizaci vektorových hodin.

### 3.4.1 Zpětná volání pro semafory

Rozšíření poskytuje zpětná volání pro inicializaci semaforu a operace V (*down*) a P (*up*). Pokrývá POSIX[5] i System V[6] semafory. Analyzátorům je poskytován identifikátor semaforu, jeho aktuální hodnota, počet čekajících procesů a číslo, o kterou daná operace změní hodnotu semaforu (protože System V semafory umožňují inkrementovat a dekrementovat hodnotu semaforu o více než 1).

Z argumentů ani návratových hodnot synchronizačních funkcí bohužel není možné zjistit aktuální hodnotu semaforu ani počet čekajících procesů. Aktuální hodnotu je možné získat pomocí funkcí `sem_getvalue()` nebo `semctl()`. Proces analyzátoru a monitorovaný program ale nesdílí adresový prostor, takže by uvedené funkce musely být volány v kontextu aplikace, což má velmi negativní vliv na dobu běhu analýzy. Nástroj ANaConDA si tudíž pro každý semafor, který monitorovaný program používá, udržuje reprezentaci, na které zrcadlí operace prováděné v monitorovaném programu. Zná tak aktuální hodnotu a je schopen odvodit, kdy bude proces zablokován a zařazen do fronty. Tyto informace potom sděluje analyzátorům.

### 3.4.2 *hb*-relace pro semafory

*hb*-relace popsaná v sekci 2.3 definuje uspořádání dvou událostí vzhledem k synchronizaci pomocí zámků. Na ty je možno nahlížet jako na speciální semafory splňující dvě podmínky:

1. Jejich maximální hodnota je 1.
2. Uvolnit zámek (čili provést operaci *up* nad binárním semaforem) může jenom to vlákno nebo proces, které jej aktuálně drží (čili bylo poslední, které úspěšně provedlo operaci *down* nad binárním semaforem).

Protože obecné semafory nesplňují ani jednu podmínku, je nutné *hb*-relaci rozšířit o definici uspořádaní

[4] Prozatím je tolerována možnost případného konfliktu, který stejný nebo podobný hash pro dvě různá paměťová místa může způsobit, protože očekáváme, že k němu bude docházet velmi zřídka. Pokud by konflikt způsoboval problémy, je samozřejmě možné dodat přísnější porovnávání na rovnost dvou přeložených "adres".

[5] https://linux.die.net/man/7/sem_overview

[6] http://man7.org/linux/man-pages/man7/sysvipc.7.html

dvou událostí vzhledem k synchronizaci pomocí semaforů.

| $P_1$ | $P_2$ |
|---|---|
| acquire(m) | $e_{21}$ |
| $e_{11}$ | $e_{22}$ |
| release(m) | $e_{23}$ |
| $e_{12}$ | acquire(m) |
| | $e_{24}$ |

a) Synchronizace zámkem

| $P_1$ | $P_2$ |
|---|---|
| $e_{11}$ | $e_{21}$ |
| up(s) | $e_{22}$ |
| $e_{12}$ | down(s) |
| | $e_{23}$ |

b) Synchronizace semaforem

**Obrázek 1.** Formování *hb*-relace mezi dvěma procesy při synchronizaci.

Na obrázku 1 je ukázka *hb*-relace mezi dvěma procesy způsobená synchronizací. Případ a) zachycuje vznik relace při synchronizaci zámky, kdy uvolnění zámku jistě předchází (*happens-before*) jeho obdržení jiným procesem. Teoreticky je na semafory možno nahlížet stejným způsobem a spojovat relací operace *up* a *down* jako na ukázce b). Semafory ale může najednou používat (a inkrementovat) několik procesů a je problém určit, které dvojice operací *up* a *down* nakonec relaci vytvoří.

| $P_1$ | $P_2$ | $P_3$ | $P_4$ |
|---|---|---|---|
| wait(s) | $e_{21}$ | $e_{31}$ | $e_{41}$ |
| | wait(s) | $e_{32}$ | $e_{42}$ |
| | | up(s) | $e_{43}$ |
| | | $e_{33}$ | up(s) |
| | continue(s) | | |
| continue(s) | | | |

**Obrázek 2.** Stopa programu se 4 procesy, které používají semafor $s$. Procesy $P_1$ a $P_2$ jsou zablokovány, zařazeny do fronty čekajících procesů a následně uvolněny procesy $P_3$ a $P_4$.

V dalším výkladu bude operace *down* rozdělena na dvě fáze:

1. Fáze *wait* nastane, když proces nebo vlákno provádí operaci *down* nad semaforem, jehož hodnota je 0, což způsobí jeho pozastavení.
2. Fáze *continue* nastane, když proces úspěšně dokončí operaci *down* a může pokračovat v exekuci. K této fázi dojde buď v případě, kdy hodnota semaforu byla kladná a proces vůbec nebyl zablokován, nebo byl uvolněn z čekání díky tomu, že jiný proces daný semafor inkrementoval.

Uvažujme nyní situaci na obrázku 2. Procesy $P_1$ a $P_2$ provedou operaci *down* nad semaforem $s$ a oba budou zablokovány. Následně proces $P_3$ provede operaci *up*, inkrementuje hodnotu semaforu a jako následek by měl být uvolněn jeden z čekajících procesů[7]. Plánovač ale místo toho přidělí procesor procesu $P_4$, který taktéž provede operaci *up* a uvolní druhý proces. Procesy $P_2$ a $P_1$ jsou následně v tomto pořadí uvolněny, úspěšně dokončí operaci a provedou fázi *continue*. Otázkou je, které události by měly být spárovány *hb*-relací.

Článek [10] pro události $e_j \prec_{hb} e_k$ přidává následující pravidlo (zde zjednodušeno):

- $e_j$ je operace *up* provedená procesem $p_1$, která odblokovala proces $p_2$ pozastavený operací *down*, a $e_k$ je fáze *continue* v procesu $p_2$.

Pro situaci na obrázku z něho vyplývají následující relace:

$$P_3.up() \prec_{hb} P_2.continue()$$

$$P_4.up() \prec_{hb} P_1.continue()$$

Po důkladném zvážení (popis dalších možných přístupů je nad rámec tohoto článku) byl pro rozšíření v nástroji ANaConDA zvolen podobný přístup, který výše uvedenou definici rozšiřuje o další pravidlo:

- $e_k$ je neblokující provedení operace *down* (čili došlo pouze k fázi *continue*) a $e_j$ je operace *up*, která byla nutná pro to, aby událost $e_k$ mohla být neblokující.

Důsledek tohoto pravidla je znázorněn na obrázku 3. Jedná se o velmi podobnou stopu jako v předchozím případě, ale operace *up* byly provedeny dříve, než operace *down*, takže procesy $P_1$ a $P_2$ nebyly zablokovány. Z logiky synchronizace by ale *hb*-relace měla vypadat stejně, jako pro obrázek 2.

### 3.4.3 Implementace *hb*-relace pro semafory

Při implementaci algoritmu pro formování *hb*-relace je potřeba pro každou operaci *down* zjistit buď:

1. operaci *up*, která uvolnila proces z čekání, nebo
2. operaci *up*, která procesu umožnila pokračovat bez zablokování.

[7] POSIX ani System V implementace nezaručují, že fronta čekajících procesů je FIFO, a tudíž dopředu nelze určit, který z čekajících procesů bude uvolněn.

| $P_1$ | $P_2$ | $P_3$ | $P_4$ |
|---|---|---|---|
| $e_{11}$ | $e_{21}$ | $e_{31}$ | $e_{41}$ |
| $e_{12}$ | $e_{22}$ | $e_{32}$ | $e_{42}$ |
| $e_{13}$ | $e_{23}$ | up(s) | $e_{43}$ |
| $e_{14}$ | $e_{24}$ | $e_{33}$ | up(s) |
| $e_{15}$ | continue(s) | | |
| continue(s) | | | |

**Obrázek 3.** Stopa programu, kdy jsou operace *up* nad semaforem *s* provedeny dříve než operace *down*, takže ty proběhnou bez blokování odpovídajících procesů.

Součástí rozšíření se stala implementace, která je schopna oba body splnit, ale její popis je nad rámec tohoto článku. Je založena na ukládání informací o operacích *up* do fronty spolu s informací o čekajících procesech a následném hledání vhodného záznamu, který mohl daný proces uvolnit. Dále je uveden algoritmus znázorňující, jak se synchronizace pomocí semaforů dotkne implementace *hb*-relace pomocí vektorových hodin za předpokladu, že výše uvedené informace jsou k dispozici.

V sekci 2.3 byly popsány vektorové hodiny, operátor spojení a jeho použití pro aktualizaci vektorových hodin vláken, která se synchronizovala pomocí zámků. Dvojice vláken si předává informace přes vektorové hodiny zámku a formování relace probíhá v okamžiku obdržení zámku. Algoritmus pro semafory je velmi podobný, ale protože se pomocí jednoho semaforu může synchronizovat více dvojic vláken/procesů, je potřeba mít vektorové hodiny pro každou operaci *up*. Formování relace taktéž proběhne ve fázi *continue*.

Algoritmus 2 zjednodušeně popisuje aktualizaci vektorových hodin procesu, který provádí fázi *continue*. Oproti synchronizaci pomocí zámků přibyl řádek 2, který hledá vhodnou operaci *up*, a řádek 3, kdy je nutné znát vektorový čas procesu v době provádění dané operace. Samotná aktualizace na řádku 4 zůstává stejná.

**Algoritmus 2** Algoritmus pro formování *hb*-relace.

**Vstup:** semafor $s$ s frontou $s.q$ operací *up*, proces $p_d$ provádějící fázi *continue*, vektorové hodiny $vc_d$ procesu $p_d$

**Výstup:** aktualizované hodiny $vc'_d$ procesu $p_d$

1: **for** $up \in s.q$ **do**
2: **if** $up$ uvolnil $p_d \vee up$ způsobil neblokující *down* procesu $p_d$ **then**
3: $vc_u = up.vc()$
4: $vc'_d = vc_d \sqcup vc_u$
5: $s.q.remove(up)$
6: **break**

## 4. Experimenty

Implementace navržených řešení v nástroji ANaConDA byla ověřena sadou automatických testů, které využívají tři speciální analyzátory implementované za tímto účelem (a nedetekují žádné paralelní chyby). Pro detekci časově závislých chyb nad daty ve víceprocesových programech byla vytvořena nová implementace analyzátoru AtomRace, která je téměř totožná s implementací pro vícevláknové programy, ale využívá překlad logických adres a sdílené API pro komunikaci procesů. S tímto analyzátorem byly provedeny experimenty nad studentskými projekty z předmětu Operační systémy.

K dispozici bylo 19 projektů, které obdržely maximální bodové hodnocení, protože v nich klasické testy neodhalily žádnou chybu. V 17 z těchto projektů chybu nenašel ani AtomRace, ale ve dvou z nich úspěšně souběh ve sdílené paměti detekoval. Nástroj ANaConDA umí získávat informace o průběhu stopy (*backtrace*), který ulehčuje hledání chyby ve zdrojovém programu. Pomocí sdíleného API byla tato funkcionalita implementována i do analyzátoru AtomRace. Výstup analýzy pak vypadá podobně jako na ukázce:

```
Data race on memory address
0x7f5a6f008000 detected.
  Process 19298 read from <unknown>
    accessed at line 412 in file x.c
  Process 19243 written to <unknown>
    accessed at line 416 in file x.c
```

Získávání názvu proměnné zatím není podporováno, proto se ve výpisu vyskytuje `<unknown>`. Uživateli jsou tedy sděleny přímo řádky kódu, kde k chybě došlo, a může tak chybu snáze opravit. V budoucnu se plánuje poskytnutí nástroje ANaConDA s rozšířením pro analýzu procesů právě studentům tohoto kurzu, aby jim pomohla při vývoji projektů. Informace pro nalezení chyby pro ně tedy budou klíčové.

## 5. Závěr

Výsledkem práce je unikátní rozšíření pro detekci paralelních chyb ve víceprocesových programech, které žádný ze známých nástrojů pro dynamickou analýzu paralelních programů neposkytuje. Článek popisuje problémy, ke kterým u procesů dochází, a zjednodušený návrh jejich řešení. Nástroj ANaConDA tak vývojářům analyzátorů poskytuje sdílené datové typy, překlad logických adres, potřebná zpětná volání, monitorování synchronizace pomocí semaforů a také vzorovou implementaci formování *hb*-relace. Některé z těchto funkcionalit byly úspěšně využity v algoritmu

AtomRace, který detekoval časově závislou chybu nad daty ve studentských projektech, kterou klasické testy nenašly.

V blízké budoucnosti bude rozšířena sada automatických testů pro ověření funkcionality rozšíření i na okrajových případech. Také budou provedeny další experimenty. Pro případnou navazující práci se nabízí hned několik rozšíření. Je možné dodat podporu pro programy, které kombinují procesy i vlákna, pro distribuované systémy, nebo pro jiné formy analýzy (např. zmíněná centralizovaná analýza). Popsané rozšíření také nepodporuje Windows semafory a není definována *hb*-relace pro operace *up* a *down* o více než 1. Také je třeba vytvořit další analyzátory využívající implementované rozšíření (např. implementovat analyzátor FastTrack využívající *hb*-relaci).

## Poděkování

Ráda bych poděkovala zejména Ing. Aleši Smrčkovi, Ph.D. a Ing. Janu Fiedorovi, Ph.D., za jejich vedení a rady po celou dobu mé práce na nástroji ANaConDA.

## Literatura

[1] Orit Edelstein, Eitan Farchi, Evgeny Goldin, Yarden Nir, Gil Ratsaby, and Shmuel Ur. Framework for testing multi-threaded java programs. *Concurrency and Computation: Practice and Experience*, 15(3-5):485–499, 2003.

[2] Cormac Flanagan and Stephen Freund. The roadrunner dynamic analysis framework for concurrent programs. *ACM SIGPLAN/SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, pages 1–8, January 2010.

[3] Nicholas Nethercote and Julian Seward. Valgrind: A framework for heavyweight dynamic binary instrumentation. In *Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation*, volume 42 of *PLDI '07*, pages 89–100, New York, NY, USA, 2007. ACM.

[4] Jan Fiedor and Tomáš Vojnar. Anaconda: A framework for analysing multi-threaded C/C++ programs on the binary level. In *Proc. of 3rd International Conference on Runtime Verification—RV'12*, volume 7687 of LNCS, pages 35–41, Istanbul, Turkey, 2012.

[5] Zdeněk Letko, Tomáš Vojnar, and Bohuslav Křena. Atomrace: Data race and atomicity violation detector and healer. In *Proceedings of the 6th Workshop on Parallel and Distributed Systems: Testing, Analysis, and Debugging*, PADTAD '08, pages 7:1–7:10, New York, NY, USA, 2008. ACM.

[6] Jan Fiedor, Monika Mužikovská, Aleš Smrčka, Ondřej Vašíček, and Tomáš Vojnar. Advances in the anaconda framework for dynamic analysis and testing of concurrent c/c++ programs. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2018, page 356–359, New York, NY, USA, 2018. Association for Computing Machinery.

[7] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. Pin: Building customized program analysis tools with dynamic instrumentation. *SIGPLAN Not.*, 40(6):190–200, June 2005.

[8] Cormac Flanagan and Stephen Freund. Fasttrack: efficient and precise dynamic race detection. In *Communications of the ACM*, volume 53(11), pages 93–101, 01 November 2010.

[9] Ricardo J. Dias, Carla Ferreira, Jan Fiedor, João M. Lourenço, Aleš Smrčka, Diogo G. Sousa, and Tomáš Vojnar. Verifying concurrent programs using contracts. In *2017 IEEE International Conference on Software Testing, Verification and Validation*, pages 196–206, 2017.

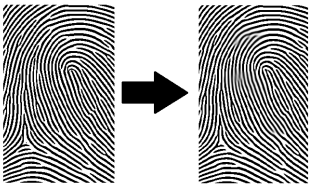
[10] Rahul Agarwal and Scott Stoller. Run-time detection of potential deadlocks for programs with locks, semaphores, and condition variables. pages 51–60, 01 2006.

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

# Rekonštrukcia poškodených častí odtlačkov prstov s využitím neurónových sietí

Michael Halinár



**Abstract**
Táto práca sa zaoberá problematikou rekonštrukcie poškodených odtlačkov prstov s využitím neurónových sietí. Rekonštruované odtlačky sú syntetické, ako aj poškodenia, ktoré sú do odtlačkov vkladané. Sieť sa snaží z poškodeného odtlačku dostať pôvodný, pri zachovaní všetkých dôležitých aspektov odtlačku prsta. Táto práca rieši dva problémy a to výber vhodnej architektúry neurónovej siete a vytvorenie databáze syntetických odtlačkov. Keďže voľne dostupné databáze reálnych odtlačkoch sú nedostatočne veľké. Špecifická architektúra neurónovej siete je schopná rekonštruovať poškodené obrázky. Takáto architektúra sa nazýva *autoencoder*. Je to druh neurónovej siete, ktorá využíva konvolučné vrstvy. Po naučení dokáže veľmi dobre redukovať šum a rekonštruovať vstupy. V práci je použitý tento typ architektúry na rekonštrukciu poškodených častí odtlačku. Pre simuláciu poškodenia je vložená do odtlačku prstu bradavica. Sieť dokáže rekonštruovať poškodené časti pri zachovaní pôvodných vlastností prsta vo väčšine prípadov. Rekonštrukcia bradavice je závislá od miesta a veľkosti bradavice. Kvalita rekonštrukcie je meraná pomocou programu Verifinger. Táto práca ukazuje, že s použitím vhodnej neurónovej siete, ktorá bude naučená na dostatočne veľkej databáze, je možné dosiahnuť rekonštrukciu poškodeného odtlačku prsta. Odtlačky prstov sa počas života môžu meniť rôznymi spôsobmi. Vytvorená aplikácia dokáže eliminovať tento faktor, zrekonštruovať pôvodný odtlačok a umožniť lepšiu identifikáciu vlastníka odtlačku.

**Kľúčové slová:** neurónová sieť — odtlačok prsta — rekonštrukcia

**Priložené materiály:** Demonštračné video — Stiahnuteľný kód

xhalin01@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

## 1. Úvod

Odtlačky prstov sa počas života menia, avšak ich individuálne znaky zostávajú nemenné. To je jeden z dôvodov, kvôli ktorým sú odtlačky prsta výhodnou cestou pre identifikáciu osôb. Uvažujme teda systém, ktorý je určený na identifikáciu osôb podľa odtlačkov prstov a obsahuje odtlačky konkrétnej osoby. Medzičasom sa však danej osobe poškodí pokožka a nie je možné už viac identifikovať túto osobu na základe pôvodných odtlačkov. S využitím neurónovej siete sme schopný rekonštruovať pôvodný odtlačok a identifikovať tak danú osobu.

Existuje mnoho druhov neurónových sietí, ktoré majú rôzne využitia. Použitie vhodnej architektúry je jedným z problémov ktoré rieši táto práca. Ďalším problémom je nedostatočná veľkosť voľne dostupných databáz, na tento fakt poukazuje aj práca [1]. Kde je malá databáza jedným z faktorom, ktoré ovplyvnili

výsledky.

Existuje pomerne málo výskumov v tejto oblasti. Podľa práce [2], je táto prvou kde je použitá architektúra GAN (Generative adversarial network) na generovanie odtlačkov prstov. GAN vo voľnom preklade znamená generatívna konfrontačná sieť. Nazýva sa tak preto lebo obsahuje dve časti a to generátor a diskriminátor. Pričom diskriminátor hodnotí výstupy vytvorené generátorom a snaží sa tak vylepšiť jeho schopnosti vytvárania výstupu. V tejto práci generátor aj diskriminátor sa skladajú z piatich vrstiev. Použité boli databáze odtlačkov FVC-2006 a Poly-U. Celkovo obsahovali zhruba 3000 obrázkov odtlačkov, pričom bol vystrihnutý ich stred a zredukovaný na veľkosť 64x64 pixelov. Treba dodať že odtlačky neboli jedinečné, čiže jeden prst bol nasnímaný viac krát. Výsledkom práce bolo generovanie syntetických odtlačkov, ktoré sa podobali na reálne odtlačky zo spomenutých databáz. V tomto riešení žiadne testy nepotvrdzovali kvality výstupu.

Ďalšia práca v tejto oblasti má názov Fingerprint Inpainting with Generative Models [1]. Ktorej cieľom bolo dokresľovať časti odtlačku, ktoré boli nahradené šedým regiónom. Použitá architektúra siete je opäť GAN kde generátor pozostáva z architektúry *autoencoder*. Databáza odtlačkov pozostávala z obrázkov s veľkosťou 128x128 pixelov, pričom boli vkladané šedé oblasti s rozmermi 32x32 pixelov, ktoré boli následne rekonštruované. Učenie použitej siete bolo veľmi časovo náročné, autori uviedli že riešenie nemuselo konvergovať k správnemu riešeniu. Učenie trvalo týždeň pri použití grafickej karty nVidia GTX 1060. Testy kvality rekonštrukcie boli urobené na klasifikátore odtlačkov. Zvýšenie správnej klasifikácie narástlo zhruba o tretinu po použití rekonštrukcie.

Táto práca sa zaoberá návrhom a implementáciou architektúry a jej parametrov pre čo najlepšiu rekonštrukciu syntetických odtlačkov. Kvôli nedostatočne veľkým databázam reálnych odtlačkov prstov, sa sieť učí na syntetických odtlačkoch s ochoreniami do nich vložených. Príprava databáze odtlačkov je teda ďalším bodom tejto práce. Výsledky aplikácie sú merané pomocou programu Verifinger, ktorý meria zhodu odtlačku pred a po rekonštrukcii. Nárast hodnoty zhody sa pohybuje okolo 10%. Vytvorená sieť je schopná, rýchlo sa učiť a kvalitne rekonštruovať odtlačky. Pomerne malý počet vrstiev a využitie *downsapling* vrstiev zabezpečuje rýchle učenie aj na výpo-četne slabom hardvéri.

Táto neurónová sieť opravuje simulované ochorenia kože, na rozdiel od predchádzajúcej práce[1], kde boli vkladané poškodenia v tvare šedého obdĺžnika. Rekonštruované odtlačky sa veľmi podobajú odtlačkom pred poškodením. Zlepšenie porovnávacieho skóre sa zvýšilo po rekonštrukcii o 16%.

## 2. Odtlačky prstov

Odtlačok prsta vzniká odtlačením papilárnych línií na určitý povrch. Forma papilárnych línií je určená hlavne génmi jedinca [3]. Ak nepríde k vážnemu mechanickému alebo inému poškodeniu, línie zostávajú rovnaké počas celého života, pokiaľ zanedbáme faktor starnutia kože, ktorý je prirodzený. Papilárne línie sú formované v nižšej vrstve kože, čo zabezpečuje ich obnovu v prípade ľahkého poškodenia [4]. Línie vytvárajú vzory, vďaka ktorým môžeme odtlačky jednoznačne identifikovať.

## 3. Databáza

Databáza odtlačkov bola vytvorená pomocou nástroja Anguli[1]. Bolo vytvorených 1000 jedinečných odtlačkov prstov. Pred tým ako môžu byť použité, je nutné upraviť ich rozmery, tak aby konvolučná sieť mohla robiť *downsampling*[2] pokiaľ možno, čo najviac krát. Teda obrázky, ktorých rozmer výšky a šírky je mocnina dvojky sú najvhodnejšie vstupy pre konvolučné siete. Vygenerované obrázky nástrojom Anguli majú rozmer 400x275 pixelov. Výška 400 pixelov je primerane vhodná, pri takomto rozmere môžeme robiť *downsampling* až 5 krát. Šírka 275 pixelov nie je vhodná pre *downsampling* preto bola šírka obrázku zúžená na 256 pixelov. Na náhodné pozície potom boli vložené syntetické bradavice, vytvorené programom SyFDaS [3].

Prvý dataset obsahuje 10 bradavíc s priemerom približne 30 pixelov. Každá bradavica bola naviac 3-krát otočená za účelom vytvorenia ďalších vzoriek bradavíc. Spolu teda 40 bradavíc bolo vložených do 1000 syntetických odtlačkov na náhodné miesto. Kombinácia natočenia bradavice a vloženie na náhodné miesta sa ukázala ako dostatočná varianta k vytváraniu 1000 originálnych bradavíc, nakoľko boli bradavice manuálne vystrihované z obrázkov generovaných pomocou programu SyFDaS [4].Bradavice majú priemer približne 80 pixelov. Do každeho odtlačku bolo vložených 6 bradavíc.

## 4. Návrh a implementácia

Ako prvé je potrebné vybrať typ neurónovej siete, ktorá sa bude učiť odtlačky rekonštruovať. Na výber je niekoľko druhov knižníc a architektúr sietí, ktoré je

[1] https://dsl.cds.iisc.ac.in/projects/Anguli/
[2] https://en.wikipedia.org/wiki/Downsampling_(signal_processing)
[3] https://www.fit.vut.cz/research/product/600/.cs
[4] https://www.fit.vut.cz/research/product/600/.cs

možné použiť. Architektúra GAN je veľmi vhodná pre generovanie nových výstupov, ktoré sa nenachádzajú medzi trénovacou množinou. Z tohoto dôvodu bude použitá architektúra *autoencoder*. Po naučení siete na vytvorenej databáze bude treba overiť kvalitu rekonštrukcie, naskytá sa tu niekoľko možností. Jednou z nich je meranie kvality samotného odtlačku podľa NFIQ[5]. Ďalšou možnosťou je meranie hodnoty zhody odtlačkov pred a po rekonštrukcii, napríklad pomocou programu Verifinger[6]. Po vyhodnotení výsledkov bude nasledovať ich analýza.

## 5. Architektúra siete

Ako východzia architektúra bola použitá sieť *autoencoder*[5]. Počet konvolučných vrstiev bol navýšený. Aktivačné funkcie a veľkosti filtrov boli zachované. Dôležitou zmenou je počet filtrov, ten bol navýšený vzhľadom na skutočnosť že pôvodná sieť pracovala s obrázkami s veľmi nízkymi rozmermi - 28x28 pixelov, oproti syntetickým odtlač-kom s rozmermi - 400x256 pixelov. Výsledná schéma architektúry je znázornená na obrázku 1. Kde zelené vrstvy predstavujú *downsampling*, modré *upsampling* a žlté predstavujú konvolučné vrstvy. Aktivačné funkcie sú typu ReLu[7].

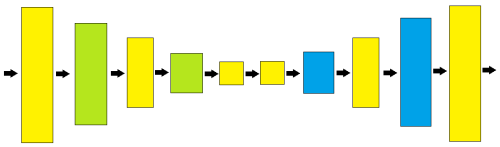


**Figure 1.** Architektúra siete.

## 6. Experimenty

Učenie prebiehalo na procesore Intel i5-8250 v niekoľkých iteráciách. Prvá iterácia pracovala nad poškodenými odtlačkami a pôvodnými odtlačkami bez poškodenia 4. V druhej iterácii sa vytvorili nové váhy, v tejto iterácií už sa sieť učila na výstupoch z prvej iterácie a originálnych obrázkoch. Týmto spôsobom prebehli tri iterácie a vznikli tri rôzne skupiny váh pre sieť. Poškodený obrázok je teda rekonštruovaný pomocou týchto rôznych troch váh, kde prvé váhy sa snažia rekonštruovať línie a posledné váhy sú naučené aby už iba vyhladzovali výsledok s predchádzajúcej iterácie. Pričom počty epoch boli: 200 pre prvú iteráciu, 80 pre druhú a tretiu iteráciu. Po ďalších iteráciách už nastával *overfitting*[8]. Týmto spôsobom dostávame rekonštruované odtlačky, ktorých porovnávacie skóre je vyššie v niektorých prípadoch až o viac ako 40% 3. Väčšie bradavice nie sú vždy rekonštruované správne, ak sa poškodenie nachádza napríklad v strede odtlačku (delte), sieť má problém korektne dokresliť túto časť 2. Porovnávanie prebiehalo manuálne na prvých 100 odtlačkoch, pre výpočet priemerného nárastu zlepšenia. Manuálne zmerané porovnávacie skóre sa nachádza v priložených súboroch. Celkovo sa zlepšilo porovnávacie skóre po rekonštrukcii o 16%.

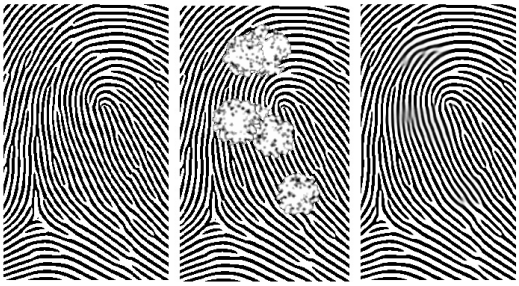


**Figure 2.** Kvalitná rekonštrukcia bradavíc.

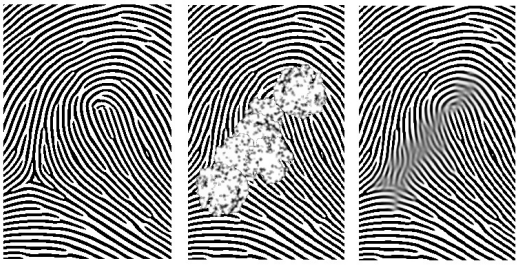


**Figure 3.** Nekvalitná rekonštrukcia bradavíc.

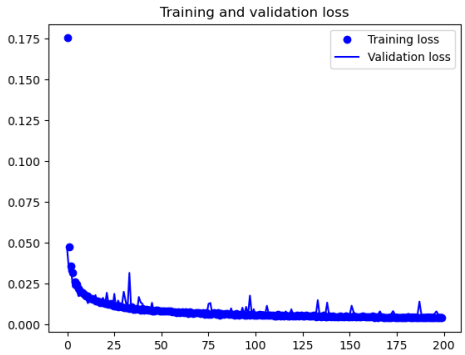



**Figure 4.** Priebeh učenia prvej iterácie.

Jedným z vedľajších produktov tejto práce je aj detektor polohy a veľkosti bradavice. Tento detektor 5, používa rovnakú sieť ako pre rekonštrukciu, avšak je naučený iba jednou epochou. Takáto sieť ešte nevie čo sa od nej očakáva preto oblasť s bradavicou celú zafarbí na šedo. Pomocou filtra teraz môžeme bradavicu zvýrazniť a zistiť tak jej polohu a tvar. Tento detektor slúži predovšetkým ako ilustrácia možností neurónových sietí a je jedným z možných rozšírení do

[5] https://github.com/usnistgov/NFIQ2
[6] https://www.neurotechnology.com/verifinger.html
[7] https://www.kaggle.com/dansbecker/rectified-linear-units-relu-in-deep-learning
[8] https://elitedatascience.com/overfitting-in-machine-learning

budúcna.

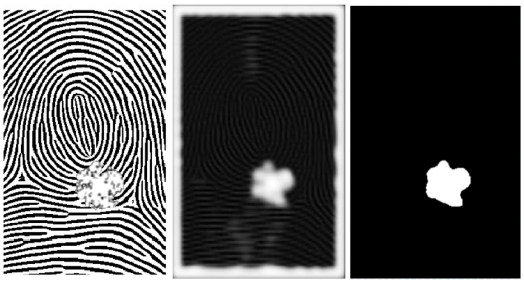


**Figure 5.** Detektor bradavice. Zľava do prava: poškodený odtlačok, predikcia rekonštrukcie a úprava predikcie filtrom.

## 7. Záver

Vytvorená neurónová sieť dokáže pomerne dobre rekonštruovať syntetické odtlačky s bradavicami. Kvalita rekonštrukcie bola meraná pomocou programu Verifinger. Učenie siete trvá zhruba 36 hodín. Skóre zhody pôvodného obrázku a obrázku po rekonštrukcii sa zvýšilo v priemere o 16%. Rekonštrukcia poškodenia syntetických odtlačkov je možná, bola ukázaná rekonštrukcia bradavíc o priemere zhruba 80 pixelov. Oproti konvenčným spôsobom táto metóda nevyžaduje náročné matematické výpočty, a je taktiež škálovateľná. Čiže zväčšením robustnosti siete môžeme získavať ešte lepšie výsledky.
Ako možné rozšírenie sa naskytá použitie inej architektúry, zväčšenie počtu vrstiev alebo rekonštrukcia reálnych odtlačkov prstov. Ďalším možným rozšírením je naučenie siete rekonštruovať ďalšie ochorenia alebo poškodenia. Preskúmanie možností detektora poškodenia alebo generátora je taktiež jedno z možných rozšírení. Ďalším možným vylepšením je zautomatizovanie merania kvality rekonštrukcie.

## Poďakovanie

Chcel by som poďakovať môjmu vedúcemu diplomovej práce, doktorovi Ondřejovi Kanichovi za poskytnuté materiály a pomoc pri vytváraní práce.

## Literatúra

[1] Group 107. Fingerprint inpainting with generative models. 2018.

[2] Shervin Minaee and AmirAli Abdolrashidi. Finger-gan: Generating realistic fingerprint images using connectivity imposed GAN. *CoRR*, abs/1812.10482, 2018.

[3] Jain Anil, Flynn Patrick, and Ross Arun. *Handbook of Biometrics*. Springer, 1 edition, 2008.

[4] Martin Drahanský. *Fingerprint Recognition Technology - Related Topics*. Lambert Academic Publishing, Lambert Academic Publishing, 2011.

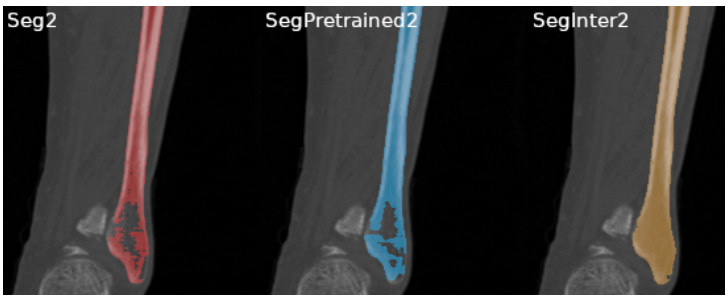
[5] Francois Chollet. Building autoencoders in keras. online, 2016.

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

# Benchmarking medical segmentation models with limited training sets

Kateřina Trávníčková*, Oldřich Kodym




**Abstract**
Deep learning based medical data segmentation methods can provide excellent results already. However, these results are obtained mostly thanks to the large training data sets. Obtaining the sufficient amount of correct annotations might be problematic in the medical field. This paper describes the problem of training medical segmentation models with limited annotations and proposes solutions to address the issue.
We compare the baseline segmentation model group with two other model groups. These groups use different means to battle the lack of data problem. First group is pretrained in unsupervised manner and the second one uses human interaction in form of guidance clicks. We train 14 models for each group on subsets with varying number of patients.
Segmentation model trained on small number of patients has better results when pretrained in unsupervised manner on the whole trainig set with 70 patients. Better results are obtained with the interactive method, where training on only two patients reaches Dice score 0.929 whereas the preitrained model reaches 0.830 and the baseline model only 0.749.
**Keywords:** Segmentation — Deep learning — Medical data

**Supplementary Material:** *N/A*

*xtravn24@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

## 1. Introduction

Medical data analysis with use of deep learning is an important field since the successful application of its methods can lessen the workload for medical professionals and can also generally lead to healthcare quality improvement. Although the methods of medical data segmentation have highly improved in the past few years, the shortage of professionally created annotations is still an issue.

This work focuses on the task of semantic segmentation of the long bones in human computed tomography (CT) scans, specifically on the possibilities of using small amounts of annotated data while maintaining satisfactory segmentation quality. Cases where no annotated data are present for the particular segmentation class are also considered and transfer learning view is used as a part of the selected solution. The main goal of this work is to examine ways of solving these challenging use-cases, and to observe the behavior of the selected solutions on varying number of training data rather than to compete with the quality of

the state-of-the-art methods by tweaking the network architecture.

Segmentation of medical data requires high level of precision. This is why semi-automatic and more traditional methods, such as thresholding or graph cut [1], are still being used even with recent development of deep learning methods. This can be partly because of the lack of the training data for specific cases, partly because the low error tolerance is a necessity while dealing with real patients and high level of control over the segmentation method is a must. Alternatively, the classical methods can be used as an integrated part of the deep learning method, such as described in paper by Kodym et al. [2].

Segmentation models, even from different domains, usually share the encoder-decoder architecture type, as used in several other works [3, 4, 5]. In the medical domain, the most widespread architecture is the U-net [6] architecture. Authors of this model followed the encoder-decoder trend in segmentation models and developed an architecture that became a standard of medical data segmentation. Which also helps to improve the segmentation results is converting the whole model to 3D with use of 3D convolutions as Çiçek et al. have done in [7].

Apart from architecture development itself, some authors focus on other means of raising the quality of the segmentation results, for example transfer learning. Authors of the Models Genesis [8] experimented with transferring the knowledge obtained from image restauration task to the task of segmentation (and other tasks as well). The general model is trained to deal with several different types of distortion. Authors claim that this practise helps the model to learn general data character and structure from unlabeled data, speeding up the convergence of the segmentation training and providing better segmentation results than randomly initialized models.

As the automatic segmentation models can be sensitive to domain changes and their accuracy can drop in specific cases, several attempts to benefit from user interaction have also been made [9, 10, 11]. In the paper [11] for example Sakinis et al. use foreground and background click maps as a part of the model input to help the model with the right object selection. This can help to battle small domain shifts and even provide a mean to generalize the segmentation to previously unseen types of data, such as irregularly shaped tumors.

We are using a variation of the earlier mentioned U-net architecture as a baseline method for the task of longitudinal bone segmentation in axial CT slices. Apart from that, two extensions are selected and tested. The first extension benefits from transfer learning principles described by authors of Models Genesis [8]. The model is pretrained on image restauration task without needing any labels and then finetuned to the segmentation task. In the second extension, user interactions are used to improve the segmentation quality similarly to the work of Sakinis et al. [11]. This interactive model uses user clicks for object or background specification.

All three models have been trained on several different subsets of data with varying number of patients in each set. The smallest set only contains one patient. Three groups of models have been created, one for each method. This benchmarking helps to show the benefit of each method for different data amount scenarios.

The main contribution of this paper lies in the comparison of the selected models and examination of their behavior when being trained with variably sized training subsets. Our experiments suggest that using unlabeled data for pretraining can be beneficial to some extent. Best improvement was reached in the scenario with one labeled patient. This scenario best shows the benefit of using unsupervised pretraining, however the final reached Dice score is quite low. More research could be done with bigger amount of unlabeled pretraining data to prove that this method can be used in practice. Alternatively, this method can be used only as a preprocessing for semi-automatic methods, serving as a means of time reduction for human experts. Experiments with interactive segmentation yield promising results even in the one patient scenario, with minimal number of interaction. In this use-case, segmentation with only one click in each axial slice yields much better results than the baseline method. Interactive training with two patients reaches Dice score 0.929, which is considerably high value for such a small training set.

## 2. Selected solutions to CT segmentation

As already mentioned in the previous section, lack of data and, particularly, the expert-annotated data is one of the problems in the field of medical image segmentation. In this work, we set one baseline method inspired by the current medical segmentation models standards and then compare it to another two experimental approaches. These two approaches are based on unsupervised transfer learning and user interaction. The proposed model scheme and its inputs are shown in Figure 1.

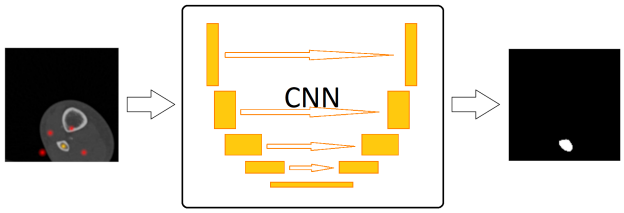



**Figure 1.** The U-net type model architecture with input and output patch. Each yellow block contains several layers consisting of convolution and ReLU. The orange arrows represent concatenation of the left block output with the right block input. Model input consists of only 1 channel of data for the baseline and the pretrained solutions. For the interactive solution, two other input channels are added (yellow color and red color in the input patch).

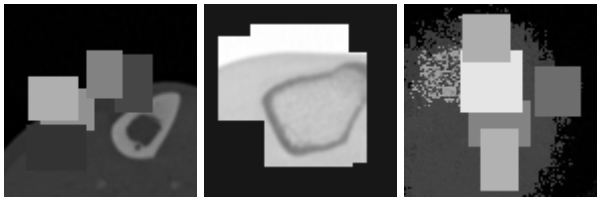



**Figure 2.** Three different training patches for the image restauration task. Inpainting (2a), Outpainting + intensity transformation (2b) and Inpainting and pixel shuffling (2c).

### 2.1 The baseline and the training process

The network architecture itself remains unchanged from 2D U-net [6] in the terms of level count and number of convolution filters in each convolution layer. Each covolution layer is followed by ReLU or Sigmoid (last layer) nonlinearity. In the training process, the network input is a $96 \times 96$ 2D patch randomly cut out of the CT data slice and corresponding groundtruth segmentation slice is used as a loss function input. Slices in each minibatch are chosen by random generator without repetition and the minibatch size is set to 32 slices. The generator is reset only after all the data has been used. We use cross entropy loss function and the Adam optimizer with learning step set to 1e-5.

We do not use the concept of epochal training because different models have been trained on different number of patients which would lead to unequally long epochs. Rather than that, we use a number of iterations for comparing the training processes of different models.

We are aware of the possibility of using regularization techniques to avoid problems with overfitting, but in this work we chose not to use them just yet, because we wanted to see the unaltered impact of different selected approaches.

The baseline model was proposed as a representation of a basic version of modern deep learning segmentational standard and serves as a reference point for comparing the efficiency of the two solutions for battling the lack of training data problem. The training process remains the same for the other two model versions if not said otherwise in the next two subsections.

### 2.2 Unsupervised pretraining

Inspired by the authors of Models Genesis [8], we use transfer learning as a means to improve the segmentation quality of a model trained on small amount of labeled data. At first, the model is trained on a secondary task of image restauration. The distortion methods are the same as in the original paper [8], specifically in- and out-painting, nonlinear intensity changes and local pixel shuffling. We use pregenerated pixel-shuffled data instead of using online generation, which we chose to do to reduce time consumption of the training process. The input of the model is a distorted patch and original patch is used as a label for simple L1 loss function. Please note that in this way the training process does not need any expert created annotations and is completely unsupervised. After the pretraining process, model can be trained on the segmentation task as described in the baseline subsection. Illustration of the restauration model data is shown in the Figure 2.

The benefit of this solution lies in the possibility of training on a big amount of unlabeled data, which should help the model learn the basic data structure resulting in increasing the segmentation quality even if trained only on a smaller amount of labeled data.

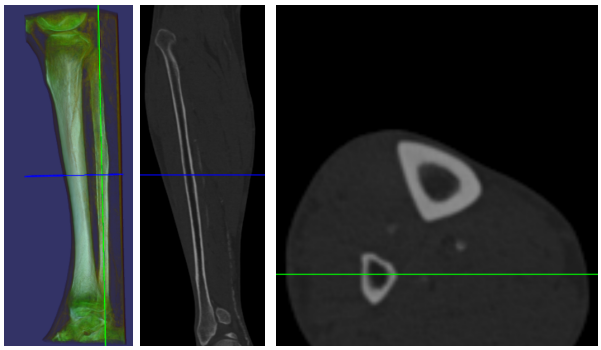
### 2.3 Adding user interaction

Allowing the possibility of user interaction in the final segmentation application, we can consider this approach to be another way to battle the lack of data issue. Same as Sakinis et al. [11], we use background and foreground click to specify the desired object within the input slice. We use the same method for creating the click maps as in the original paper which is as follows. Two click maps are created, one for object and one for background and both are then concatenated with the input data slice, creating a model input with 3 channels. When creating the clicks maps, pixels of a value 1 are placed to the desired click positions and whole map is then smoothed with Gaussian filter and normalized to range $\langle 0, 1 \rangle$.

In each training iteration, the model works in prediction mode at first and several user clicks are generated from the groundtruth data as well as from the

previous interaction output. First click is placed in the innermost position of the groundtruth object area and following clicks are then placed in the biggest erroneous areas. Final number of interactions is randomly set to $t \in \langle 1,5 \rangle$ for each slice in the minibatch. Unlike the original paper, we did not use multilabel data for the training. Because of that, we randomly convert the groundtruth to all-background and in such a case we only provide the background clicks. We hope that this will force the model to pay more attention to user clicks.

This solution was originally proposed to create more general model which should be able to segment previously unseen data, as suggested by the authors. Yielding good results, we investigate to what degree can this solution help to improve the results of a segmentation trained on only small amount of training data.



(a) (b) (c)

**Figure 3.** Example of the fibula bone and the surrounding tissue. 3D visualisation from the lateral view (3a), sagittal slice (3b) and axial slice (3c).

### 2.4 Dataset

All the models have been trained on a fibula subset of a fairly large dataset of human body CT scans provided by the TESCAN 3Dim company. The expert groundtruth segmentation was provided as well. This subset consists of CT scans of fibula bone and its surroundings (Figure 3) from 95 patients. Up to 70 patients data was used for the training, other 9 patients data was used for validation during the training and another 10 patients data was used for evaluation of the final models. Note that the evaluation data was not used during the training process nor as the checkpoint selection criteria.

The data in Hounsfield units was clipped to range $\langle -1024, 3000 \rangle$ and then scaled within the range of $\langle 0, 1 \rangle$. Otherwise no changes or augmentations were done to this data.

## 3. Experiments with varying dataset sizes

For the best demonstration of the impact of each of the selected solutions, we created three groups of models, one for each method and one for the baseline. Each group consists of 14 models which uses the following numbers of patients for the training process: 1, 2, 3, 4, 5, 10, 15, 20, 25, 30, 40, 50, 60, 70. The naming convention for the models in this work is as follows. Three different names are used (Seg, SegPretrained and SegInter) for defining the models group. The following number identifies the number of patients used for the model training. Model named Seg1, for example, is a model that was trained on one patient and belongs to the baseline group.

The Dice score on validation data is being recorded during the training process. This metric is defined by Equation 1, where $Gt_i$ is the ground truth segmentation and $P_i$ is the prediction for the $i$-th pixel. The model checkpoint with the best validation Dice score is always saved and is considered to be the final model after the last iteration ends. All models groups were trained for 65 000 iterations of the segmentation training process.

$$Dice = \frac{2\sum_i^N Gt_i \cdot P_i}{\sum_i^N Gt_i^2 + \sum_i^N P_i^2} \tag{1}$$

### 3.1 The baseline models

One group of the models has been trained with the baseline method for 65 000 iterations. The graph of convergence (Fig. 4a) suggests that further Dice score improvement is possible even after the 65 000 iterations. The hard limit of 65 000 iterations has been set due to the computing grid time limits. Even though there is a possibility of a further improvement, we believe that models trained this way are sufficient to illustrate the results of our experiments. As expected, the models with fewer training patients reach much lower Dice score than the model trained on the maximum of 70 patients. It is also possible to observe the trend of decreasing impact of adding more data with higher patients counts.

### 3.2 The secondary task of image restauration

The image restauration task itself is an interesting experiment. The restauration model was trained for 43 000 (checkpoint 1) and 104 000 (checkpoint 2) iterations on the 70 patients data. Both checkpoints were compared and it is noticeable that the second checkpoint model is surprisingly good in restoring the

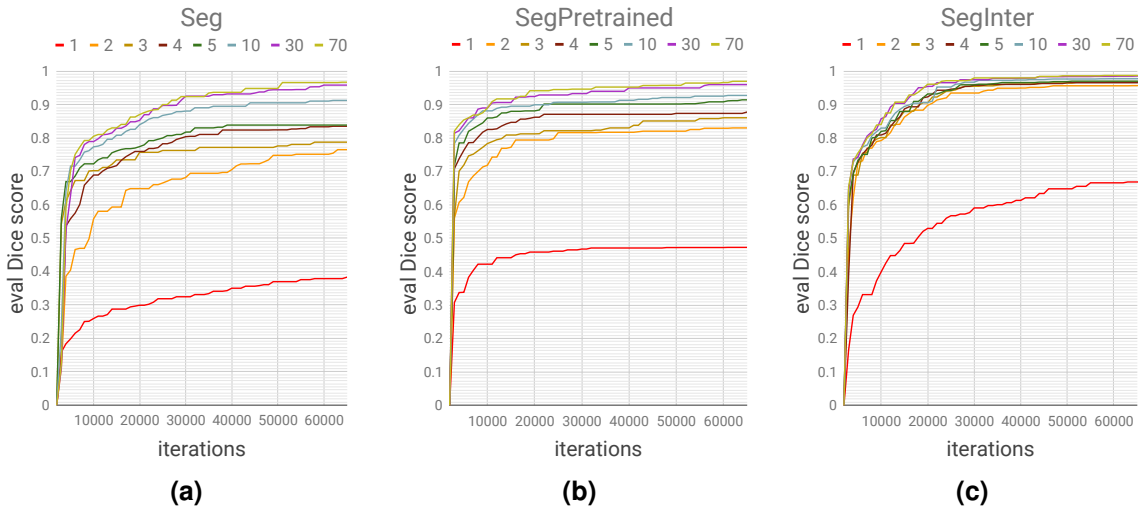



**Figure 4.** The comparison of the training process of the different models in the three groups. The graphs shows the development of the best Dice score reached on the validation data during the training process.

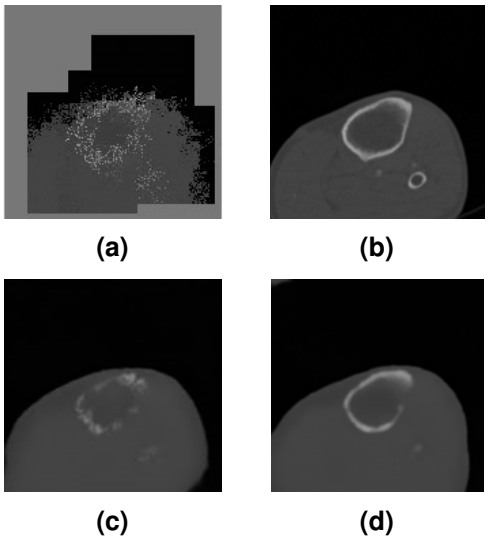



**Figure 5.** The secondary task of image restauration. Distorted input (5a), original lateral slice (5b), restauration model output after 43 000 iterations (5c) and after 104 000 iterations (5d).

bone details (Figure 5d). After the pretraining, SegPretrained models are trained the same way as the baseline group.

### 3.3 Interactive training process

As mentioned above, the interactive training process was implemented as in the work of Sakinis et al. [11]. However, there might be some changes needed in the future. The simulated interaction training process should mimic the way an actual user would click to improve the segmentation results, but in cases where the model reaches almost perfect results (mostly in later iterations), the simulated clicks degrade to clicking on very small areas of only few pixels. It is most likely that a real user would not place any guidance clicks on such areas and training the model to expect such a behavior might be inefficient. During the training process, there is maximum of 5 interactions for the training data, but only one click is used in the case of validation data and also during the evaluation of the final model.

## 4. Evaluation

For each model, the checkpoint with the best evaluation Dice score is considered to be the final one. Each of the final models in all the three groups has been tested on the evaluation data. The average Dice score for chosen models is shown in the Table 1.

Please note that there is always at least one user click present when evaluation the interactive model, which gives it a major advantage in comparison to the baseline. This is not considered a problem for the model comparison as we are researching the benefit of adding the user interaction.

### 4.1 Testing the final models

The numerical results on the testing data suggest that there is certain benefit in using either the unsupervised pretraining or the interactive approach. The most visible difference is, as expected, between the models with fewer number of training patients. In the case of the pretrained models, this is also caused by the fact that the difference between the pretraining data and the training data amount is increasing with the decreasing number of training patients, as there is only 70 patients data used for the pretraining iteself. The first noticeable thing about the training process is the

**Table 1.** Table of average dice reached by some of the models in the three groups (Seg, SegPretrained, SegInter) on the testing data. Patients column determines the number of training patients for particular model in the group.

| Patients | Seg | SegPretrained | SegInter |
|---|---|---|---|
| 1 | 0.348 | 0.392 | **0.588** |
| 2 | 0.749 | 0.830 | **0.929** |
| 3 | 0.795 | 0.855 | **0.949** |
| 5 | 0.819 | 0.858 | **0.943** |
| 10 | 0.831 | 0.904 | **0.948** |
| 30 | 0.974 | 0.977 | 0.977 |
| 70 | 0.981 | **0.984** | 0.983 |

speed up of the models convergence (Figure 4c). It seems that pretrained models reach higher Dice score on the validation data than the baseline models, especially in the scenarios with smaller amount of labeled data. The convergence speed up might bias the evaluation of the final model but although the baseline method might still improve a little in case of longer optimization, the trend is clear. Pretraining the model on larger unannotated data, possibly with adding whole human body data, would be an interesting extension to this experiment and might be examined in the future. The interactive approach yields slightly better results, which is most likely thanks to better localization of the segmented object, as described in the next subsection.

### 4.2 The benefits and the limitations of interactive training with one class

The visual examination of the outputs of different models reveals that interactive training helps the model with locating the correct object, which is shown in Figure 6. Even though it improves results of the model trained on the one patient data, it is apparent that model is still relying on the known bone shape rather than on the user clicks, at least to some extent. In some cases, clicking on the false positive areas does not help the model ignore them. It is most likely that the model just learned to either use or not use the filters trained to do the particular bone segmentation and it would probably require training on a mixed bone dataset and/or dataset with multiple classes to improve the utilization of the filters.

## 5. Conclusions

We compared the three possible solutions to bode segmentation in CT data with respect to the lack of data problem. The baseline solution was a rather traditional approach to the medical data segmentation, using the

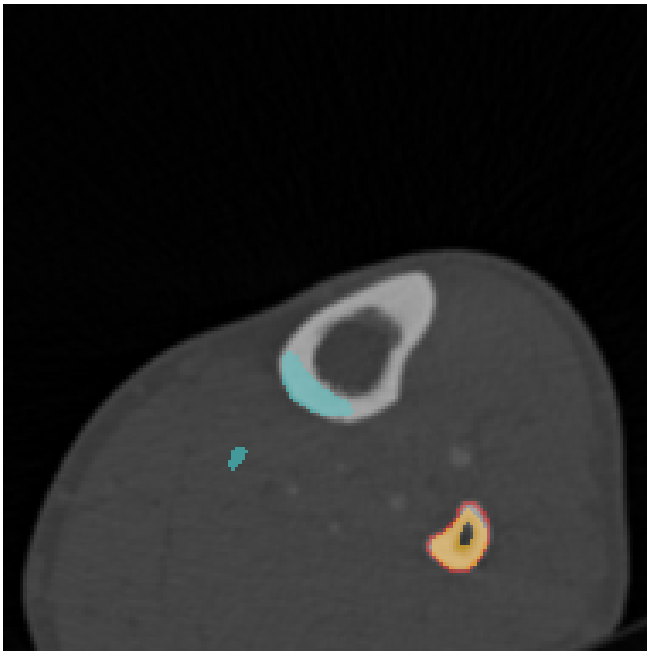


**Figure 6.** Comparison of SegInter1 and Seg1 models output. In this case, the interactive model (orange) trained on just one patient has much better result than the baseline (blue) model. Seg1 completely fails to identify the correct bone. (The ground truth segmentation is delineated with red color.)

unpretrained U-net type architecture with training pairs consisting of 2D input patch and the corresponding ground truth segmentation patch. The two other solutions were chosen to battle the problem of the lack of expert-created annotation in the medical field. The first solution uses the unsupervised pretraining on a secondary task of image restauration. The second solution adds the possibility of a user interaction in form of click maps. We trained three groups of models, one for each solution. Each group consisted of 14 models with varying number of training patients, to illustrate the efficiency of each solution. The results on the testing data suggests that both of the solutions provide certain benefit.

The best improvement was between the model Seg1 and SegInter1. Both models were trained only on one training patient, but the interactive model test Dice score was 0.24 better than the baseline model. The interactive model trained on two patients already yields promising results with test Dice score 0.929. This was achieved while using only one user interaction per slice, which could be later replaced by drawing only one line in the lateral view.

Both of the selected solutions provide some degree of improvement to the lack of data problem. The use of the interactive approach could, once implemented, lessen the workload for medical experts by being the

mean to reducing the necessary amount of training data. For data with a complexity of long bones such as fibula, which has both left and right variant, we only need to create manual segmentation for two bones (left and right) to obtain relatively good results. This can provide a significant boost to the creation of segmentation dataset for new types of data.

Even though we already have some promising results, the final method should include some improvements, such as using regularization methods, replacing the loss function with Diceloss or converting the whole model to 3D. The immediate future work will, however, lie mostly in the improving the interactive solution or combining both of the solutions to see if this brings some improvement. One of the possible improvements of the interactive model is adding the model output from the previous iteration/interaction as another channel to the next model input.

## Acknowledgements

I would like to thank Oldřich Kodym for contributing to this work with his knowledge from the deep learning field as well as for his patience as my supervisor. I would also like to thank the TESCAN 3Dim company for providing the CT datasets.

Computational resources were supplied by the project "e-Infrastruktura CZ" (e-INFRA LM2018140) provided within the program Projects of Large Research, Development and Innovations Infrastructures.

## References

[1] Y. Y. Boykov and M. . Jolly. Interactive graph cuts for optimal boundary region segmentation of objects in n-d images. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, volume 1, pages 105–112 vol.1, July 2001.

[2] Oldřich Kodym, Michal Španěl, and Adam Herout. Segmentation of defective skulls from ct data for tissue modelling, 2019.

[3] Evan Shelhamer, Jonathon Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39:1–1, 05 2016.

[4] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation, 2015.

[5] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *Computer Vision (ICCV), 2015 IEEE International Conference on*, 2015.

[6] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. 2015.

[7] Özgün Çiçek, Ahmed Abdulkadir, Soeren S. Lienkamp, Thomas Brox, and Olaf Ronneberger. 3d u-net: Learning dense volumetric segmentation from sparse annotation, 2016.

[8] Zongwei Zhou, Vatsal Sodha, Md Mahfuzur Rahman Siddiquee, Ruibin Feng, Nima Tajbakhsh, Michael Gotway, and Jianming Liang. *Models Genesis: Generic Autodidactic Models for 3D Medical Image Analysis*, pages 384–393. 10 2019.

[9] Ning Xu, Brian Price, Scott Cohen, Jimei Yang, and Thomas Huang. Deep interactive object selection, 2016.

[10] G. Wang, W. Li, M. A. Zuluaga, R. Pratt, P. A. Patel, M. Aertsen, T. Doel, A. L. David, J. Deprest, S. Ourselin, and T. Vercauteren. Interactive medical image segmentation using deep learning with image-specific fine tuning. *IEEE Transactions on Medical Imaging*, 37(7):1562–1573, July 2018.

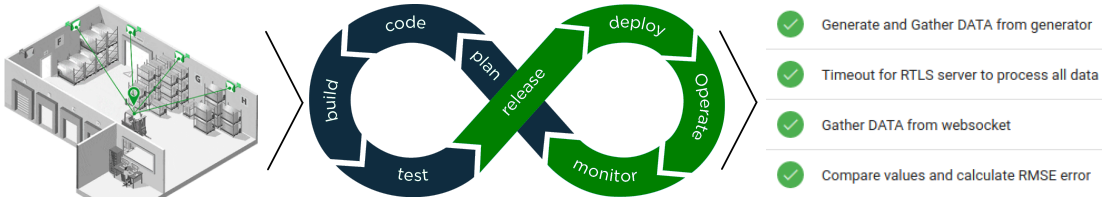
[11] Tomas Sakinis, Fausto Milletari, Holger Roth, Panagiotis Korfiatis, Petro Kostandy, Kenneth Philbrick, Zeynettin Akkus, Ziyue Xu, Daguang Xu, and Bradley J. Erickson. Interactive segmentation of medical images through fully convolutional neural networks, 2019.

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

# Infrastructure for Testing and Deployment of the Real-Time Localization Platform

Michal Ormoš*




**Abstract**
Fast development and deployment of the software are the new phenomena of the era. It is not different in the field of real-time localization systems. In our global world where the global positioning system (GPS) is the everyday utility, there is a necessity of localizing under the roof where the GPS cannot access. Here come the local position systems based on Ultra Wideband, which bring the ultimate precision. This work solves the problem of fast delivery of the software responsible for the real-time localization systems. It produces a case study on how to develop, test, and deploy this system in the continuous integration and delivery environment with the help of DevOps principles. This requires introducing the new techniques and methods for how to validate and test the precision of these systems. With these improvements, we can deliver this type of software faster by reducing the time needed for testing and validate software as it is developed. Also, we can guarantee and demonstrate it's quality across versions easier.

**Keywords:** RTLS systems — UWB — CI/CD — Indoor Localization

**Supplementary Material:** *N/A*

*xormos00@stud.fit.vutbr.cz / mi.ormos@gmail.com, *Faculty of Information Technology, Brno University of Technology*

## 1. Introduction

In recent years I have been focusing my attention on indoor localization systems and products for solving the problem of indoor asset localization. I am part of the team developing the real-time localization systems (RTLS), and I was not always satisfied with how the software part of this system was tested and delivered. Recently, I started to think about how to improve this process with conventional testing methods and also anew developed methods only for this type of software. With the help of the crucial test-suite for the RTLS system, we can deliver the new features faster and more securely than before.

A real-time location system, as the name characterizes are systems requiring a zero-delay response to their inputs. RTLS system is a system based on familiar software architecture with a database for storing the data, back-end for serving the data, and front-end for presenting the data. However, in the heart of every RTLS system is location algorithm. At input of this algorithm are thousands of data streams from location hardware. Streams of this data, in raw Ultra Wideband (UWB) patterns, goes through the entire system. Data are parsed, calculated, optimized and shown in an understandable format. This data can be used for example in logistic applications, navigation and asset tracking. Guarantee that this data will be calculated as fast as possible in the correct way is crucial. As this system is growing and optimizing, we have to ensure that this will be preserved across the new releases.

Indoor localization based on UWB is quite a new concept in early development and deployment in the industry [1, 2, 3]. However, testing and validating the software products is old as software development itself. In half of the problems, we can easily apply conventional tools with little adjustments. However, the other half requires new methods and the dose of creativity. The ideas of this infrastructure come from the DevOps principles [4, 5]. It is more the right philosophy than the right tool. It is combination of the word's development and operations. DevOps assimilates development and operations teams to improve the collaboration process. A DevOps Engineer will work with IT developers to facilitate better coordination among operations, development, and testing functions by automating the integration and deployment processes. Continuous integration and continuous delivery (CI/CD) is often referred to as pillars of successful DevOps [6, 5]. To establish and optimize the CI/CD development model and receive the benefits, companies need to build an active pipeline to automate their build, integration, and testing processes.

The second half of the problem is coping with the difficulty of how to abstract this system from their hardware components. As we want to test fast, remotely, and in a stable environment, we have to abstract hardware in a software manner. All RTLS systems are dependent on devices which are transmitting the signal as RTLS Tags or merely mobile devices as well as devices receiving the signal, called RTLS Anchors [1, 7]. These devices record the signal representation of the localization data where the RTLS software then transforms this data to Cartesian coordinates system understandable by everybody. The unique part of this work will be to abstract this hardware credible so we can test the software without it as we have it.

The outcome of this work is test suite, which accelerates the delivery of the new version of the RTLS system and ensure safer and more secure development with confidence in the delivery process.

## 2. Background

### 2.1 Indoor Localization

A local positioning system (LPS) is a navigation system that provides location information anywhere within the coverage of the network in all conditions. If there is an unobstructed line of sight to three or more signaling devices of which the exact position on the place is known. A particular type of LPS is the real-time locating system (RTLS), which also allows real-time tracking of an object or a person in an enclosed area such as a buildings and factories [8, 9].

Ultra Wideband (UWB or ultraband) is any radio technology that has bandwidth exceeding 500 MHz or 20 percent of the arithmetic center frequency, whichever is lower. UWB is a carrierless communication scheme. The early applications of UWB technology were primarily related to a radar. A UWB-based locating system is very much like any other RTLS except that it uses UWB signals [4].

### 2.2 CI/CD Pipeline

A CI/CD pipeline helps you automate steps in your software delivery process, such as initiating code builds, running automated tests and deploying to a staging or production environment. Automated pipelines eliminate manual errors, provide standardized development feedback loops and enable fast product iterations [10].

### 2.3 Implementation

Software testing is the most frequently used method for verifying and validating the quality of the software. Testing is the method of executing a program or system to detect faults. It is a significant activity of the software development life cycle. It helps in developing the confidence of a developer that a program does what it is intended to do.

## 3. Implementation

RTLS system consists of hardware and software parts. In this test-suite, we only aim for the software parts and omit the hardware parts. But as the software is dependent on the hardware inputs, we will create the hardware abstraction for software testing. This will make testing and development faster and more easygoing.

RTLS system software consist of RTLS Manager application which is communicating with the hardware and is parsing the input data. RTLS Server application for calculating the positions and RTLS Sensmap for displaying the localization data. This all is back-up with the database.

The next section deals with the individual parts of the test-suite as we will call our final pipeline containing all test cases.

### 3.1 REST Tests

Many third party applications are connecting to the RTLS system and are fetching the data. The most convenient method to fetch this data is WebSockets and Open REST API. In this section, we aim for the REST API. This API consists of many requests for getting the data from the database. The test-suite aims for all these calls (GET, POST, PUT and DELETE). It

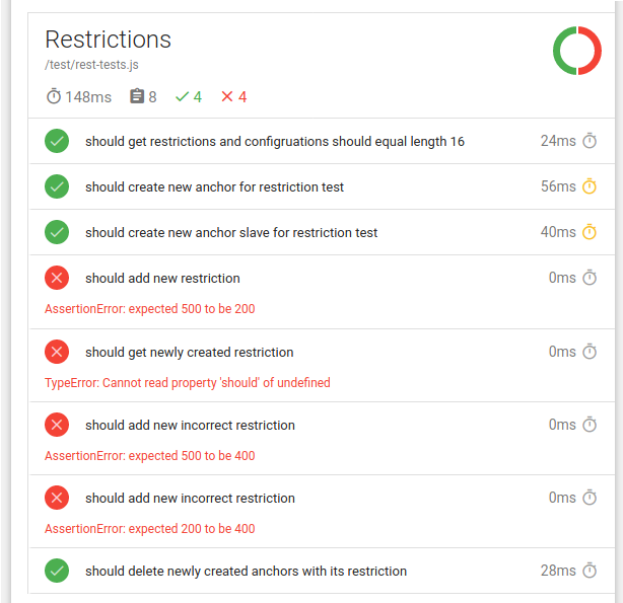



**Figure 1.** Example of an invalid test from the REST Tests section of our test-suite pipeline. Individual test cases are self-explaining and conditions are described.

tests the positive and negative results of these requests. By these tests, we can be sure the RTLS system always responds to the API call as it was designed to. The part of testing WebSockets will be mentioned in the next sections.

### 3.2 Syntax Checkers

RTLS systems usually have many contributors within the team. DevOps principles emphasize that the same tools and the same principles have to be used across the team to guarantee the quality and validity of the software. By this principle, we introduce the linters. Lint, or a linter, is a tool that analyzes source code to flag programming errors, bugs, stylistic errors and suspicious constructs. The team settles on the shared rules for writing the source code and these rules will be forced in the pipeline. If the developer do not obey this rules, the CI/CD pipeline will reject his commit.

### 3.3 Unit Tests

Unit tests are the first level of the testing process. They are validating the individual parts of every module. The unit test is written in a white box method, where the developer writes the tests based on information gathered from the source code. From this point the test-suite will be in a black-box method where the tests are written only with the specification of what the software should do and without the information from the source code.

### 3.4 Performance Tests

An essential part of our test-suite will also be to find out where are the performance limits of the software implementation depending on the hardware organizations. The limitation of hardware parts can be easily calculated by their design and theoretical limits of the UWB. Nevertheless, as always, the software is usually built together with many third-party applications and running on different operating system distributions, so a real performance test under the load has to be performed. Our performance tests will be included within the REST Tests as well as our Report generator tests.

### 3.5 pcap Player

This is the first unique test case built for the RTLS system. As RTLS systems are already built and deployed in the installations around the globe, we can use this in our benefit. In the first way of application testing we take the network recordings of the real application. Then we eliminate the packets that are not connected to the localization itself. After that, we replay this recording to the RTLS system.

But before that, we need to take the original system database and settings, and import them to the system where we are replaying. Thereby we can replay the same scenario to the system again and again. By this method we can ensure that the system works the same across the distribution and we can easily tune the system remotely.

### 3.6 Report Generator

In the pcap Player, we were limited to the same payload which we could send to the RTLS system. In this part, we create the generator of our unique payload. This will give us the benefit to create our own scenarios. Hereafter we mention RLTS Tags as tags and RTLS Anchors as anchors.

Tags are sending the signal to the environment and Anchors are receiving them. Tags are dynamic devices in the movement and Anchors are static devices with a predefined position. The most important part of Anchors is the synchronization between them. The RTLS system is receiving the RTLS Blink signal from Anchors about the tag signal time of flight from Tag to the Anchor, and also the RTLS Sync signal about the synchronization between the Anchors. By learning how this Blinks and Syncs are created and successfully generating them we can create authentic RTLS payload generator.

To have full control of creating UWB payload by the software we can now create random scenarios that suit our purpose. Moreover, most localization errors originate in synchronization inaccuracies. By that we

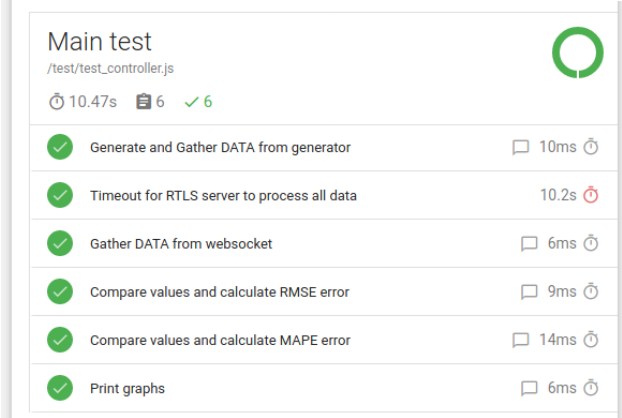



**Figure 2.** Example of a valid report generator test in our test-suite. Passing on all items, which are self-explaining.

mean that we can put intended errors to our synchronization data on purpose. Thereby we can observe how the system and his filters cope with the errors and use some statistical methods as RMSE or MAPE to fully validate this scenarios.

### 3.7 CI/CD integration

To make this pipeline work, the final step is to put it to the system code repository where the project is developed and ensure this pipeline will run on every significant change of the code in the repository system to preserve system quality and validity. We decided to use the GitLab repository system, where the codes are already stored. GitLab offers CI/CD built in every repository based on GitLab runners, an open-source project that is used to run your jobs and send the results back to GitLab. It allows you to set different scenarios when the test should run, as for example ours at every commit or every midnight.

## 4. Evaluation

In this short evaluation, we introduce the final output of the pipeline and introduce a few methods of how can we validate it. In REST Tests, Unit Tests and Syntax Checker we are validating results pretty straightforwardly. We will specify limits and values when the test passes or fails.

This was already shown in Figure 1 and Figure 2. We set conditions in which individual tests success or fail, and the output of our test pipeline results in these conditions passing or failing based on data they received.

But this is not so straightforward in pcap Player and Report Generator. In Figure 3 we can see the output of the report generator as we plot the data to the plan. The orange line representing the data generated by the report generator. This data are transformed into data similar to UWB traffic, as Anchors would generate them and passed to the RTLS software on its input. The green line represents this UWB traffic passed through RTLS system and positions are calculated. By this simple demonstration, we can assume our report generator is working as we design to. This is suitable for manual validation but unacceptable by pipeline.

One way we can address this is to use statistical methods as Root-mean-square deviation (RMSE equation 1) and Mean absolute percentage error (MAPE equation 2) and calculate the deviation of points generated compared to points gathered as shown in the Figure 4. Variable $\overline{y}$ in both equations represents predicted values, variable $y$, observed value. Variable $n$ in both equations means number of observations.

$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(\overline{y}_i - y_i)^2} \tag{1}$$

$$MAPE = \frac{100\%}{n}\sum_{t=1}^{n}\left|\frac{\overline{y}_i - y_i}{y_i}\right| \tag{2}$$

Calculated MAPE and RMSE of our evaluation case can be seen in Table 1. Here we can see deviation of x-axis ans y-axis respectively. Deviation is calculated between the every pair of points as shown in Figure 4. Orange points are from report generator and greens points are from RTLS system localization output.

| | Values | |
|---|---|---|
| Method | x-axis | y-axis |
| MAPE | 2.57% | 1.05% |
| RMSE | 6.12801 | 1.41575 |

**Table 1.** Localization accuracy.

By this approach, we can determine the boundaries of deviation where the localization is acceptable and when unacceptable.

## 5. Conclusions

We introduce an indoor localization system based on the UWB. We create test-suite based on CI/CD and DevOps principles. This test-suite is specially designed for the RTLS systems. It consists of commercially used tools as well as our proprietary tools designed and created for testing RTLS systems without the necessity of real hardware components.

By applying this pipeline, we reduce delivery time of the new RTLS system version. As this systems are delicate and their real-time principles are crucial for

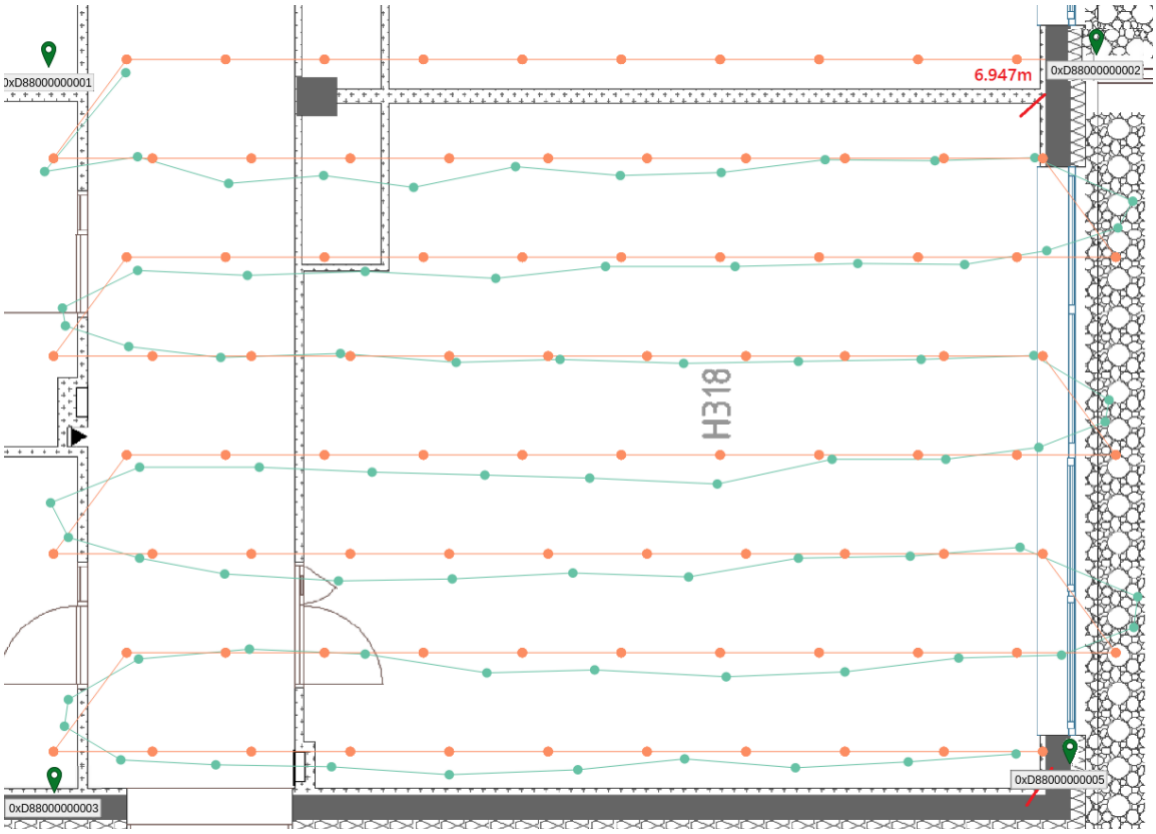



**Figure 3.** The demonstrative representation of the report generator. The original trajectory from the report generator (orange) with output trajectory from RTLS system (green). Localization is never a straight line due to many approximation filters algorithms of RTLS system use. The localization movement starts from the top right corner and going from side to side until finishing in the bottom right corner.

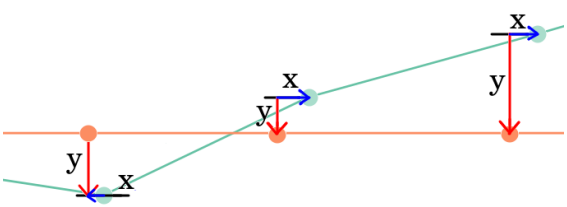



**Figure 4.** Example of how we use RMSE and MAPE for automatic pipeline validation for deviation in both axis of localization.

logistic and business applications, we can now deliver this software more securely and with validation that everything works as it should.

This paper is a case study of RTLS systems and describes how to deliver them quickly and safely. It analyzes this new rising technology and necessity to introduce the way it can develop and deliver with confidence.

This pipeline can be easily adjusted for different RTLS systems in the future. But for this particular UWB technology it could for the future include of more statistical metrics for validation, as well as front-end test cases for even faster CI/CD process.

## Acknowledgements

I would like to thank my supervisor Ing. Vladimír Veselý Ph.D., who helped me by leading this thesis through the academic environment. I also want to express my thanks to employees of Sewio Networks s.r.o, who helped me with their knowledge about the RTLS system I worked on, and allow me to create a thesis on that exciting and rising topic.

## References

[1] Len Bass. An evaluation of indoor location determination technologies. *IEEE Software*, 35(1):8–10, 2018.

[2] Fouzia Boukour Elbahhar and Atika Rivenq. *New Approach of Indoor and Outdoor Localization Systems*. InTech, 1 edition, 2012.

[3] N. Oster F. Saglietti and F. Pinte. White and grey-box verification and validation approaches for safety- and security-critical software systems. *Information Security Technical Report*, 13:10–16, 02 2008.

[4] L. Bass. The software architect and devops. *IEEE Software*, 35(1):8–10, January 2018.

[5] Joost Evertse. *Mastering GitLab 12: Implement DevOps culture and repository management solutions*. Packt Publishing, 1 edition, 2019.

[6] Adam Debbiche, Mikael Dienér, and Richard Berntsson Svensson. Challenges when adopting continuous integration:: A case study. volume 8892, pages 17–32. Springer, Cham, 2014.

[7] Mengda Wang, Bing Xue, Wei Wang, and Junjie Yang. The design of multi-user indoor uwb localization system. In *2017 2nd International Conference on Frontiers of Sensors Technologies (ICFST)*, volume 2017-, pages 322–326. IEEE, 2017.

[8] Johan Hjelm Krzysztof W. Kolodziej. *Local Positioning Systems LBS Applications and Services*. Taylor Francis Group, LLC, 1 edition, 2006.

[9] Sewio public documentation. online, 2019.

[10] Manish Virmani. Understanding devops bridging the gap from continuous integration to continuous delivery. pages 78–82, 2015.

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

# Framework pro tvorbu informačních systémů nad Universal Windows Platform (ISUF)

Jan Rajnoha*

**Abstrakt**
Informační systém je v dnešní době jeden z nejpoužívanějších typů aplikace a je potřeba zjednodušovat jejich návrhy a realizaci pomocí frameworků, kterých existuje celá řada, ale zatím neexistuje framework, který by byl schopný pracovat nad Windows 10 a technologií Universal Windows Platform.
V tomto článku se zaměřím právě na návrh frameworku pro Windows 10 s požadavky na jednoduchost tvorby jednotlivých modulů, aktualizaci a automatickou tvorbu generovaných formulářů. Součástí řešení je i ukázková aplikace, která prezentuje možnosti frameworku jednoduchou formou včetně zdrojových kódů.
Windows 10 je v aktuální době nejrozšířenější operační systém a nabízí velmi širokou škálu zařízení a je tedy výhodné směřovat právě na něj.

**Klíčová slova:** Framework — Informační systém — Windows 10 — UWP — Aplikační design

**Přiložené materiály:** GitHub repozitář řešení — Ukázková aplikace

*xrajno09@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

## 1. Úvod

Frameworky a informační systémy jsou nedílnou součástí dnešních rutin, obzvláště pak ve firemní nebo studijní sféře. Existuje pouze malé procento aplikací a programů, které jsou vytvořeny "na zelené louce" bez jakékoliv podpory nějakého frameworku a, v případě stavby aplikace nad nějakým operačním systémem, je to už skoro nemožné.

Pokud se podíváme na oblast informačních systémů, tak většina programů je v dnešní době právě z této kategorie. Na fakultě informačních technologií v Brně se na denní bázi používá WIS, který slouží jak pro studenty, tak pro zaměstnance. VUT v Brně používá Apollo, vývojáři například Azure DevOps na verzování kódu nebo ekonomové např. program Pohoda

Dalším velkým oborem v informačních technologiích, na který je potřeba dbát, jsou operační systémy, na který informační systém cílíme. Posledních pár let se rozšiřuje operační systém Windows 10 od společnosti Microsoft. Pro vývojáře je to velmi pohodlný systém obzvláště v oblasti distribuce software pro Windows a jejich aktualizací díky Microsoft Store, který zároveň nabízí i široké možnosti testování mezi uživateli a distribuci doplňujících balíčků.

Windows 10 také nabízí vlastní prostředí nazvané Windows Universal Platform (UWP), jenž díky frameworku .NET Native [1], na kterém když vytvoříme aplikaci, je spustitelné na všech zařízeních běžících na tomto operačním systému. Další výhodou využití Windows 10 je jeho rozšíření, kdy v aktuální době tento operační systém používá více jak 1 miliarda zařízení

[2].

V současné době neexistuje framework, který by se zaměřoval právě na kombinaci těchto aspektů, a z toho důvodu jsem se rozhodl vytvořit vlastní, jenž kombinuje vlastnosti těchto oblastí a nabízí velmi jednoduchý způsob tvorby aplikací, jejich distribuce a aktualizace a přenositelnosti mezi zařízeními právě s Windows 10. Při návrhu bylo dbáno na co nejjednodušší možnost vytvoření funkční aplikace, ať už se jedná o tvorbu modulů, formulářů, správu nastavení a obecně dalšího chování aplikace, kterou je poté možné dále upravovat k vlastním potřebám. Framework nese název "ISUF", v delším znění pak "Framework pro informační systémy nad Windows 10".

## 2. Požadavky na framework nad UWP

Když se řekne framework, spousta lidí se představí nějakou knihovnu s množstvím obecných funkcionalit zaměřené na danou kategorii. Stejné je to i v našem případě frameworku pro informační systém, kde se očekává řada nástrojů pro jednoduchý návrh a možnou přizpůsobitelnost stávajícího řešení, ať už jde o rozšíření nebo upravení celé funkcionality.

V prvé řadě je potřeba nabídnout vývojáři možnost pracovat odděleně a nezávisle na ostatních částech, aby výsledný produkt nenabobtnal funkcemi, které v závěru ani nevyužijeme. Dále je potřeba oddělit celé řešení na jednotlivé části i z důvodu jejich zaměření. Ne vždy totiž potřebujeme například část pro tvorbu grafického rozhraní nebo pro přístup do úložiště.

Stejně tak je potřeba se zamyslet nad vytvářením jednotlivých modulů, práce s nimi, propojování mezi sebou a jejich zapojení do systému a následném spravování na úrovni reprezentace v celém frameworku. Ne vždy chceme modul zobrazovat pomocí UI nebo všechny jeho data ukládat (používání vypočítaných hodnot). Některé moduly mohou být pouze informativní a jako celek se neukládají, protože se používají pro reprezentaci stavu aplikace.

Kromě rozdělení na samostatné části je nutné dbát i na samotné vývojáře a ulehčení jejich práce vhodnou volbou návrhu jednotlivých modelů, práce s atributy a úpravou vlivu modelu na chod aplikace, jako je např. schování položky ve formuláři nebo její ignorování při ukládání do databáze. Samozřejmostí je také možnost úpravy chování pomocí globálních konstant a nastavení aplikace, jejich ukládání a rozšiřování.

Zaměřit se také musíme i na oblast, na jakou cílíme. Jestli výsledný produkt bude spíše obecný nebo bude řešit jednu velmi specifickou úlohu. Jestli budeme v návrhu počítat s úpravami ku potřebám vývojáře nebo to bude čistě pevně dané řešení, které se upraví pouze o základní oblasti, jako jsou např. modely. Zde se vychází především z již existujících řešení, jako jsou např. informační systémy GTrade a ILTEGRO pro stejnojmenné firmy vyvinuté lidmi z IT Spektrum nebo řešení Microsoft Dynamics [3].

Dalším aspektem, jenž je důležitý pro výsledný produkt, je rychlost řešení, tedy optimalizace. Zde je potřeba správně navrhnout přístup do úložiště, načítání uživatelského rozhraní a celkovou náročnost řešení (množství provedených operací mezi požadavkem uživatele a odpovědí aplikace). Do této kategorie spadá i volba technologie, na které bude daný framework realizován. Jak již zaměření napovídá, vybral jsem technologii Universal Windows Platform pro Windows 10.

Jak jsem již ale zmínil, Windows 10 a prostředí UWP nabízí pro vývojáře velmi příjemné prostředí pro distribuci software a velmi povedenou optimalizaci práce aplikace se systémovými zdroji, ale v případě vývoje může být značně omezující, protože se snaží, aby aplikace vytvořené právě pomocí UWP technologie byly co nejbezpečnější, což v důsledku znamená možné nedostatky a omezení. Jedním z nich je například přístup k souborovému systému. Aplikace je ve výchozím stavu schopná pracovat ve svém adresáři nebo v Downloads složce, ale již nedokáže získat přístup třeba k základním uživatelským složkám, jako jsou Pictures, Music nebo třeba Videos a dalším složkám souborového systému. Další omezení přináší samotná technologie jako taková, protože je limitována pouze na operační systém Windows 10 a instalace balíčků je doporučována přes Store (nicméně lze instalovat aplikace i mimo něj), kde, pokud chceme publikovat, si musíme koupit licenci.

Když to shrneme a vytvoříme nad těmito body SWOT analýzu, můžeme vidět, že problémy při návrhu budou dělat značná omezení použité technologie. Na druhou stranu, unikátnost řešení a možnosti distribuce vyvažují celý další postup natolik, že výsledný produkt (v případě této práce pouze koncept) bude jedinečný a ukáže možnosti, které tato technologie má.

**SWOT analýza:**

Strengths - silné stránky:

- Distribuce aktualizací
- Jednoduchost řešení
- Možnosti přizpůsobení chování

Weaknesses - slabé stránky:

- Omezený přístupu k souborovému systému
- Omezené možnosti při vývoji
- Pouze Windows 10

Distribuce přes Microsoft Store vyžaduje licenci

Opportunities - příležitosti:
Jedinečný koncept v rámci Windows 10

Threats - hrozby:
Aktuálně nejasný vývoj UWP platformy

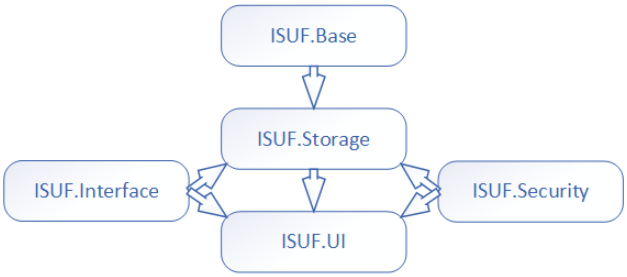
## 3. Dostupná řešení

Pokud se podíváme, jaká jsou dostupná řešení, tak v oblasti UWP zatím podobný produkt neexistuje a tím se tento koncept stává ještě více ojedinělým. V případě jiných desktop frameworků, založených na technologiích WinForms a WPF, již můžeme vybírat. Mezi známější patří například:

DevExpress eXpressApp Framework
Společnost Developer Express Inc. patří v dnešní době k jedním z největších dodavatelů různých vývojových komponent v oblasti vzhledu aplikací, ale i služeb. Jejich frameworky jsou cílené na spousty platforem a řešení. Jednou z těchto oblastí je i tvorba informačních systémů, kterou pokrývá právě eXpressApp Framework [4]. Tento nástroj je placený, ale nabízí i 30denní zkušební dobu. Distribuce je pomocí jednoho velkého balíku řešení (přes 600 prvků grafického rozhraní), který je nutné zakoupit jako celek a je tedy finančně náročnější, nicméně součástí je velmi podrobná dokumentace s postupným návodem, jak pracovat s daným frameworkem. Řešení podporuje tvorbu uživatelského rozhraní a nabízí podporu pro většinu specifik, jenž byly zmíněny v sekci "Požadavky na framework nad UWP" (Sekce 2).

MM .NET Framework
Framework společnosti Oak Leaf Enterprise, Inc. [5] nabízí řešení jak pro WinForms, tak pro WPF. Cena za jednoho vývojáře na rok není příliš vysoká, ale při porovnání s řešením od DevExpress je limitující, obzvláště pak v případě práce s databází a automaticky generovaných formulářů, které nepodporuje vůbec. Dokumentace k řešení je podrobná a na vyžádání je možné vyzkoušet zkušební verzi.

Vlastní řešení od A do Z
Další možností je také tvorba vlastního řešení z již existujících komponent při využití více či méně pomocných knihoven. Mezi tvůrci komponent je nutné zmínit společnost Syncfusion, Inc. [6] nebo Progress Software Corporation a jejich balíček Telerik [7]. Problémem vlastního řešení je ztráta všech specializovaných funkcí, jako je např. automaticky generované tabulky v databázi nebo generované formuláře. Na druhou stranu, dané řešení může být rychlejší, protože není zatížené obecnými funkcemi, které musí framework obsahovat.




**Obrázek 1.** Vazby mezi částmi ISUF frameworku

## 4. Návrh frameworku ISUF

Nejprve, než se podíváme na návrh řešení, vysvětlíme si základní pojmy, které se zde vyskytují.

- **Modul** - Jednotka aplikace. Obsahuje způsob, jak s ním aplikace pracuje a datovou třídu (entita).
- **Komponenta** - Část designu. Uživatelský prvek, z nichž se potom budou skládat jednotlivé view (zobrazení).
- **View, zobrazení** - Části okna, jenž využívají view modely.

Ještě jednou vyjmenuji požadavky na framework, které jsme si rozebrali v kapitole 2. a vyberu i oblasti, které jsem se rozhodl realizovat:

- Rozčleněné řešení do více samostatných částí
- Jednoduchá tvorba modulů s následnou tvorbou databáze
- Možnost propojovat moduly mezi sebou a ostatní úpravy
- Přednastavené moduly
- Automatické generování formulářů
- Obecný přístup, které si vývojář přepíše v případě nutnosti pro vlastní potřeby
- Optimalizace
- Využití možností Windows 10 a UWP technologie a možná přenositelnost na jiné platformy

Nyní se podíváme na nejdůležitější požadavky, které nejvíce ovlivňují následný vývoj celého řešení a které si zároveň i podrobněji rozebereme.

**Rozdělení do samostatných částí**
Při návrhu jsem se snažil dát vývojáři možnost pracovat pouze s nějakými částmi řešení. Pokud chce, může si na něm postavit vlastní, které již vyšší části

potřebovat nebude. Stejně tak může využít celý framework a upravovat jen dílčí chování a reakce na události. Další výhodou je také jednodušší aktualizování, kdy se nemusí výsledná aplikace distribuovat s velkým balíkem, ale pouze jen s upravenou částí.

### Jednoduchá tvorba modulů

Každý modul je tvořen velmi jednoduchým způsobem, jako samostatná třída, která musí implementovat základní rozhraní. Vše ostatní se poté řeší pomocí atributů. Každý datový typ sebou nese jistá omezení atributů, ale v základu má vývojář volnou ruku a podporovány jsou všechny základní datové typy. Jednotlivé moduly jsou registrovány a při vytváření spojení s databázi se kontroluje, zda jsou tabulky aktuální a není nutnost je aktualizovat. Tím má vývojář zjednodušenou práci, protože nemusí znát žádné parametry použité databáze, jen mu stačí znát přístup, nejlépe připojovací řetězec (connection string) a framework obstará všechny ostatní nutné kroky k nastavení spojení.

### Propojování modulů

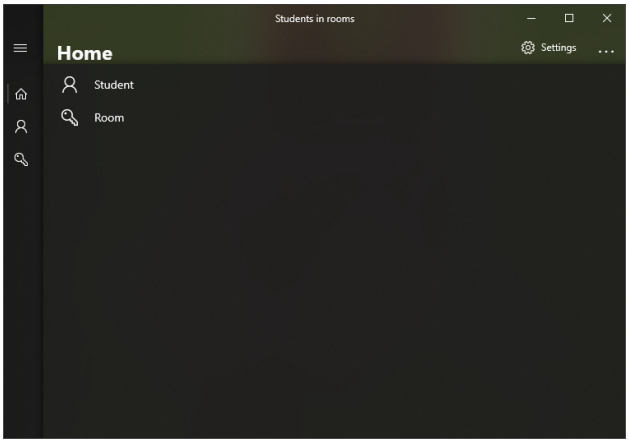
Jak jsem již zmiňoval, veškeré úpravy se tvoří pomocí atributů. Relace mezi tabulkami se pak dělají pomocí číselných datových typů a seznamu číselných datových typů, které určují vazbu 1:N. Pomocí atributu specifikujeme modul, ke kterému se daná vazba váže a uložená hodnota pak určuje ID jednotlivých propojených záznamů. Vazba N:M v aktuální době není podporována.

### Automatické generování formulářů

Další specialitou frameworku jsou automaticky generované formuláře, které právě pomocí modulu a atributů jednotlivých vlastností modulu vytvoří daný design. Jsou 2 typy formulářů, jeden pro práci se záznamem (přidání a editace), další pak pro zobrazení dat v režimu pouze pro čtení, kdy se zobrazí i další vypočítané hodnoty, které se do tabulky neukládají. Navíc všechny data ke vzhledu formuláře se po vygenerování ukládají, aby při dalším otevření bylo jejich načítání rychlejší.

### Windows 10, UWP a možná přenositelnost

Stěžejní požadavek na řešení, propojení frameworku s Windows 10 a postavení na technologii UWP tak, aby využíval co nejvíce jejich možností, jako jsou aktualizace právě přes Microsoft Store. Framework je již od prvního momentu stavěn výhradně pro platformu UWP, ale vzhledem k vývoji na .NET Core je jeho možnost vytvoření alternativy pro ostatní systémy velmi jednoduchá.




**Obrázek 2.** Hlavní stránka ukázkové aplikace

Po podrobné analýze těchto požadavků byl navržen systém registrace modulů a přednastaveného UI, kdy vývojáři stačí vytvořit modul a registrovat jej při spuštění aplikace. V případě registrace je zapotřebí mít unikátní datovou třídu použitou v našem modulu a unikátní název daného modulu. Pokud chceme využít vyšších možností modulu, např. zobrazení v aplikaci, je nutné implementovat další požadované vlastnosti. Framework proto nabízí kromě základního modulu také 2 další typy, UI modul a datový modul, který je možné použít i pro ukládání do databáze a vytváření relací mezi jednotlivými tabulkami.

Při každé registraci se pak provede analýza modulu pomocí reflexe a anotací jednotlivých vlastností, zda splňuje požadované podmínky, aby nedošlo k práci s chybnými datovými typy anebo nastavení protichůdných atributů. Framework interně pracuje v oblasti designu pomocí návrhového vzoru MVVM (Model-View-ViewModel), a proto pro tvorbu UI částí je nutné vytvořit všechny view (design, zobrazení) modulu a jeho view modely, ale jen pro potřeby dědění, protože framework už většinu dat doplní sám a jsou pak místem pro vývojářovi další úpravy.

## 5. Použití frameworku ISUF v aplikaci

Implementace frameworku je velmi jednoduchá. V aplikaci stačí vytvořit základní zobrazení (hlavní stránka a zaobalení celé aplikace), jejich příslušné view modely, moduly, UI částí modulu a jeho view modely, ale pro všechny tyto části jsou připraveny abstraktní třídy, které stačí v daném souboru zdědit. V této kapitole ukážu, jak by se vytvořil jednoduchý systém s přednáškovými místnostmi a studenty v nich.

Prvně si naimportujeme z NuGet balíčky ISUF.UI, který stáhne všechny potřebné balíčky nižších vrstev, dále Template10 ve verzi 1.1.12 a Newtonsoft.Json ve

verzi 10.0.3.

Pro ukázku studenta nám postačí vytvořit modul se jménem studenta a místností, ve které se nachází. Pro místnost obdobně název a seznam studentů v ní.

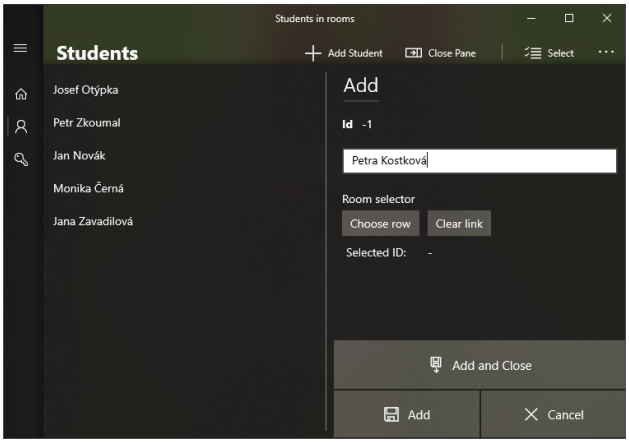
```
public class Student : AtomicItem
{
 public string Name { get; set; }

 [LinkedTable(
  LinkedTableRelation =
  LinkedTableRelation.One,
  LinkedTableType = typeof(Room))]
 public int Room { get; set; }
}

public class Room : AtomicItem
{
 public string Name { get; set; }

 [LinkedTable(
  LinkedTableRelation =
  LinkedTableRelation.Many,
  LinkedTableType = typeof(Student))]
 public List<int> Students { get; set; }
}
```

Tím jsme vytvořili základní moduly. Oba moduly budeme chtít zobrazit v aplikaci, bude tedy nutné k nim vytvořit další třídy, jako jsou hlavní stránka modulu, view modely a komponenty pro zobrazení detailu a přidání záznamu. Nejprve začneme view modely. V ukázce si představíme view model pro modul studenta, který dědí od třídy ModuleVMBase a který očekává jako generický parametr třídu modulu. Poté naimportujeme všechny funkce abstraktní třídy a doplníme funkčnost. Důležité je upravit funkci AddPane, která obstarává volání a vytváření všech vedlejších panelů (přidat/editovat a detail).

```
public class StudentViewModel :
 ModuleVMBase<Student>
```

Další komponenty, které si vytvoříme, budou panely pro práci a zobrazení záznamu. Všechny tyto komponenty mají i design část napsanou v jazyce XAML, kde je nutné nahradit základní třídu UserControl za příslušné třídy odkazující na daný typ. Pro detail je ModuleDetailControlBase a pro práci se záznamem ModuleAddControlBase. Implementace obou tříd je stejná, takže si ukážeme pouze způsob vytvoření komponenty na práci se záznamem.

```
public sealed partial class Add :
 ModuleAddControlBase
{
 public Add(UIModule uiModule,
  Type viewModelType,
  params object[] viewModelArgs) :
  base(uiModule,
  viewModelType,
  viewModelArgs)
```




**Obrázek 3.** Ukázka zobrazení modulu včetně bočního panelu

```
 {
 }
}

<isuf:ModuleAddControlBase
 xmlns:isuf="using:ISUF.UI.Views"
 x:Class="FITVUTBasicIS.Controls.Add"
```

A samozřejmě k nim vytvořit příslušný view model, zde opět ukázka pro práci se záznamem a pro modul Student. U view modelu pro práci se záznamem (anebo pro zobrazení detailu záznamu) je nutné dědit třídu ModuleAddVMBase (ModuleDetailVMBase), která očekává generický parametr typu daného modulu. Opět implementujeme abstraktní třídu a doplníme nutné chování.

```
public class StudentAddViewModel :
 ModuleAddVMBase<Student>
```

Pro práci se zobrazeným modulem se pak jedná o třídu ModulePageBase. V ukázce implementace pro modul Student.

```
public sealed partial class StudentPage :
 ModulePageBase
{
 public StudentPage() :
  base(typeof(StudentViewModel))
 {
 }
}
```

V případě komponenty pro práci se záznamem a detailem záznamu je nutné implementovat bohatý konstruktor, který určuje jak UI modul, tak view model, který se má použít pro tuto komponentu, a jeho argumenty. Pro zobrazení pak jen stačí určit typ view modelu, který odpovídá danému modulu pro zobrazení a v základu není nutné pro něj přidávat žádné argumenty. Tímto způsobem lze použít jednu komponentu pro všechny moduly a jen specifikovat její chování při inicializaci. V případě UI modulu není nutné speci-

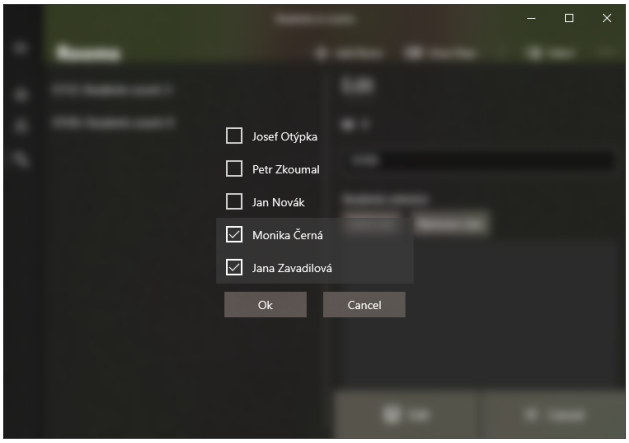
fikovat modul, protože je vyčten z view modelu, který v základu také nepotřebuje specifikovat argumenty.

Nyní máme vytvořenou datovou část, tudíž se můžeme vrhnout na obecné prvky aplikace. Nejprve si vytvoříme view model pro hlavní stránku. Zde stačí pouze implementovat dědičností abstraktní třídu MainPageVMBase. Postup je stejný jako u výše zmíněných view modelů.

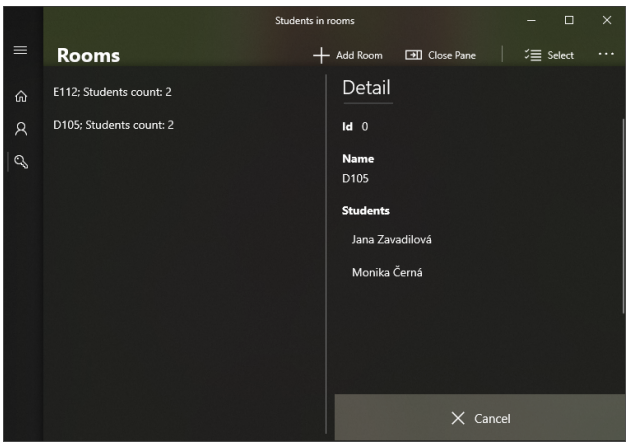
Další zobrazení (view), které je nutné implementovat, je stránka pro hlavní stránku, tedy to, co uvidíme hned po spuštění a co nám zobrazí seznam všech dostupných modulů, se kterými můžeme pracovat v aplikaci, a zobrazení, které zaobalí celou aplikaci. Opět bude nutné implementovat dědičnost abstraktní třídy do obou souborů (xaml a xaml.cs) zobrazení. V případě hlavní stránky pro dědičnost se využije třída ModulePageBase a po implementaci abstraktní třídy vidíme konstruktor, do kterého vložíme typ view modelu, který použijeme pro hlavní stránku. Pro zaobalení aplikace se používá třída ShellBase, kterou stačí implementovat a již ve výchozím stavu je plně funkční.

Nyní máme připravené všechny potřebné části a můžeme provést registraci ve třídě App. Zde budeme dědit třídu ApplicationBase, dědičnost použijeme v obou souborech (xaml a xaml.cs) a implementujeme abstraktní třídu. Dědičnost pro XAML jsme si ukázali již v příkladech výše, proto tedy jen základ implementace v kódu v C#.

```
public sealed partial class App :
 ApplicationBase
{
 public App() :
  base(Package.Current.DisplayName,
  typeof(Shell),
  typeof(MainPage),
  null)
 {
  InitializeComponent();
 }
 public override void RegisterModules()
 {
  ModuleManager =
   new UIModuleManager(
    typeof(XmlDbAccess));

  var studModule = new UIModule(
   typeof(Student),
   typeof(BaseItemManager),
   "Students", (Symbol)0xE13D,
   typeof(StudentPage));

  var roomModule = new UIModule(
   typeof(Room),
   typeof(BaseItemManager),
   "Rooms", (Symbol)0xE192,
   typeof(RoomPage));

  ModuleManager.RegisterModule(studModule);
```




**Obrázek 4.** Zobrazení propojené tabulky pro přidání do záznamu




**Obrázek 5.** Zobrazení informací o položce

```
  ModuleManager.RegisterModule(roomModule);
 }
```

Můžeme si všimnout konstruktoru, který obsahuje název aplikace, který se zobrazí v horní liště programu, typ třídy, která zaobalí celou aplikaci, dále samozřejmě typ hlavní stránky a jako poslední lze využít typ pro automatické vytvoření stránky pro nastavení aplikace.

Nejdůležitější částí je ale funkce RegisterModules, která slouží pro registraci všech modulů v aplikaci. Nejprve musíme inicializovat správce modulů, který očekává jako argument třídu pro přístup k datům. Dále budeme registrovat dva moduly se zobrazením pomocí třídy UIModule, která očekává v prvním argumentu typ modulu, dále typ třídy, která bude pracovat se záznamy, název modulu v aplikaci, ikonu z fontu Segoe MDL2 Assets a jako poslední je typ třídy pro zobrazení. Nakonec moduly zaregistrujeme do správce a máme hotovou aplikaci pro správu studentů a jejich přítomnost v přednáškových místnostech.

## 6. Ukázková aplikace

Abychom mohli nějak prezentovat výsledný produkt, bylo zapotřebí vytvořit aplikaci, která využívá možno-

sti frameworku. Z toho důvodu vznikla aplikace BrnoParkingIS. Základem jsou 2 moduly pro reprezentaci auta a parkoviště. Modul auta je reprezentován značkou a SPZ, parkoviště pak názvem. Mezi moduly je vazba 1:N, kdy na parkovišti je N aut, ale auto jen na 1 parkovišti. Aplikace jako způsob ukládání dat používá XML soubor ve složce aplikace.

Zdrojové kódy jsou dostupné na GitHub a pro zprovoznění je potřeba Windows 10 verze 1903.

## 7. Závěr

Práce se věnuje problematice tvorby frameworků nad operačním systémem Windows 10 a jeho technologií UWP, možnosti zjednodušené tvorby informačního systému a jeho použitelnost v praxi. Navrhované řešení je prezentováno jako koncept a ukázka možného postavení se k problému. Pro potřeby prezentace funkčního prototypu vznikla aplikace pro správu Brněnských parkovišť BrnoParkingIS, která využívá v největší možné míře možnosti frameworku a nechává celé řízení na něm.

Zde představená práce je sice koncept, ale věřím, že mnou navrhnuté řešení je použitelné v komerční sféře, a to i přes aktuální nedokončenost. Když se podíváme na zdrojový kód, můžeme si všimnout množství znovu implementovaných postupů, které bylo nutno vytvořit, protože použitá technologie v kombinaci s řešením je nepodporuje. Na druhou stranu nám ale nabízí jednoduchý způsob aktualizací a jednotnou platformu s obrovským množstvím zařízení, jejichž počet se stále rozrůstá.

## Literatura


[1] .net native github readme. Online documentation, March 2019. https://github.com/Microsoft/dotnet/blob/master/releases/UWP/README.md.

[2] Yusuf Mehdi. Windows 10: Powering the world with 1 billion monthly active devices. Blogpost, March 2020. https://blogs.windows.com/windowsexperience/2020/03/16/windows-10-powering-the-world-with-1-billion-monthly-active-devices/.

[3] Microsoft dynamics. Online documentation, April 2020. https://dynamics.microsoft.com/en-us/.

[4] expressapp framework. Online documentation, August 2019. https://docs.devexpress.com/eXpressAppFramework/112670/expressapp-framework.

[5] Mm .net application framework. Product page. https://www.oakleafsd.com/products.

[6] Syncfusion. Documentation. https://help.syncfusion.com/.

[7] Telerik. Documentation. https://www.telerik.com/support.

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

# Evoluční návrh konvolučních neuronových sítí

Michal Piňos

**Abstrakt**
Cílem této práce je návrh a implementace programu pro automatizovaný návrh konvolučních neuronových sítí (*CNN*) s využitím evolučních výpočetních technik. Z praktického hlediska tento přístup redukuje potřebu lidského faktoru při tvorbě CNN, a tak eliminuje zdlouhavý a namáhavý proces návrhu. Tato práce využívá speciální formu genetického programování nazývanou kartézské genetické programování, které pro zakódování řešeného problému využívá grafovou reprezentaci. Tato technika umožňuje uživateli parametrizovat proces hledání CNN, a tak se zaměřit na architektury zajímavé z pohledu použitých výpočetních jednotek, přesnosti či počtu parametrů. Navrhovaný přístup byl otestován na standardizované datové sadě CIFAR-10, která je často využívána výzkumníky pro srovnání výkonnosti jejich CNN. Prvotní experimenty ukázaly, že vytvořená implementace (využívající GPU akceleraci) je schopna vytvořit či vylepšit přesnost CNN. Výsledkem experimentů, kdy bylo pro trénování k dispozici pouze několik epoch, byla řešení s přesností 64.5 % a počtem parametrů 146K při využití základních vrstev a řešení s přesností 74.5 % s počtem parametrů 475K při využití reziduálních vrstev. Záměrem těchto experimentů bylo dokázat funkčnost implementovaného programu a *proof-of-concept* navržené metody.

**Klíčová slova:** Neuroevoluce — Neuronové sítě — Evoluční výpočetní techniky

**Přiložené materiály:** N/A

*xpinos03@stud.fit.vutbr.cz, *Fakulta informačních technologií, Vysoké učení technické v Brně*

## 1. Úvod

Návrh architektury konvolučních neuronových sítí dnes hraje důležitou roli ve výzkumu umělé inteligence (konkrétně v oboru rozpoznávání obrázků). Návrh nových architektur CNN vyžaduje velké úsilí, mnoho zkušeností a pokusů. Tento proces je tak velmi namáhavý a ne vždy efektivní. Z tohoto důvodu je nutné přemýšlet o nových způsobech tvorby těchto sítí, které by nebyly tak složité a náročné, a přitom poskytovaly výsledky minimálně srovnatelné s ručně navrženými sítěmi. Technika automatizovaného hledání architektur umělých neuronových sítí, nazývaná NAS (Neural Architecture Search) [1], je v dnešní době velmi populární [2, 3] a již dala vzniknout hlubokým neuronovým sítím překonávajícím ty odborně navržené [4, 5, 6].

Tvorba nových architektur CNN rovněž obnáší problém optimalizace navrhované sítě z hlediska počtu parametrů, hyperparametrů (počtu vrstev, skrytých neuronů, ...) a potřebného výpočetního výkonu. Tento požadavek je dán tím, že velikost výsledné sítě (ať už co se týká počtu parametrů nebo hyperparametrů) hraje důležitou roli z pohledu požadavků na zdroje zařízení, na kterém poběží. V dnešní době mobilních a vestavěných zařízení s omezenými zdroji je tento požadavek zcela pochopitelný. Najít optimalizační metodu, která by byla schopna tento úkol zcela vyřešit, je ale prakticky nemožné, a tak je často nutné uchýlit se k různým heuristickým metodám. Ty jsou schopny na úkor přesnosti, úplnosti či efektivity nalézt takové řešení, které se co nejvíce blíží zadaným požadavkům. Jednu skupinu takovýchto metod tvoří evoluční výpočetní techniky [7], založené na poznatcích ze světa biologické evoluce.

Cílem této práce je využití evolučních výpočetních technik jako optimalizační metody při automatizovaném návrhu nových architektur konvolučních neuronových sítí. Pro tyto účely byla zvolena speciální technika nazývaná kartézské genetické programování [8]. S využitím této techniky se lze zaměřit na hledání

zajímavých architektur z pohledu přesnosti, počtu parametrů či technických požadavků na výslednou síť.

Výsledkem této práce je implementace programu pro automatizovaný návrh konvolučních neuronových sítí, využívající uživatelem definované výpočetní prostředky (v našem případě GPU), určených k řešení specifických problémů.

## 2. Teoretické základy

### 2.1 Konvoluční neuronové sítě

Konvoluční neuronové sítě (Convolutional Neural Networks, zkráceně *CNN*) tvoří skupinu hlubokých neuronových sítí, které se specializují na zpracování obrazových dat. Moderní konvoluční neuronové sítě se skládají ze čtyř základních typů vrstev – *konvoluční*, *aktivační*, *seskupující* a *plně propojené*. Jejich možné umístění v architektuře CNN je uvedeno na obrázku 1.

Konvoluční vrstvy jsou zodpovědné za extrakci užitečných rysů ze vstupních obrázků. K tomu využívají operaci konvoluce, která realizuje aplikaci filtrů (označovaných též jako jádro či kernel) na vstupní obrázek. Ten má zpravidla několik kanálů (v případě RGB obrázku 3), přičemž konvoluce je prováděna zvlášť pro každý z nich.

Aktivační vrstvy plní důležitou roli, jelikož do modelu umělých neuronových sítí vnáší prvek nelinearity. Bez aktivačních vrstev by sebesložitější vícevrstvá neuronová sítí představovala jen o něco složitější, jednovrstvou neuronovou síť. Aktivační vrstvy jsou zpravidla umisťovány za lineární operace jako je konvoluce či bázová funkce perceptronu. Nejpoužívanější aktivační funkcí je usměrněná lineární jednotka (Rectified Linear Unit, zkráceně *ReLU*) [9].

Účelem seskupujících vrstev je zachovat pouze důležité informace z obrázku a zbavit se nepodstatných detailů, jako je například umístění či natočení extrahovaných rysů. Seskupující vrstvy rovněž redukují prostorové rozměry vnitřní reprezentace vstupního obrázku (*downsampling*). Tato vrstva se nejčastěji umisťuje za konvoluční, přičemž pracuje zvlášť pro každý vstupní kanál.

Poslední část klasifikačních CNN tvoří plně propojená neuronová vrstva, určená ke klasifikaci vstupních obrázků do výstupních tříd. Před tuto část je zpravidla nutné vložit zplošťující vrstvu, která transformuje vnitřní tří dimenzionální reprezentaci obrázku (s dimenzemi výška, šířka a počet kanálů) do podoby vhodné pro zpracování plně propojenou vrstvou.

### 2.2 Evoluční algoritmy

Evoluční algoritmy (zkráceně *EA*) jsou stochastické, heuristicky založené algoritmy inspirované biologickou evolucí. Jejich primárním úkolem je optimalizace problémů, pro které neexistuje žádný efektivní algoritmus[1]. Základní premisou konvergence EA je, že jedinci, kteří splňují určité požadavky (jsou nadprůměrně zdatní), budou mít potomky, a tak budou moci předat své užitečné vlastnosti do další generace.

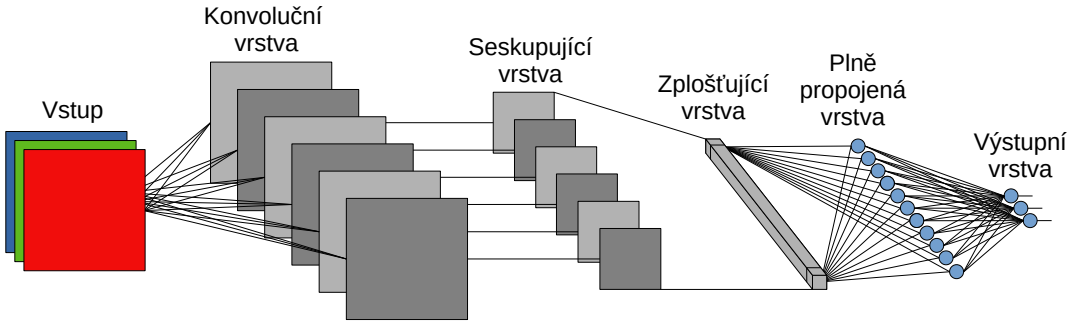
EA pracují nad množinou *jedinců*, kteří představují nějaká validní řešení daného problému. Evoluce probíhá postupně po iteracích, nazývaných *generace*, do té doby, dokud nejsou splněny *ukončující podmínky*. Mezi ty patří například nalezení řešení s dostatečnou kvalitou nebo vyčerpání maximálního počtu generací. Množina jedinců, se kterými algoritmus pracuje v rámci jedné generace, se nazývá *populace*. V každé generaci je s využitím operátoru *selekce* vybráno několik jedinců, kteří se budou podílet na tvorbě nové populace. Z vybraných jedinců jsou pomocí operátorů *křížení* a *mutace* vyrobeni *potomci*, kteří tvoří populaci následující generace. Jedinci jsou obvykle uloženi v lineární struktuře nazývané *genotyp*. Jednotlivé části genotypu se nazývají *geny*, přičemž konkrétní forma genu se nazývá *alela*. Skutečná forma jedince se nazývá *fenotyp*.

### 2.3 Kartézské genetické programování

Kartézské genetické programování (Cartesian Genetic Programming, zkráceně *CGP*) [8] je speciální forma genetického programování, ve které jednotliví jedinci představují programy reprezentované acyklickými orientovanými grafy (Directed Acyclic Graph, zkráceně *DAG*). Uzly DAG jsou tvořeny výpočetními jednotkami, uspořádanými ve dvoudimenzionální mřížce (odtud název *"kartézské"*).

CGP má tři hlavní, uživatelem definované, parametry, počet řádků $r$, počet sloupců $c$ a levels-back $l$. První dva parametry udávají maximální počet výpočetních uzlů, který je $r \times c$. Nastavením parametru $l$ uživatel udává, z kolika předchozích sloupců může daný uzel brát své vstupy. Tento parametr tak ukládá omezení při tvorbě propojení jednotlivých uzlů. Pokud $l = 1$, tak může každý uzel brát vstup z výstupů uzlů z předchozího sloupce nebo primárních vstupů. Při $l = 2$ může uzel brát svoje vstupy z výstupů uzlů ve dvou z levé strany sousedících sloupců nebo z primárních vstupů. Změnou těchto parametrů lze docílit různých topologií grafu. Speciálním případem je situace, kdy $r = 1$ a $l = c$. Při tomto nastavení může vzniknout libovolně propojený DAG, omezený pouze počtem uzlů, který je roven $c$.

[1] Efektivním algoritmem se z pohledu teorie složitosti rozumí algoritmus, který je schopný poskytnout řešení v polynomiálním čase.



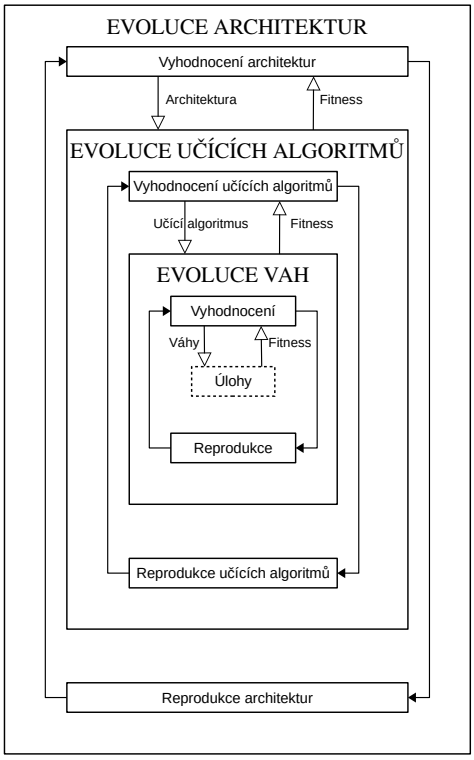

**Obrázek 1.** Ukázka jednoduché architektury CNN obsahující jednu konvoluční vrstvu, následovanou seskupující vrstvou, zplošťovací vrstvou, plně propojenou vrstvou a výstupní vrstvou.

Uživatel dále definuje počet vstupů a výstupů výpočetní mřížky, která se skládá z funkčních uzlů, přičemž funkce uzlu je vybrána z uživatelem definované množiny funkcí. Genotyp jedince má konstantní velikost a skládá se z genů jednotlivých uzlů a výstupních genů. Geny uzlů jsou dále rozděleny na funkční geny (udávající funkci uzlu) a propojovací geny, které stanovují, odkud bude uzel brát svůj vstup. Výstupní geny pak určují, z výstupů kterých uzlů bude tvořen výstup grafu.

Proces evoluce je u CGP řízen algoritmem inspirovaným evoluční strategií $(\mu + \lambda)$, nejčastěji ve formě $(1+4)$ [8]. V tomto jednoduchém algoritmu značí $\mu$ počet nejlepších jedinců, kteří jsou v každé generaci vybráni jako rodiče. Z nich je následně pomocí mutace vytvořeno $\lambda$ potomků. V drtivé většině případů používá CGP pouze operaci mutace a nejčastěji pracuje s jedním rodičem a $\lambda$ potomky v každé generaci.

## 2.4 Neuroevoluce

Neuroevoluce se zabývá využitím evolučních výpočetních technik při návrhu umělých neuronových sítí. Evolučně navržené neuronové sítě se označují jako EANN (Evolutionary Artificial Neural Networks) [10]. Proces evoluce je v kontextu tvorby umělých neuronových sítí využit v několika instancích, přičemž asi nejčastější je využití evoluce pro návrh architektury neuronové sítě. Zbylé dvě instance se zabývají evolucí vah nebo ještě obecněji návrhem učícího algoritmu. Evoluční algoritmy určené k evoluci architektur a vah neuronových sítí se označují jako TWEANN (Topology and Weight Evolving Artificial Neural Network). Nejznámějším takovýmto algoritmem je genetický algoritmus NEAT (NeuroEvolution of Augmenting Topologies), představený autorem Kennethem O. Stanleym v roce 2002 [11]. TWEANN algoritmy se nejčastěji dělí na konstruktivní a destruktivní [10]. Konstruktivní algoritmy začínají s minimální funkční architekturou a v průběhu evoluce přidávají další prvky (uzly nebo spojení) do výsledné architektury. Destruktivní algoritmy naopak začínají s architekturou s maximálním množstvím prvků a v průběhu evoluce náhodné prvky eliminují. Obecné schéma neuroevoluce je uvedeno na obrázku 2.




**Obrázek 2.** Univerzální schéma neuroevoluce. Převzato z [10].

# 3. Související práce

Tato sekce obsahuje krátké shrnutí a porovnání již existujících prací, zabývajících se tematikou evolučního návrhu konvolučních neuronových sítí. Zvláštní pozornost je zde věnována řešením zaměřujícím se na evoluční návrh architektur CNN s využitím CGP.

Využití CGP při návrhu umělých neuronových sítí se označuje jako *CGPANN* (Cartesian Genetic Programming of Artificial Neural Networks) [12]. Jedná se o neuroevoluční algoritmus, který využívá přímého zakódování architektury, vah a funkcí do genotypu.

CGPANN tak představuje TWEANN algoritmus, který při procesu evoluce využívá výhradně operátoru mutace. Na rozdíl od genetických TWEANN algoritmů je CGPANN konstruktivní i destruktivní algoritmus.

Příklad úspěšného využití CGP při návrhu CNN je uveden v práci [13]. Její autoři vytvořili metodu nazvanou CGP-CNN, která používá CGP kódování jedinců, schopné reprezentovat architekturu a propojení výsledné CNN. Výhodnou tohoto kódování je jeho flexibilita, jelikož umožňuje reprezentaci různě hlubokých sítí i využití *skip* propojení. Mimo jednoduché výpočetní uzly CGP mřížky, jako jsou konvoluční a seskupující vrstva, autoři uvedli i složitější moduly, nazvané *ConvBlock* a *ResBlock*. Tyto moduly představují komplexnější výpočetní uzly vytvořené z jednodušších operací.

ConvBlock se skládá z konvoluce s krokem 1 a ReLU aktivační funkce. Parametry tohoto modulu jsou velikost filtru ($3 \times 3$ nebo $5 \times 5$) a počet výstupních kanálů (32, 64 nebo 128).

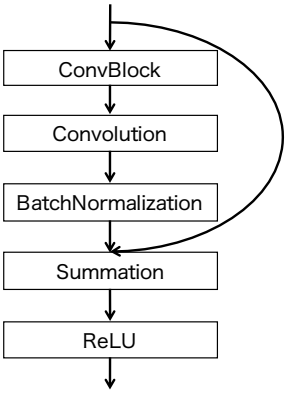
ResBlock je složen z konvoluce, konkatenace a ReLU aktivační funkce. *Skip* propojení přeskakuje konvoluční část tohoto modulu a přivádí nezměněný vstup modulu do konkatenačního uzlu, kde dochází ke konkatenaci s výsledkem konvoluce. Na výsledek konkatenace je následně použita aktivační funkce ReLU. Tento modul obsahuje parametry shodné jako u modulu ConvBlock. Architektura modulu ResBlock je uvedena na obrázku 3.

Experimenty s touto metodou její autoři rozdělili na dva případy, podle toho, zda byly použity ConvBlock nebo ResBlock moduly. Experimenty byly prováděny na datasetu CIFAR-10. Nejlepší vytvořená architektura v případě použití modulů ConvBlock dosáhla přesnosti 93.25 % s 1.52M parametrů. Architektura vytvořená z modulů ResBlock dosáhla přesnosti až 94.02 % s 1.68M parametrů. Provedené experimenty ukázaly, že metoda CGP-CNN je schopna vytvořit architektury porovnatelné s těmi ručně navrženými. V případě využití ResBlock modulů výsledná síť překonala i jedny z nejlepších ručně navržených sítí [13].

### 3.1 Srovnání existujících řešení

Tabulka 1 obsahuje srovnání přesnosti a počtu parametrů konvolučních neuronových sítí na datasetu CIFAR-10, navržených různými evolučními algoritmy. Pro srovnání obsahuje tabulka i současnou *state-of-the-art* odborně navrženou architekturu CNN nazvanou DenseNet [14].

Model NAS [4] využívá rekurentní neuronovou síť LSTM s posilovaným učením pro generování architektur CNN. Model nazvaný CNN-GA [15] je založen na využití genetického programování pro návrh architektur CNN. Skvělých výsledků CNN-GA dosáhlo díky využití takzvaných *skip* propojení, které přeskakují některé vrstvy v architektuře. Díky tomu dochází k lepší zpětné propagaci chyby do počátečních vrstev, a tak k rychlejší konvergenci. Posledním zkoumaným modelem v této sekci je CGP-CNN [13], využívající metodu CGP pro automatické hledání optimálních architektur CNN.




**Obrázek 3.** Architektura modulu ResBlock. BatchNormalization provádí normalizaci hodnot po procesu konvoluce. Převzato z [13].

| Srovnání algoritmů pro CIFAR-10 | | | |
|---|---|---|---|
| Název modelu | Přesnost (%) | Počet parametrů | Doba běhu (GPU dny) |
| DenseNet [14] | 96.54 | 25.6M | – |
| NAS [4] | 93.99 | 2.5M | 22 400 |
| CGP-CNN [13] | 94.02 | 1.68M | 27 |
| CNN-GA [15] | 95.22 | 2.9M | 35 |

**Tabulka 1.** Srovnání parametrů CNN, které byly získány pomocí různých přístupů (jeden GPU den znamená, že algoritmus strávil jeden den na jednom GPU).

## 4. Navržené řešení

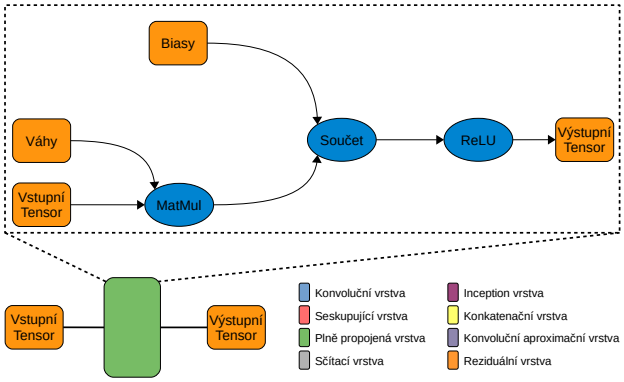
Pro automatizovaný návrh CNN architektur byla zvolena metoda CGP, kde jednotlivé uzly CGP mřížky představují určité vrstvy CNN modelu. CGP tak hledá architekturu sítě CNN, přičemž učení vah je ponecháno na algoritmu stochastického gradientního sestupu s využitím zpětného šíření chyby.

Pro práci s neuronovými sítěmi byla zvolena knihovna TensorFlow. Hlavními dvěma motivátory pro volbu této knihovny byla podpora akcelerace procesu učení na GPU a fakt, že výzkumná skupina *Evolvable Hardware* pracuje na rozšíření *tf-approximate*[2] knihovny TensorFlow o využití aproximačních jednotek.

Implementovaný program vytváří z výpočetních grafů knihovny TensorFlow jednotlivé vrstvy konvo-

[2] https://github.com/ehw-fit/tf-approximate

lučních neuronových sítí. Na obrázku 4 je uveden příklad implementace jedné vrstvy CNN, konkrétně plně propojené vrstvy, pomocí výpočetního grafu. Jak obrázek naznačuje, tak obdobným způsobem lze realizovat libovolnou vrstvu, jako je například konvoluční, seskupující, sčítací či konkatenační. Z těchto jednoduchých vrstev lze vytvořit i složitější výpočetní celky, jako je reziduální blok (popsaný v sekci 3) nebo tzv. *Inception* modul [16].




**Obrázek 4.** Příklad implementace plně propojené vrstvy pomocí TensorFlow výpočetního grafu.

Funkční uzly CGP mřížky obsahují implementaci individuálních vrstev CNN architektury, jak je uvedeno na obrázku 5. Každý DAG tak představuje nějakou validní architekturu CNN. Nastavením počtu parametrů CGP mřížky může uživatel ovlivnit, jaké architektury bude navržený program vytvářet. Počet sloupců mřížky $c$ udává maximální hloubku navrhovaných CNN. Pomocí počtu řádků $r$ lze zase dát programu na výběr z několika alternativ funkčních uzlů v každém sloupci. Parametr $l-$back pak udává omezení navrhovaných CNN z pohledu možných propojení jednotlivých vrstev. Tento parametr rovněž umožňuje vytváření užitečných *skip* propojení.

Dalším užitečným vstupem, který může uživatel programu poskytnout, je definice samotné výpočetní mřížky CGP. Vhodně rozmístěné funkční uzly ve výpočetní mřížce mohou totiž ulehčit hledání efektivních CNN architektur.

Primárním vstupem CGP je obrázek, vyjádřený jako 3D tensor, s dimenzemi výška, šířka a počet kanálů vstupního obrázku. CGP pracuje s tensory s hodnotami typu float. Primárním výstupem CGP je 1D tensor (vektor), vyjadřující příslušnost vstupního obrázku do jedné z výstupních tříd. Výstupní tensor pro tyto účely používá kódování 1 z $N$.

## 4.1 Mutace

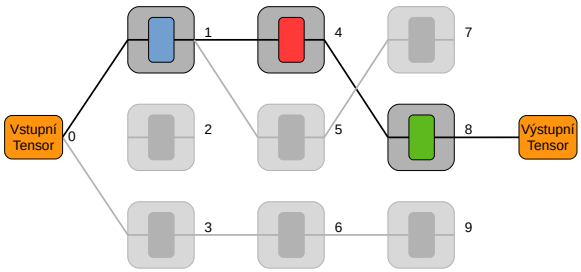
Implementovaný program pro zakódování jedinců nevyužívá klasické kódování chromozomu jako sekvence celých čísel, jak je tomu u většiny genetických algoritmů. V tomto případě je jedinec reprezentován přímo jako DAG, tedy dvojice $G = (U, E)$, kde $U$ značí množinu uzlů a $E$ množinu hran. Mutace je prováděna na grafu $G$, přičemž při mutaci dochází pouze ke změně propojení uzlů. V terminologii genetického programování to znamená, že mutace mění pouze propojovací a výstupní geny chromozomu. Funkční geny zůstávají nezměněny.

Mutace probíhá tak, že je náhodně vybrán uzel k mutaci $n \in U$. Pro uzel $n$ jsou odstraněny všechny vstupní hrany, tedy $E = E \setminus \{(x,n) \mid x \in U\}$. Následně je spočítána množina

$$L(n) = \{u \mid u \in U \text{ a } u \text{ je uzel respektující parametr l-back pro vybraný uzel } n\}$$

Z množiny $L(n)$ je následně náhodně vybrán jeden uzel $m \in L(n)$. Do množiny hran $E$ je poté přidána nová hrana $(m,n)$, tedy $E = E \cup (m,n)$. Pokud uzel $n$ reprezentuje funkci s vyšší aritou než je 1, tak je proces přidání nové hrany opakován.

Mutace jedince je prováděna do té doby, dokud není vytvořen aktivní podgraf spojující vstupní uzel s výstupním.



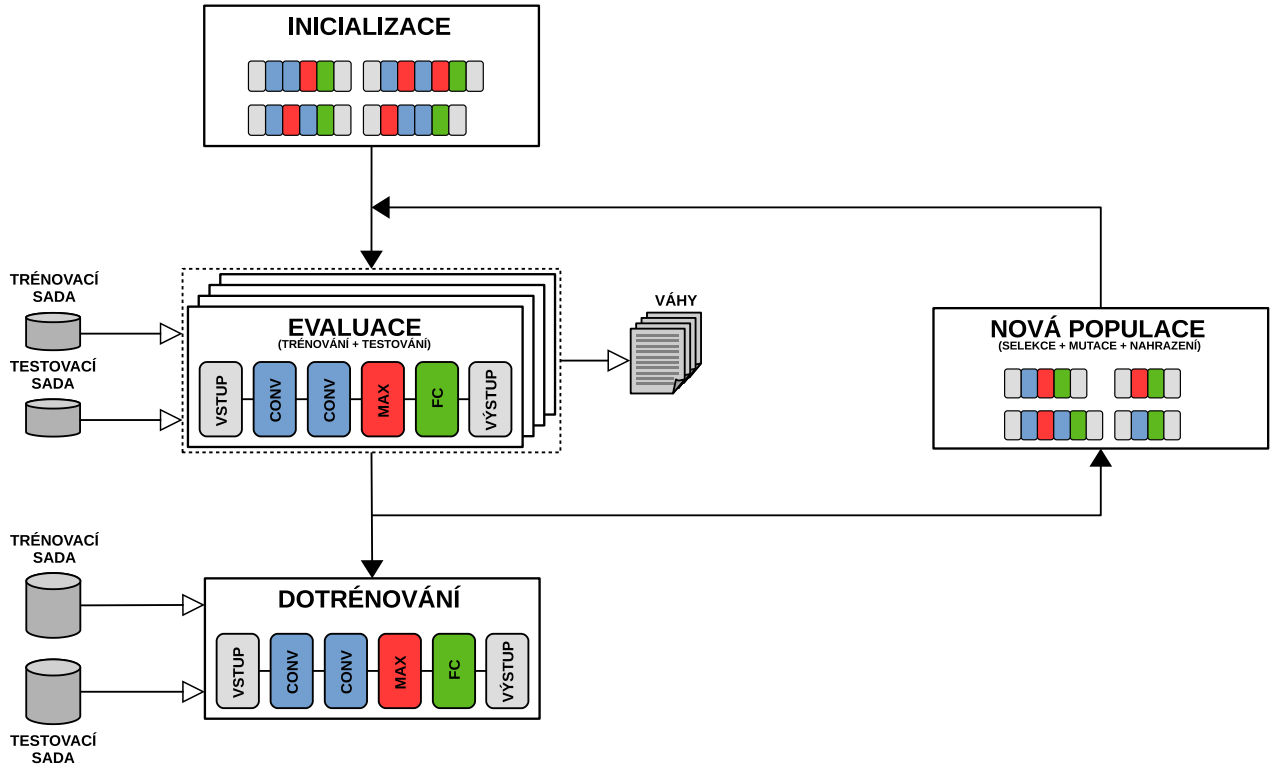

**Obrázek 5.** Ukázka DAG v CGP mřížce představující nalezené řešení, skládající se z konvoluční vrstvy (modrá), seskupující vrstvy (červená) a plně propojené vrstvy (zelená).

## 4.2 Prohledávací algoritmus

Implementovaný program využívá prohledávací algoritmus založený na evoluční strategii (1 + 4). Diagram prohledávacího algoritmu je uveden na obrázku 6.

Na úvod jsou vygenerovány 4 kandidátní řešení. Algoritmus dále vstupuje do smyčky, která provádí evaluaci každého jedince. Ta se skládá z trénování jedinců pomocí náhodně vybrané podmnožiny trénovací datové sady. Následné testování jedinců je opět prováděno na náhodně vybrané podmnožině testovací datové sady. Všichni jedinci v určité generaci jsou trénováni a testováni na stejných, náhodně vybraných podmnožinách originálních datových sad. Tento postup byl převzat z [13].

Během evaluace je jedincům přiřazena fitness hodnota podle jejich přesnosti a počtu parametrů. Výpočet




**Obrázek 6.** Diagram prohledávacího algoritmu. Jak je zde znázorněno, tak všichni jedinci v každé generaci jsou trénováni a testováni s využitím náhodně vybrané podmnožiny originální datové sady. Naučené váhy každého jedince jsou uloženy do souboru. Po skončení algoritmu je nejlepší jedinec (do)trénován na originální datové sadě.

fitness je proveden pomocí vzorce

$$f = a \times \left( \frac{k}{\log(p) + 1} + 1 \right), \tag{1}$$

kde $a$ značí přesnost, $p$ počet parametrů a koeficient $k$ udává vliv počtu parametrů na výslednou fitness hodnotu. Tato fitness funkce byla převzata z [17].

Z ohodnocených jedinců je poté vybrán nejlepší jedinec – *rodič*. Pokud je v současné populaci více jedinců s nejlepší fitness hodnotou, tak je vybrán jedinec, který ještě nebyl rodičem. V případě, že je takovýchto jedinců více, je zvolen náhodný z nich.

S využitím operátoru mutace, popsaném v podsekci 4.1, jsou z rodiče vytvořeni potomci, kteří tvoří populaci následující generace.

Hlavní smyčka prohledávacího algoritmu končí, až je dosaženo maximálního počtu generací. Nejlepší nalezené řešení je následně (do)trénováno s využitím originální datové sady.

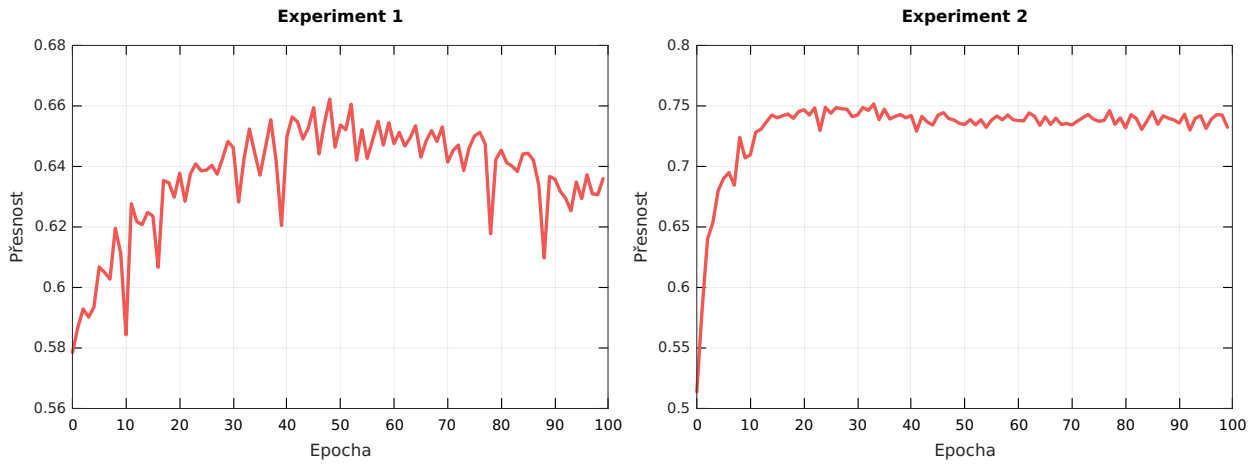
### 4.3 Uložení nejlepších vah

Z popisu prohledávacího algoritmu plyne, že v každé generaci jsou všichni jedinci podrobeni trénování před tím, než je určena jejich přesnost. Do další generace se však dostanou jen někteří jedinci, z čehož plyne, že trénování některých jedinců bylo zbytečné a jejich naučené váhy se zahodí. Další negativní vlastností tohoto přístupu je to, že noví jedinci budou muset začínat od začátku, s náhodně vygenerovanými váhami.

Implementovaný program tento problém řeší tak, že každý funkční uzel v CGP výpočetní mřížce si pamatuje naučené váhy nejlepšího jedince, který tento uzel využíval. Noví jedinci si pak při trénování načtou tyto váhy a nebudou tak muset začínat od znovu. Jelikož jsou ale rozměry váhového tensoru každého funkčního uzlu určeny tím, ke kterému funkčnímu uzlu je připojen, tak se mohou rozměry váhového tensoru pro ten samý uzel lišit. V tomto případě je váhový tensor buď oříznut nebo doplněn náhodnými hodnotami tak, aby odpovídal potřebným rozměrům.

## 5. Experimenty

Experimenty byly prováděny na datové sadě CIFAR-10, která se skládá z 60 000 barevných obrázků o velikosti $32 \times 32$, rozdělených do 10 tříd. Datová sada je dále rozdělena na trénovací sadu obsahující 50 000 vzorků a testovací sadu o velikosti 10 000 vzorků. Experimenty byly spuštěny na stroji se čtyřmi grafickými kartami NVIDIA GTX 1080 (Pascal), 8GB RAM, které byly použity pro akceleraci procesu učení CNN. Všechny ostatní výpočty byly prováděny na CPU.




**(a)** Přesnost během finálního dotrénování nejlepšího řešení, nalezeného v experimentu 1.

**(b)** Přesnost během finálního dotrénování nejlepšího řešení, nalezeného v experimentu 2.

**Obrázek 7.** Grafy experimentů.

## 5.1 Experiment 1

První experiment měl za cíl navrhnout architekturu CNN s použitím jednoduchých výpočetních bloků, jako jsou konvoluční, seskupující, konkatenační, sčítací a plně propojené vrstvy.

### 5.1.1 Nastavení programu

CGP mřížka obsahovala 6 řádků a 31 sloupců s parametrem l-back nastaveným na 8. Mřížka byla vytvořena tak, že sloupce blíže primárnímu vstupu obsahovaly konvoluční vrstvy s větší velikostí filtru a menším počtem výstupních kanálů. Čím blíže se pak sloupce blížily výstupu CGP mřížky, tím se velikosti filtrů zmenšovaly a počty výstupních kanálů zvyšovaly. Tento postup byl zvolen z toho důvodu, že hlubší vrstvy si mohou dovolit mít více výstupních kanálů, jelikož pracují s menšími vstupy. Konvoluční sloupce CGP mřížky byly příležitostně proloženy sloupci se seskupujícími, konkatenačními a sčítacími vrstvami. Poslední sloupec se skládal z plně propojených vrstev s počtem neuronů 64, 128, 256, 512, 1024 a 2048.

Počet generací prohledávacího algoritmu byl nastaven na 50, přičemž algoritmus v každé generaci pracoval s populací velikosti 4. V každé generaci bylo z trénovací datové sady náhodně vybráno 10 000 vzorků a z testovací datové sady 5 000 náhodných vzorků. Pomocí této mini-trénovací sady bylo provedeno trénování každého jedince po dobu 10 epoch. Přesnost každého kandidátního řešení pak bylo určeno evaluací na mini-testovací sadě. Parametr $k$ fitness funkce 1 byl nastaven na 0.5.

Po skončení prohledávacího algoritmu bylo nejlepší nalezené řešení (do)trénováno na plné trénovací sadě po dobu 100 epoch. Výsledná přesnost pak byla spočítána pro originální testovací sadu.

## 5.2 Experiment 2

Cílem druhého experimentu bylo navrhnout architekturu CNN s využitím ResBloku, popsaném na obrázku 3.

### 5.2.1 Nastavení programu

CGP mřížka obsahovala 6 řádků a 15 sloupců s parametrem l-back nastaveným na 4. Jednotlivé sloupce CGP mřížky obsahovaly ResBloky, konvoluční, seskupující, sčítací a konkatenační vrstvy s různými parametry. Poslední sloupec se skládal z plně propojených vrstev s počtem neuronů 64, 128, 256, 512, 1024 a 2048.

Nastavení prohledávacího algoritmu tohoto experimentu bylo shodné s nastavením v experimentu 1.

## 5.3 Výsledky

Přesnosti nejlepších nalezených řešení obou experimentů jsou ukázány na obrázku 7.

Výsledkem prvního experimentu bylo řešení s přesností 64.5 % a počtem parametrů 146178. Graf 7a zobrazuje průběh učení nejlepšího řešení při finálním dotrénování, které trvalo 100 epoch. Jak je vidět, tak přesnost dosáhla až k 66 % a přibližně od 50. epochy se snižovala. To ukazuje, že při procesu učení začalo docházet k přeučení (overfitting), kdy model začal ztrácet schopnost generalizace.

Výsledkem druhého experimentu bylo řešení s přesností 74.5 % a počtem parametrů 475722. Graf 7b zobrazuje průběh učení nejlepšího řešení při finálním dotrénování, které trvalo 100 epoch. Jak je vidět, tak přesnost dosáhla až k 75 % a přibližně od 30. epochy se opět začala pozvolna snižovat. To opět ukazuje na přeučení.

## 6. Závěr

Cílem této práce bylo navrhnout a implementovat neuroevoluční algoritmus pro automatizovaný návrh architektur konvolučních neuronových sítí. Toho bylo dosaženo použitím speciální techniky CGP. Pro práci s CNN byla zvolena knihovna TensorFlow, umožňující akceleraci výpočtů na GPU. Pro ověření funkčnosti implementovaného programu byly provedeny dva experimenty na datové sadě CIFAR-10, které dosáhly přesnosti 64.5 % s 146178 parametry a 74.5 % s počtem parametrů 475722.

Výsledkem této práce je program, implementující evoluční návrh architektur konvolučních neuronových sítí. Implementovaný program rovněž umožňuje uživateli specifikovat si různé požadavky a omezení na navrhované CNN. Uživatel tak může programu říci, jaké výpočetní jednotky může používat nebo jaká může být minimální či maximální hloubka sítě. Z praktického pohledu je tato možnost velmi užitečná, uvážíme-li, že bychom například hledali architekturu CNN pro nějaké zařízení s omezenými výpočetními prostředky.

### 6.1 Pokračování práce

Styl, jakým je program navržen a implementován podporuje možnost rozšíření CGP mřížky o nové, výkonnější funkční bloky. Implementovaný program může být rovněž použit pro návrh takovýchto bloků. Další alternativou je inicializace CGP nejlepšími známými CNN a hledání kompromisu mezi přesností a velikostí sítě.

Jiným podstatným rozšířením tohoto programu je využití aproximačních jednotek, které na úkor přesnosti snižují požadavky na příkon nebo plochu na čipu. Rozšíření *tf-approximate* implementuje tyto aproximační jednotky, ale nepodporuje proces učení. Tyto jednotky by ale mohly být lehce integrovány do implementovaného programu tak, že v procesu evaluace jedince by trénování probíhalo na přesných jednotkách, zatímco testování na těch aproximovaných. Výsledkem programu by pak byla naučená architektura CNN, která by mohla být na čipu implementována aproximačními jednotkami a tak ušetřit cenné zdroje vestavěného zařízení.

## Poděkování

Rád bych poděkoval svému vedoucímu práce panu profesoru Ing. Lukáši Sekaninovi, Ph.D. za odborné vedení, konzultace, čas a cenné rady při tvorbě této práce. Dále bych rád poděkoval doktoru Ing. Vojtěchu Mrázkovi a docentu Ing. Jiřímu Jarošovi, Ph.D. za zprostředkování přístupu na školní výpočetní server.

## Literatura

[1] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. *Neural Architecture Search: A Survey*. Journal of Machine Learning Research, 20(55):1–21, 2019.

[2] Md Ashiqur Rahman. *Neural Architecture Search (NAS) - The Future of Deep Learning*, June 2019. URL: https://towardsdatascience.com/neural-architecture-search-nas-the-future-of-deep-learning-c99356351136.

[3] Kenneth O. Stanley, Jeff Clune, Joel Lehman, and Risto Miikkulainen. *Designing neural networks through neuroevolution*. Nature Machine Intelligence, 1(1):24–35, 2019.

[4] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *International Conference on Learning Representations*, 2017.

[5] Lewei Yao, Hang Xu, Wei Zhang, Xiaodan Liang, and Zhenguo Li. *SM-NAS: Structural-to-Modular Neural Architecture Search for Object Detection*, 2019. URL: https://arxiv.org/abs/1911.09929.

[6] Ning Zhu. *Neural Architecture Search for Deep Face Recognition*, 2019. URL: http://arxiv.org/abs/1904.09523.

[7] Patrick Siarry, Alain Petrowski, and Sana Ben Hamida. *Metaheuristics*, chapter Evolutionary Algorithms. Springer International Publishing, 2016.

[8] J. F. Miller. *Cartesian Genetic Programming*, pages 17–34. Natural Computing Series. Springer Berlin Heidelberg, 2011.

[9] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress.

[10] Xin Yao. *Evolving artificial neural networks*. Proceedings of the IEEE, 87(9):1423–1447, September 1999.

[11] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[12] Maryam Mahsal Khan and Gul Muhammad Khan. A novel neuroevolutionary algorithm:

Cartesian genetic programming evolved artificial neural network (cgpann). In *Proceedings of the 8th International Conference on Frontiers of Information Technology*, New York, NY, USA, 2010. Association for Computing Machinery.

[13] Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao. A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, page 497–504, New York, NY, USA, 2017. Association for Computing Machinery.

[14] G. Huang, Z. Liu, L. v. d. Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, July 2017.

[15] Yanan Sun, Bing Xue, Mengjie Zhang, and Gary G. Yen. Automatically designing CNN architectures using genetic algorithm for image classification. 2018.

[16] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[17] Filip Badáň and Lukáš Sekanina. Optimizing convolutional neural networks for embedded systems by means of neuroevolution. In *Theory and Practice of Natural Computing*, pages 109–121. Springer International Publishing, Listopad 2019.

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

# Ultimate Arcade Suitcase

Lukáš Erlich*

**Abstrakt**
Cieľom tejto práce je vytvoriť zábavný kufrík (hardware aj software), ktorý umožňuje spúšťanie retro arkádových hier a funguje taktiež ako jukebox na prehrávanie mp3 piesní. Jadro kufru je tvorené Raspberry Pi a k nemu pripojenými audiovizuálnymi a kontrolnými periférami. Software pre konzolovú časť a jukebox je prevzatý z GitHub open-source projektov RetroPie a FruitBox. Nad operačným systémom je vytvorené jednoduché rozhranie a užívateľovi je taktiež umožnené spravovať kufor pomocou webového manažéra vo frameworku Laravel. Vytvoril som systém aplikácií, komunikujúcich s OS a hardwarovými komponentami. Hardwarová výbava kufra je kompletná a webový manažér je aktuálne z 80 % hotový, rozhranie je udržiavané jednoducho. Práca ukazuje využitie jednoduchých počítačov v komplexnejších systémoch a ich možnosti.

**Kľúčové slová:** Raspberry Pi — Web manager — Raspbian

**Priložené materiály:** RetroPie — FruitBox — GitHub

*xerlic00@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

## 1. Úvod

Mojím zámerom je vytvoriť kufrík, ktorý by umožnil užívateľom spúšťať jednoduché hry, z ich obľúbených herných konzol z mladosti alebo prehrávať mp3 piesne iným spôsobom ako je zvyčajné a zároveň si zachoval prenositeľnosť.

Za týmto účelom sú v kufri nainštalované open-source programy **RetroPie**, ktorý funguje ako emulátor starých hier (formát ROM) a **FruitBox**, mp3 prehrávač v štýle jukeboxu. Oba sú priamo prevzaté z verejných repozitárov dostupných na GitHub.com. Tieto programy sú navrhnuté s optimalizáciami priamo na Raspberry Pi.

Hardwarovú časť tvorí počítač Raspberry Pi a ku nemu pripojené vstupné a výstupné periférie a senzory. Intenzívne využitá je aj zbernica GPIO, na ktorú sú pripojené tlačidlá, senzory a relé. Externý zosilňovač umožňuje meniť aj výšky a basy výstupného zvuku, prípadne napojenie kufríku ku externej audio sústave.

Veľkú časť práce tvorí webový server poskytujúci zobrazovanie systémových informácií a ich aktuálneho stavu, jednoduchého správcu súborov na prácu s hrami a piesňami a ovládanie niektorých prvkov kufra. Ďalšiu časť softwarového vybavenia tvoria skripty na čítanie vstupov, ovládanie chladenia a iné.

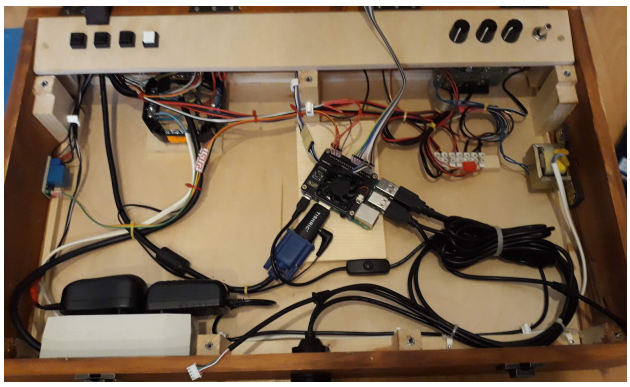
Podobné riešenia vytvoreného hardwaru existujú len osobitne s rôznymi formami, rozhodol som sa spojiť tieto obidve formy do jedného kufru a pridať prvky konzoly (joysticky) a jukeboxu (keypad). Týmto prepojením by vytvorený objekt umožnil príjemný zážitok pri rôznych príležitostiach.

Naše riešenie je použiteľné na rôzne udalosti, ako napríklad svadba (hudba), oslavy, teambuildingové akcie. Využité výkonnostné prostriedky sú v norme a zatiaľ sa nepreukázal ich nedostatok.

## 2. Hardwarová časť

Schránka kufra je zhotovená zo smrekového dreva a preglejky, spoje medzi schránkou a doskami držiacimi ovládacie prvky a monitor sú spojené skrutkami s metrickým závitom, čo umožňuje v prípade potreby prístup ku vnútorným komponentom. Za účelom stability a zachovania otvoreného kufra sa vo vrchnom diele nachádza vysúvacia nožička.

**Obrázok 1.** Vrchný diel kufra. V strede sa nachádza ukotvený monitor, po stranách vystlané plochy kvôli akustike a v pravo dole mechanizmus vyťahovacej nožičky. Po stranách sú ventilátory, zapojené opačným smerom za účelom lepšieho prúdenia vzduchu a vpravo hore sa nachádza teplotný senzor.



**Obrázok 2.** Spodný diel kufra. V strede Raspberry, vpravo hore zosilňovač, s výstupom do reproduktorov v hornom diely, vľavo sa nachádza rozvod prúdu a relé spúšťacie ventilátory.

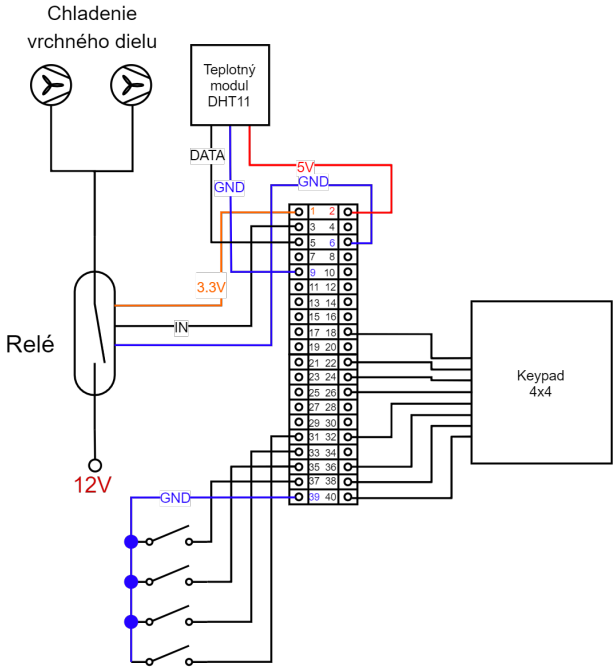
Vodiče senzorov a reproduktorov, dátový kábel monitora a napájanie vrchného dielu medzi časťami sú obalené za účelom ochrany koženým púzdrom. Rozvod správneho napájania jednotlivých súčiastok je rozdelený medzi viacero zdrojov (adaptérov), ktoré znižujú napätie na potrebnú hodnotu (5V, 12V) pri dostačujúcom elektrickom prúde (pri Raspberry až 3A). Spustenie kufra je potom jednoduché, pomocou pripojenia jedného napájacieho kábla a zapnutie hlavného vypínača pri zásuvke spodného dielu.

### 2.1 Vrchný diel

Vo vrchnej časti (Obrázok 1) je uprostred uchytený monitor, po stranách dva reproduktory pripojené ku zosilňovaču v spodnom diele. V stenách tejto časti sa taktiež nachádzajú softwarovo aktivované ventilátory a teplotný senzor (Sekcia 3.1.2). Ostré vnútorné hrany a rovné plochy kufra sú vyplnené molitanom, kvôli lepšej akustike.

### 2.2 Spodný diel

Spodná časť (Obrázok 2) je tvorená riadiacou jednotkou Raspberry Pi 3B+, na jeho GPIO zbernicu 3 sú pripojené tlačidlá ovládacieho panelu v režime pull-up, teplotný senzor z vrchného dielu a relé, ktoré spína obvod ventilátorov a keypad, ktorý je zatiaľ nepoužitý, ale v budúcnosti bude umožňovať ovládanie jukeboxu. Na Raspberry je pripojený aktívny softwarovo riadený chladič, ktorý je spúšťaný len po prekročení vopred stanovenej hodnoty (55°C, 65°C a 60°C) a na rôzny výkon (10%, 50% a 100%). Cez univerzálne sériové zbernice sú pripojené arkádové joystiky a tlačítka s vlastnými riadiacimi jednotkami.




**Obrázok 3.** Zbernica GPIO s pripojenými perifériami

Všetky tieto komponenty sú ovládané skriptami popísanými v sekcii 4.1. Ďalším významným komponentom je zosilňovač, ktorý umožňuje okrem hlasitosti nastavovať aj výšky a basy. V prípade potreby je možné pripojiť kufor ku repro sústave pomocou audio jack konektoru (6,3 mm).

# 3. Práca s OS Raspbian

V Raspberry Pi je vložená karta microSD s odľahčenou (Lite) verziou operačného systému Raspbian Buster. Na Raspberry Pi je povolené automatické prihlasovanie (autologin) a možnosť pripojenia pomocou SSH.

## 3.1 Štart operačného systému

### 3.1.1 Téma Plymouth

Operačný systém umožňuje vytvorenie vlastnej prihlasovacej a spúšťacej obrazovky (aj s animáciou), čoho som využil aj v mojej práci pre lepší a unikátnejší užívateľský zážitok. Vychádzal som z už existujúcich oficiálnych riešení. Tieto spúšťacie témy sú riadené skriptom, ktorý môže čítať systémové informácie a aktuálny stav a na základe nich vykresľovať obrázky alebo text priamo do framebufferu.

**Obrázok 4.** Prispôsobená plymouth téma, za účelom zachovania autenticity.

### 3.1.2 Spúšťanie aplikácií

Po štarte systému, je nutné spustiť aj mnou implementované aplikácie. Medzi ne patria skript ovládacieho panelu a skript na komunikáciu s teplotným senzorom (Sekcia 4.1). Oba tieto programy sú spúšťané ako systémová služba (súbor **systemd**). V tomto súbore je špecifikovaný spôsob spúšťania, v našom prípade po spustení všetkých ostatných systémových služieb.

Na definovanie vlastných premenných prostredia a niektorých ďalších nastavení je použitý súbor **/etc/profile**, ktorý sa spúšťa pri prihlásení každého, aj vzdialeného užívateľa.

## 3.2 Prístupový Bod

V rámci programového vybavenia kufra je aj webový server a je umožnené aj SSH pripojenie, je potrebné aby malo Raspberry zabezpečené pripojenie do siete. Keďže bežnému užívateľovi nie je povolený prístup k textovému rozhraniu a kufor neobsahuje klávesnicu na konfiguráciu Wi-Fi pripojenia do nových sietí, bolo nutné vytvoriť z Raspberry prístupový bod. Na jeho vytvorenie boli použité **DNSMasq** (poskytuje DHCP server) a **HostAPD** (software na tvorbu prístupového bodu). Po pripojení do lokálnej bezdrôtovej siete, je na IP adrese 192.168.125.1 a porte 8000 dostupný webový server (Sekcia 4.3).

## 3.3 Rozdelenie pamäti RAM

Obe hlavné aplikácie, RetroPie aj Fruitbox intenzívne pracujú s grafickou jednotkou. Keďže GPU na Raspberry neobsahuje osobitnú pamäť, je nutné vyčleniť z pamäte RAM určité percento pre správne fungovanie potrebných aplikácií (v prípade FruitBox sa jedná o 256 MB a Raspberry má 1 GB RAM).

# 4. Programová časť

## 4.1 Skripty na prácu s operačným systémom a periférami

Väčšina mnou vytvorených skriptov je používaných vo Webovom manažérovi na prácu s Raspbianom a súbormi. Táto skupina je implementovaná ako shell a php skripty.

### 4.1.1 Skript na ovládanie chladenia

Tento skript je implementovaný v jazyku Python a pracuje primárne so zbernicou GPIO 3. Z pinu číslo 5 číta informácie z teplotného senzoru aktualizované každých 5 sekúnd. Prijatá hodnota je porovnaná s hranicou 27°C a pri prekročení sa zopne relé na pine 3. Ventilátory sú zapnuté, kým sa teplota vrchného dielu nezníži aspoň o 4 stupne.

### 4.1.2 Skript ovládacieho panelu

Ovládací panel kufra sa skladá zo štyroch tlačidiel a troch potenciometrov, priamo napojených ku externému zosilňovaču. Tento skript slúži na prácu s tlačidlami. V súbore sú 4 GPIO piny 3 nastavené ako pull-up tlačidlá (interná schopnosť Raspberry). To znamená, že keď je tlačidlo rozopnuté, je na GPIO piny konzistentne logická 1 a pri zopnutí je treba priviesť logickú 0.

Skript číta tieto hodnoty a aktuálne spustené aplikácie. V prípade zopnutia niektorého z tlačidiel je vykonaná priradená funkcionalita. Ovládací panel umožňuje:

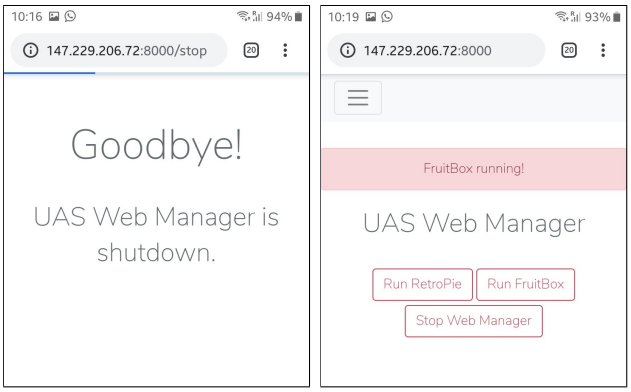
1. Spustenie/zastavenie aplikácie RetroPie
2. Spustenie/zastavenie aplikácie FruitBox
3. Spustenie/zastavenie Webového manažéra
4. Softwarové vypnutie celého kufru

## 4.2 Grafické rozhranie

Grafické rozhranie je aktuálne tvorené jedným statickým obrázkom a skriptom ovládacieho panelu (Sekcia 4.1.2). Vo finálnej fáze by malo byť užívateľovi umožnené grafické „listovanie“ medzi jednotlivými aplikáciami, čo umožní intuitívnejšie ovládanie.

**Obrázok 5.** Prototyp grafického rozhrania. Na obrázku je zobrazený výber z dvoch aplikácií.

## 4.3 Webový manažér

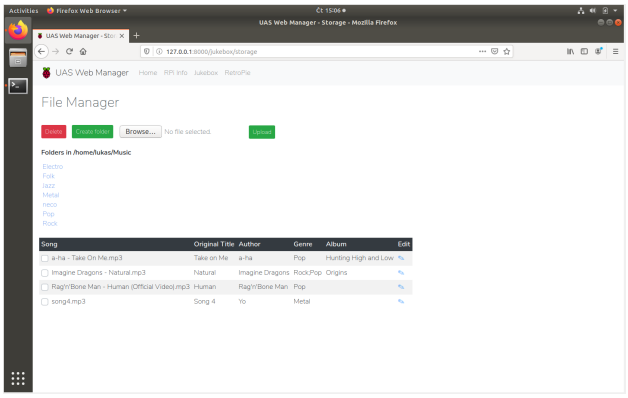



**Obrázok 6.** Úvodná stránka a stránka vypnutia manažéra.

Jedná sa o webový server implementovaný pomocou frameworku Laravel v jazyku PHP. Značnú časť tvoria funkcie v Javascripte na strane klienta, čím sme dokázali odľahčiť zaťaženie samotného Raspberry. Pomocou jQuery sú volané PHP scripty, pracujúce priamo s požadovanými časťami OS alebo súborového systému a vykresľované len nutné časti až u klienta. Užívateľovi je tu umožnené spúšťať a vypínať aplikácie priamo z webového prehliadača. Skladá sa z troch nosných častí: nástroje na prácu s RetroPie, Fruitbox a zobrazovanie systémových informácií.

### 4.3.1 Nástroje na prácu s FruitBox a RetroPie

Vzhľad a vlastnosti programu FruitBox sú založené na hlavnom konfiguračnom a prípadne dodatočných

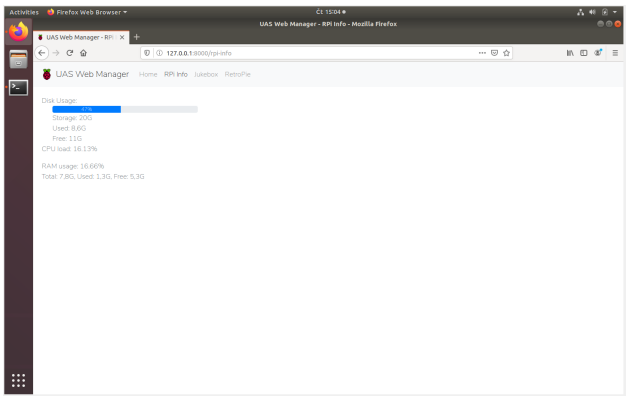



**Obrázok 7.** Správca súborov

súboroch (napr. na špecifikovanie ovládacích prvkov, databázy piesní a iné). Umožňuje výber vzhľadu, usporiadania piesní, výber fontu a iných. Keďže väčšina užívateľov nebude mať dostačujúce informácie a prístup ku konfiguračným súborom, rozhodol som sa vytvoriť niektoré funkcie grafickou formou. Medzi ne patria:

1. **Výber vzhľadu jukeboxu** – stránka umožňuje grafický výber témy a jej uloženie. Táto téma je nastavená ako štandardná, to znamená, pretrváva aj po vypnutí kufra.
2. **Správca súborov (piesní)** – nástroj na zobrazovanie, nahrávanie a odstraňovanie piesní z dátového úložiska kufra. Umožňuje tiež tvorbu priečinkov, upravovanie a zobrazovanie metadát piesní. Metadáta sú načítavané a upravované pomocou nástroja mid3v2 a rozanalyzované a naformátované pomocou PHP skriptu.
3. **Tvorba databáz piesní** – databáza piesní je súbor uchovávajúci podmnožinu piesní, ktoré budú dostupné po zapnutí aplikácie.

Keďže RetroPie umožňuje väčšinu nastavení priamo v aplikácii, nie je potrebné duplikovať funkcionalitu do webového rozhrania. Ako jediná sprístupnená funkcia v manažéri je správca súborov na pridávanie a odstraňovanie hier.




**Obrázok 8.** Zobrazovanie informácií o Raspberry

### 4.3.2 Systémové informácie

Užívateľovi sú sprístupnené aj informácie o aktuálnej záťaži procesora (analýzou súboru **/proc/stat**), využití pamäte RAM (príkaz **free**) a taktiež aj o voľnom mieste na disku (nástroj **df**). Tieto informácie sú spracované a naformátované do jednoduchého pochopiteľného tvaru. Prebieha pravidelná aktualizácia týchto údajov.

## 5. Záver

V práci sú ukázané možnosti, ktoré majú jednoduché počítače, aj s menšou výpočetnou kapacitou a prináša riešenia na niektoré základné problémy (napr. Sekcia 3.2 o probléme pripojenia). Ďalej sa zaoberá využitím webového serveru a komunikáciou s OS.

Pri súbežne spustenom serveri a jednej z hlavných aplikácií je procesor vyťažený priemerne na 65 %. To ukazuje, že aj mikropočítače dokážu obsluhovať naraz viacero aplikácií a užívateľov a sú vhodné aj pre niektoré náročnejšie aplikácie. Celková aktuálna pamäť Raspberry je 32GB, z čoho 25 % zaberá operačný systém a samotné aplikácie. V prípade nedostatku miesta, nutnosti pripojenia na internet alebo rozšírenia hry z dvoch na 4 hráčov, je užívateľovi umožnené pripojenie externého zariadenia pomocou USB.

Práca využíva existujúce riešenia dvoch nosných aplikácií a vytvára im autentické prostredie. Spája moderné technológie a dizajn webového rozhrania s retro hardwarovou výbavou a vzhľadom aplikácií.

V rámci ďalšej práce by som sa chcel venovať dokončeniu webového manažéra, opraveniu niektorých jeho bezpečnostných dier a grafického rozhrania. Z hardwarovej časti chýba izolácia elektrického rozvodu. Ako komerčné riešenie by bolo možné hmotnosť kufra odľahčiť zmenou monitora a vloženie jednotného elektrického zdroja s rôznym napätím (v mojej práci by to však radikálne navýšilo náklady). Všetky zdrojové kódy a presná implementácia budú zverejnené v GitHub repozitári až bude hotová.

## Poďakovanie

Chcel by som poďakovať vedúcemu mojej práce Prof. Ing. Adamovi Heroutovi, Ph.D. a môjmu otcovi Danielovi Erlichovi za ich pomoc.

## Literatúra

[1] Simon Monk. *Raspberry Pi Cookbook - Software and Hardware problems and solutions*. O'Reilly Media Inc., 2014. ISBN: 1449365221.

[2] Steve Krug. *Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability*. Pearson, 2014. ISBN: 9780321965516.

[3] Derek Molloy. *Exploring Raspberry Pi – Interfacing to the real world with embedded Linux*. John Wiley Sons, Inc., 2016. ISBN: 9781119188681.

[4] GPIO – Raspberry Pi Documentation. https://www.raspberrypi.org/documentation/usage/gpio/.

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

# Improving robustness of neural networks against adversarial examples

Martin Gaňo*

**Abstract**
The main goal of this work is to design and implement the framework that yields robust neural network model against whatever adversarial attack, while result models accuracy is not significantly lower comparing to naturally trained model. Our approach is to minimize maximization the loss function of the target model. Related work and our experiments lead us to the usage of Projected gradient descent method as a reference attack, therefore, we train against data generated by PGD. As a result, using the framework we can reach accuracy more than 90% against sophisticated adversarial attacks on MNIST dataset. The greatest contribution of this work is an implementation of adversarial attacks and defences against them because there misses any public implementation.

**Keywords:** Neural networks — Optimization — Machine learning

**Supplementary Material:** Downloadable Code

*xganom00@fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

## 1. Introduction

Professionals are worried for a long time about using neural networks in safety-critical applications. Safety-critical systems are employed in many areas including transport industries, medicine and defence [1], for instance, autonomous cars or operating robots. The reason of concern is the fact that neural networks contain an enormous number of trainable parameters and that's why they are too complex for human comprehension, in other words, they are so-called black-boxes for humans. Because of that, it is infeasible to explain or fix a misclassification, or another kind of error caused by seemingly faultless model. A significant danger comes with simple methods for creating adversarial examples. An adversarial example is an almost indistinguishable input from the original sample, however, it is misclassified [2]. This way can enemy affect the result of model prediction (breaking face detection, fooling autonomous car, etc...) and manipulate the target application. In some extreme cases, this kind of attack can result in fatal consequences like serious endangerment humans health, life or property. It will never be possible to resist all kinds of attacks used by enemies. They keep improving their criminal strategies and will probably never stop. However, we have no other choice, but to continue on this research, which leads to improving safety and robustness of all machine learning models.

The main contribution of this work is the framework that enables to perform adversarial training for any neural network architecture with any dataset and that is important for experimenting with adversarial attacks and defences to make progress in this field of research. There's a strong need for such a tool because there's no available public implementation of defence methods for general usage. We must mention frame-

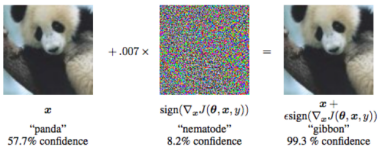



**Figure 1.** Little perturbations can cause misclassification of a model with high confidence. Adapted from [2].

work *Cleverhans* with implemented attacks, however, this work currently misses defence methods. Madry et al. provided an implementation of adversarial training for some concrete model both for CIFAR10 and MNIST datasets [3] but this is also not sufficient for ML community and it's the reason why we are presenting the general implementation of adversarial training for any model in Python language using high-level library *Keras*.

Notice, that the defence against adversarial attacks (or adversaries) differs according to a type of attack. Related work and our experiments gave us hypothesis, that we can train a model that is robust to all attacks (wrt. the distance between adversarial example and natural sample), which are using only *first-order* information [3]. In other words, we are now able to create model resistant to the whole class of attacks. We have experimentally proven the hypothesis for few white-box attacks implemented by Ian Goodfellow et al.[1] and our implementation of Generative adversarial network designed for generating adversarial samples called AdvGAN [4]. All these attacks were not able to decrease the network's accuracy under 90%. There are no other methods of attack that *first-order* used in this work.

## 2. Background

This section contains definitions and explanations of important terms used in this paper.

### 2.1 Adversarial example

Informally, it is a sample, which is almost indistinguishable from the original sample for human, while the model misclassified it. A valid explanation is also that it is a little perturbed original data, that are misclassified by the target model. We need also a formal and unified definition of adversarial example. We need a formal definition of adversarial example.

**Definition 1** *Adversarial example is such n-dimensional vector $x^{adv}$ that there exists such n-dimensional vector $x$ in set of original data $X$ that $D(x^{adv}, x) < \varepsilon$, while $t(x^{adv}) \neq C$, where $D$ is some metric, $\varepsilon$ is a small constant, $t$ is target model and $C$ is correct class of $x$.*

Typical metrics used in previous work are $L^\infty$ and $L^2$ distances since they properly simulate similarity for humans [3].

$L^\infty$ metric, sometimes called *Chebyshev distance* between two vectors is defined as the greatest value of their differences along any coordinate dimension.

$$d_{chebyshev}(x, y) = \max_i(abs(x_i, y_i)) \tag{1}$$

$L^2$ metrics is also called Euclidean distance.

$$d_{euclidean}(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{2}$$

where $n$ is the dimension of the vector.

### 2.2 Robustness

The general definition of robustness is the ability of a system to handle perturbations or errors that might affect it without changing the initial configuration.

There's a need to adapt this definition for the problem of robust optimization. Theoretically, it would be really simple to define robustness using the definition of adversarial examples this way: *The model is robust if it is not possible to generate any adversarial example*. This definition would be useless in most practical cases because only models with 100% accuracy are robust according to the mentioned definition. Our definition must cover this case, but also must to be much weaker to make it feasible to realize it.

**Definition 2** *An optimal model is robust if it stays optimal under any allowed perturbation of input data.*

Definition 2 is valid but we need a more specific one for the final evaluation of robustness.

**Definition 3** *Let $a_{nat}$ be an accuracy of optimal target model $t$ evaluated on natural data and $a_{adv}$ is an accuracy of $t$ evaluated on adversarial examples $X^{adv}$ generated by any method, while for each sample $X_i^{adv}$ of $X^{adv}$ exists $X_j^{nat}$ such that $d(X_i^{adv}, X_j^{nat}) < \varepsilon$, where $d$ is some metric and $\varepsilon$ is small constant distance. Then $t$ is robust if $a_{nat} - a_{adv} < c$, where $c$ is constant value.*

### 2.3 Transferability phenomenon

Many black-box methods are based on so-called *transferability phenomenon*. According to [5] are adversarial examples transferable between neural network architectures i.e. black-box attacks often rely on that

[1] https://github.com/Tensorflow/cleverhans

generating malignant samples against the target model will be also harmful to another one.

According to Madry et al. [3] it is also useful for the attacker to choose suitable model because training only against strongest models doesn't guarantee that the adversarial samples will be most transferable as they could.

### 2.4 Adversarial attack

Generally, we understand finding adversarial examples as a *constrained optimization problem*. For this paper, we are defining adversarial attacks as functions with a variable amount of parameters and the function returns adversarial examples (see the subsection 2.1). The number of attack parameters differs according to the type of attack and used method. It is common for attacks to take as an argument real data with their labels - this is an essential variable for any kind of attack. Sometimes the original data is used for training adversarial models (e.g. Adversarial generative networks) or in other cases the output data are deriving directly from the original data (e.g. Fast gradient sign method). Another possible parameter is the target model. This is often (but not necessary) used, because we could design our target model and expect that the real target model will behave similar way (see 2.3). We distinguish two types of adversarial methods, according to information about the model that is passed as an argument. If adversarial functions don't access to target models parameters, the method is called **black-box**. The second type is called **white-box** attacks. We consider attack as a white-box when the enemy has access to all model parameters and the whole model structure.

## 3. Related work

Szegedy at al. in 2014 [6] demonstrated few properties of neural networks, including but not limited to fact that one adversarial sample is often misclassified by few models, neural networks are vulnerable to adversarial examples and interesting phenomenon that some adversarial examples (for some dataset) are indistinguishable for human. Our understanding danger of adversarial examples in neural networks is based on these observations.

After that Ian Goodfellow et al. explained adversarial examples in [2]. They showed strategies that could generate adversarial samples to fool even the most precise neural networks with really little computation sources. Mentioned work proposed method like FGSM and Basic Iterative Method, that we adapted for our experiments and our extended implementation of these methods is an important part of the framework presented in this work.

Framework for safety verification of neural networks was proposed by Huang et al. [7]. The framework guarantees to find all existing adversarial examples by exhaustive search around the region of original sample implemented by technologies like Z3.

Research by Kurakin et al. [8] brought us a few ideas about adversarial training including a summary of attacks. They demonstrated the transferability phenomenon, label leaking and difference between adversarial training using one-step attacks and iterative methods (with many steps). This will be explained later in this paper.

We adapted approach to adversarial training introduced by a group of scientist from MIT in their paper [3]. In this work, they proposed hypothesis, that if the neural network is robust against PGD (Projected gradient descent) attack, it is also robust against every other *first-order* attack. This statement is experimentally proven in the mentioned paper and also in my work against several adversarial attacks. The mentioned paper also studies the relationship between network architecture and capacity with robustness. They show, that the larger networks handle adversarial examples better than smaller networks with similar accuracy on natural data. The result of their work is a model that achieves more than 89% accuracy against state-of-the-art attacks (on MNIST data).

The method that attacked most successfully robust model trained using defence by Madry is extended implementation of Generative adversarial networks (GAN) [4]. GAN was proposed by Ian Goodfellow et al. [9]. Authors created a framework called *AdvGAN* for perturbing original data to adversarial examples.

## 4. Adversarial methods

The term Adversarial attack was explained in subsection 2.4 and we are using terms adversarial method and adversarial attack with the same meaning. In this section, I will describe the methods used for creating adversarial examples and evaluating the robustness of the target model. All methods use only *first-order* information.

### 4.1 Fast gradient sign method

The method was firstly proposed by Ian Goodfellow et al. in 2015. It is based on computing an adversarial image by adding a perturbation with magnitude one pixel in the direction of the gradient, thus this method is certainly white-box attack. The formula for computing

result sample is

$$X^{adv} = X + \varepsilon * sign(\nabla_x L(X,Y)) \tag{3}$$

where $X^{adv}$ is adversarial sample, $X$ is original sample, $\varepsilon$ is size of perturbation, $L$ is loss function, $Y$ is label of original data and $sign(x)$ is function that returns $-1$ if $x < 0$; 0 if $x = 0$ and 1 if $x > 0$.

The method is very efficient in terms of computation time because it is computed just with one single step. [8]

### 4.2 Basic iterative method

The basic iterative method is a straightforward extension of FGSM 4.1, that is applied in iterations with small step size and we need to clip pixel values after each step - this ensures that they are in $\varepsilon$-ball of original data sample [10]. As $X_n^{adv}$ we denote adversarial example computing with n iterations. To compute $X_n^{adv}$ we first need to set $X_0^{adv}$ as follows:a

$$X_0^{adv} = X \tag{4}$$

and to compute $X_n^{adv}$ we use this formula:

$$X_n^{adv} = clip_{X,\varepsilon}\{X_{n-1}^{adv} + \alpha * sign(\nabla_x L(X_{n-1}^{adv}, Y)\} \tag{5}$$

where $sign(x)$ is function with same meaning as in Equation 4.1 and $clip_{X,\varepsilon}(x^{pert})$ is a function that returns $x^{pert}$ if $D(X, x^{pert}) < \varepsilon$, otherwise it returns nearest point to $x^{pert}$ that is within $\varepsilon$-ball around $X$.

This method also requires access to model parameters, so we consider it as a white-box attack.

### 4.3 Projected gradient descent

Projected gradient descent (PGD) is the most successful white-box method mentioned in this work. Cause this is a key method for the model robustness, it will be described more detailed and with special attention. Similarly, as methods mentioned above, we are using PGD to find such adversarial inputs, that maximize the loss of the model while the distance between generated data and original data according to certain metric does not exceed $\varepsilon$. Correctly chosen $\varepsilon$ (with the correctly chosen norm) can guarantee, that generated examples will be similar, or even indistinguishable, or at least imperceptibly different to original samples.

The algorithm is an extension of both FGSM and Basic iterative method. We first set $X_0^{adv}$ to random point within the $L^p$ ball (in our case $p = \infty$) with radius of $\varepsilon$ same way as in equation 4. Then we make a gradient step with size $\alpha$ in the direction of greatest loss and project back to $L^p$ ball [2] (clip) as in equation 5. Then we apply equation 5 until convergence, i.e. until the value of loss function slows down increasing from step to step [11] as you can see in the figure 2.

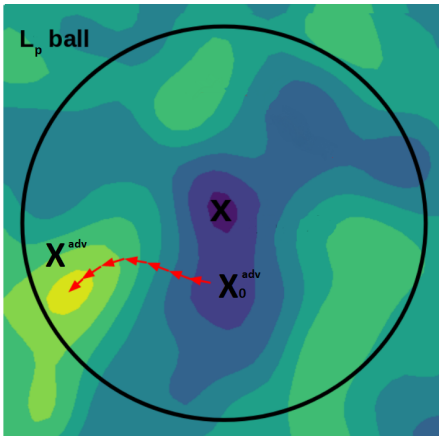



**Figure 2.** PGD after $n$ iterations found the highest value for loss function. Figure is adopted from [12].

### 4.4 Generative adversarial networks

Method firstly proposed by Ian Goodfellow et al. in 2014 [9]. The main principal of Generative adversarial networks (GAN) is that two networks are fighting against each other. These networks are usually called discriminator $D$ and generator $G$. $G$ generates samples and $D$ evaluates probability that sample comes from original data or is generated by $G$. Goal of $G$ is to maximize error rate of $D$, which is, certainly, trying to have the best accuracy in classifying samples. This leads to *saddle point* problem and to treat it through the lens of *Game theory* the GAN's trying to find Nash equilibrium. Networks $D$ and $G$ are playing 2-player *minimax game* with value function $V(G,D)$:

$$\min_G \max_D V(G,D) = \mathbb{E}_{X\sim p_{data}(X)}[log(D(X))] + \mathbb{E}_{Z\sim p_Z(Z)}[log(1 - D(G(Z)))] \tag{6}$$

#### 4.4.1 GAN for generating adversarial examples

In 2019 was proposed framework AdvGAN in [4] and took the first place in MNIST Challenge[3] in the black-box category. This is the only one *black-box* attack described and used in this paper. The purpose of this framework is to train *generator* to be able to generate perturbations from the original data - after the model is trained, there is no need to access any target model. This is also a reason why is *AdvGAN* much

[2] projecting back to $L^p$ ball means to move to the closet point inside the $L^p$ ball

[3] In the time of writing this work it is still the best submission

more effective than other adversarial attacks. When the generator is trained, we can generate perturbation for a sample using a feed-forward network, while other methods must apply gradient descent for every single sample. Implementation of AdvGAN is an important part of our framework that enables us to experiment also with this black-box attack.

#### 4.4.2 Architecture

Unlike many other GAN-based architectures, our generator does not take as an input random noise, but the original instance. Output of $G$ is a noise that is added to original sample (see Figure 3). The perturbed sample $X+G(x)$ goes into $D$ along with original sample and $D$ distinguishes if $x+G(x)$ and $x$ are similar. At the same time $x+G(x)$ takes target model $f$ as an input and returns the loss function $L_{adv}$, which represents the distance between the predicted sample and target class. The output of $D$ is the adversarial loss [9]:

$$L_{GAN} = \mathbb{E}_{X\sim p_{data}} log\, D(x) + \mathbb{E}_{X\sim p_{data}} log(1-D(x+G(x))) \quad (7)$$

The loss of target model is:

$$L_{adv} = \mathbb{E}_{X\sim p_{data}} l_f(x+G(x),t) \quad (8)$$

where $t$ is target class, $f$ is target model and $l_f$ is loss function of target model. This loss is used for a targeted attack, an untargeted attack could be performed by maximizing the distance between prediction and the sample's label. We also need to constrain the noise generated by $G$ to prevent exceeding previously defined $c$ (see Definition 1). For this purpose we use hinge loss, what is usual practise in previous work [13] (unlike referenced work we use $L^\infty$ norm instead of $L^2$). Hinge loss is formally expressed as follows:

$$L_{hinge} = \mathbb{E}_{X\sim p_{data}} max(L^\infty(G(X))-c,0) \quad (9)$$

Finally, the generator is trained on the objective function

$$L = L_{adv} + \alpha * L_{GAN} + \beta * L_{hinge} \quad (10)$$

where $\alpha$ and $\beta$ are experimentally chosen constants and represent the relative importance of loss functions.

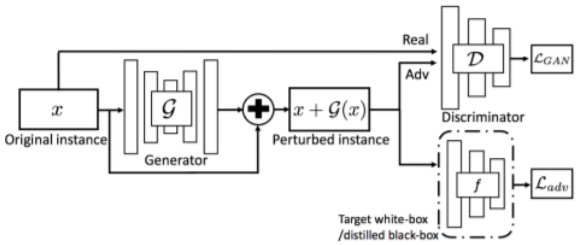



**Figure 3.** Architecture of AdvGAN. Figure is adopted from [4]

## 5. Defence

Our framework is designed to experiment with attacks and defences and it provides a full implementation of the method proposed by Madry et al. [3]. First of all, we need to present and explain a few observations that reasoned the correctness of methods in our framework.

### 5.1 Observations

Selection of parameters for our method is partially based on these observations and they are also important for building a robust model. We are providing adversarial training for any model, however, for best results user should beware these discoveries.

#### 5.1.1 Capacity of network

Madry et al. in their work [3] show that just increasing the network's capacity helps increasing robustness against adversarial examples, while the network is trained only on natural samples. We can achieve only small improvement by using better architecture on natural data, but increasing capacity could improve the accuracy of a network on adversarial examples by more than 10%.

#### 5.1.2 Label leaking

Observation by Kurakin et al. [8] that training on adversarial examples generated by single-step methods (especially FGSM) does not yield model robust against other attack strategies, because models trained on such samples use overfitting. We believe this is because transformations by single-step methods are simple and model can easily recognize them. The experiment supporting this statement is described in subsection 6.3.1.

### 5.2 Saddle point problem

Defence strategy implemented in this work is based on optimization problem called *saddle point* or *minimax* problem. A saddle point is a point on the graph of a function where the derivatives in orthogonal directions are all zero, however it is not a local extreme. We are finding such point to minimize maximization an error of our model using adversarial methods Formal expression of the task for our framework follows:

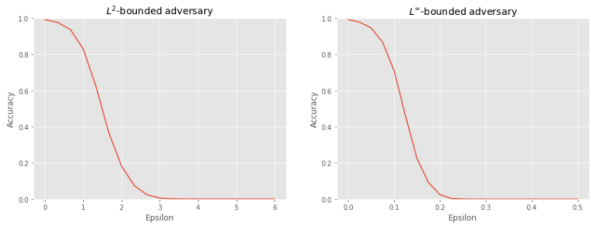



**Figure 4.** Comparison training convergence of $L^2$ and $L^\infty$ norms. Figure is adopted from [12].

$$\underset{\theta}{argmin}\ E_{(x,y)\sim D}[\underset{\delta}{max}\ L(x+\delta, y, \theta)] \tag{11}$$

where $\theta$ are model parameters, $D$ is data distribution, $L$ is loss function and $\delta$ is perturbation such that $\delta \in S$, where $S$ is set of allowed perturbations.

### 5.3 Optimization against PGD samples

The method is introduced by Madry et al. [3], they also stated that the training model against PGD attacks will satisfy robustness against all methods using only first-order information.

#### 5.3.1 Robustness wrt. *first-order attacks*

Every usage of *first-order methods* in this work will refer to *first-order methods* in **Optimization theory**. Generally, $n$-order methods is an algorithm that requires at least one $n$-th derivation/gradient. In this work are discussed only *first-order methods*.

It doesn't matter what is the starting point of PGD algorithm (we can select it randomly or set a starting point as an original sample) - the local maximum losses are almost the same in all cases. This suggests a view on the problem in which robustness against the PGD-generated samples yields robustness against all first-order samples. There is still a possibility of existence some local maximum with much higher value, but according to experiments [3] it is hard to find.

This assumption implies that we can build model robust against all first-order methods and the vast majority of optimization tasks in ML are solved using first-order, therefore the framework can yield model that is safe against state-of-the-art attacks.

## 6. Experiments

This section contains reasoning for using hyperparameters and experiments demonstrating the results of this work. Every experiment uses $\varepsilon = 0.3$ and we use only one network architecture[4] in another case it will be explicitly stated.

[4]Network architecture is defined here `github.com/sio13/turtleNet/blob/master/target_model.py`

| Dataset | Natural | PGD | FGSM | BIM |
|---|---|---|---|---|
| MNIST | 99.1% | 6.9% | 12.8% | 7.4% |
| CIFAR10 | 70.6% | 5.5% | 8.1% | 12.1% |
| CIFAR100 | 29.8% | 1.2% | 1.3% | 2.2% |

**Table 1.** Comparing three attacks on three models

### 6.1 Choosing the norm and value of $\varepsilon$

For the problem of adversarial examples are commonly used two norms - $L^2$ and $L^\infty$. According to [12] is experimentally shown that using $L^\infty$ it is possible to generate effective adversarial examples using 10 times smaller $\varepsilon$ (see Figure 4). In the Figure 4 we can also see that it is useless to use greater $\varepsilon$ than 0.3 (measured with the $L^\infty$ norm) for generating adversarial examples, and this also the reason why we are using this value of $\varepsilon$ is this work.

### 6.2 Attacking network

In this experiment, we demonstrate the vulnerability of neural networks by using three attack strategies. Target models trained on MNIST, CIFAR10 and CIFAR100 datasets achieve accuracy outstanding accuracy. By applying FGSM attack we can generate 10000 malign samples on which models achieve significantly lower accuracy and the generation process is really fast (for imagination it won't last more than 10 seconds on any laptop). On the other hand, the most successful attack was PGD, as we expected. Target models achieve an accuracy as little below 10% for all datasets. Generation process lasted about 10-times more than with FGSM. See the experiment results in Table 1.

Another experiment with adversarial attacks was more focused on MNIST trained model. For more detailed statistics see Figure 6. Notice, that the fastest attack is with method AdvGAN because the adversarial generator is already trained.

### 6.3 Comparison of target adversaries

This subsection contains experiments related to training against adversarial examples generated by some methods and comparison of its results.

#### 6.3.1 Single-step method

The first approach to train model against adversarial examples was trying to train it using FGSM examples. This method seems to work well against FGSM examples and is also effective. Since it is a one-step method and it is much faster than iterative methods. Training with these samples and a suitable number of iterations yields model that achieves against FGSM accuracy more than 80%, but against other methods, it fails (see Figure 5). This behaviour is probably caused

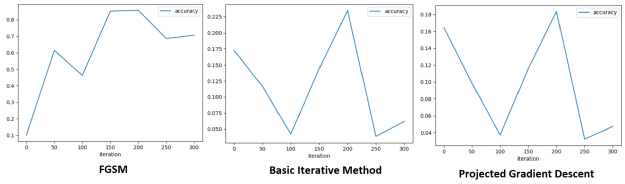



**Figure 5.** Comparison training accuracy of a model trained on FGSM examples against 3 adversarial methods.

| attack | accuracy | loss | attack_time |
|---|---|---|---|
| MomentumIterativeMethod | 0.09269999712705601 | 9.61143680114746 | 53.58076977297974 |
| MadryEtAl | 0.06849999725818601 | 12.184943614196778 | 53.06886506806274 |
| BasicIterativeMethod | 0.07029999792758 | 14.372136480712891 | 53.0767183303833 |
| ProjectedGradientDescent | 0.06840000301599501 | 12.217894227600098 | 54.56422805786133 |
| FastGradientMethod | 0.15090000629425002 | 2.641806679916381 | 5.482558012008667 |
| AdvGAN | 0.10120000529425102 | 7.848866679916381 | 2.569878012008667 |
| None | 0.9886000156402588 | 0.040568183993019995 | 0 |

**Figure 6.** Table showing damage of attacks to the naturally trained network.

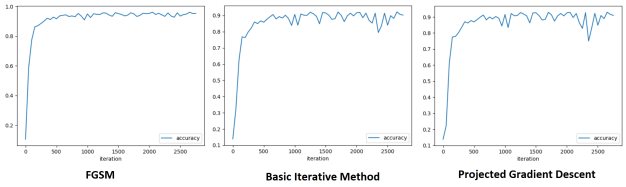



**Figure 7.** Comparison training accuracy of a model trained on PGD examples against 3 adversarial methods.

by overfitting to samples generated using a too simple method (see subsection 5.1.2).

Training against this method is just slightly better than no adversarial training 6. After about 200 iteration training with FGSM samples model achieves just less than 20%, which is not much more comparing to 6.5% achieved by naturally trained model.

### 6.3.2 PGD for training

Another and much more successful method for adversarial training is using PGD samples training. The method is much more resources consuming because it requires a high amount of iterations and every iteration requires the generation of PGD samples, but it converges to high accuracy. Using this method we can achieve accuracy more than 90% against attacks showed in Figure 7. Robustness was demonstrated also against other first-order methods, all experiments are visualized in the public repository[5].

## 7. Conclusions

Along with this work, we implemented a Python library called *turtleNet* with the implemented method for attacking and training a robust network.

[5] https://github.com/sio13/turtleNet/tree/master/result_pictures

This work also provided evidence about the vulnerability of neural networks to synthetically generated malignant samples. We also showed and implemented the solution for this issue as a part of *turtleNet* and experimentally proved that it relatively satisfies current requirements. Our implementation should be used for further experiments and for performing adversarial training on many other models. The main advantage of the framework is a possibility to make robust a model implemented using Keras or any other library. The only requirement is to convert a model into h5 format. We also provide an easy way to verify models robustness using several adversarial attacks.

Even we showed that we can face first-order adversaries, which are most common and easy to realize, there are still ways how to attack resistant model with high damage. It is almost clear that enemies will always improve their strategies and the only way how to keep this field safe is to constantly research new and better defence methods.

This research does not aim to change the world. It's main and greatest goal is to prevent changing world of machine learning and AI the way we don't want to.

### 7.0.1 Weaknesses

Our research assumes some constant value of $\varepsilon$, however, there exist samples with high distance from original data, but looks similar, therefore there is still a possibility to easily generate adversarial examples with higher $\varepsilon$, which look pretty similar to natural data and fool the mode this way.

### 7.0.2 Future work

The following research should investigate higher-order attacks and explore possibilities to fool our robust model. An idea behind *turtleNet* is to create software that unifies a large amount of attack and defence strategies for neural networks. This goal is not satisfied at all and it requires a lot of work in the future.

## Acknowledgements

I would like to thank my supervisor RNDr. Milan Češka, Ph.D. for his help.

## References

[1] Zeshan Kurd, Tim Kelly, and Jim Austin. Developing artificial neural networks for safety critical systems, 2006.

[2] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2015. https://arxiv.org/pdf/1412.6572.pdf.

[3] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks, 2019.

[4] Chaowei Xiao, Bo Li, Jun-Yan Zhu, Warren He, Mingyan Liu, and Dawn Song. Generating adversarial examples with adversarial networks, 2019.

[5] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning, 2017. https://arxiv.org/pdf/1602.02697.pdf.

[6] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, 2014. https://arxiv.org/pdf/1312.6199.pdf.

[7] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks, 2016. https://arxiv.org/pdf/1610.06940.pdf.

[8] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial machine learning at scale, 2017. https://arxiv.org/pdf/1611.01236.pdf.

[9] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, David Warde-Farley Bing Xu, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets, 2014. https://arxiv.org/pdf/1406.2661.pdf.

[10] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world, 2017. https://arxiv.org/pdf/1607.02533.pdf.

[11] Tianhang Zheng, Changyou Chen, and Kui Ren. Is pgd-adversarial training necessary? alternative training via a soft-quantization network with noisy-natural samples only, 2019. https://arxiv.org/abs/1810.05665.

[12] Oscar Knagg. Know your enemy, 2019. https://towardsdatascience.com/know-your-enemy-7f7c5038bdf3.

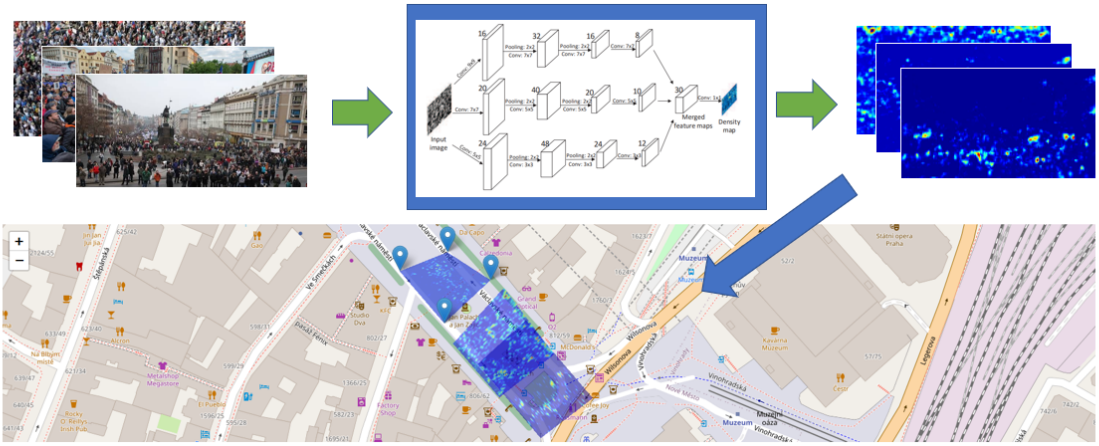
[13] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks, 2017. https://arxiv.org/pdf/1608.04644.pdf.

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

# Počítání lidí v davu pomocí neuronových sítí s integrací do mapy

Adam Ferencz



**Abstrakt**
Cílem této práce je umožnit získat věrohodný odhad počtu lidí v davu na demonstraci či jiné hromadné akci z několika fotografií pořízených dronem, či jiných fotografií. Výsledkem jsou obarvené části mapy podle hustoty lidí v daném místě. Jednotlivé fotografie jsou dány do souvislosti s jejich umístěním do topologické mapy. Pro počítání lidí z fotky je použita metoda konvoluční neuronové sítě, která dokáže k fotografii vytvořit příslušnou mapu hustoty lidí. Integrace více fotek dohromady se pak provádí transformací do mapového podkladu, čímž se vezme v potaz i rozmístění davu v prostoru.
Aplikace je rozdělena na server a webový klient. Serverová část se stará o analýzu davu a vytvoření map hustoty k jednotlivým obrázkům. Toho dosáhne pomocí natrénovaného modelu neuronové sítě Multi-Column Convolutional Neural Network. Klient se pak kromě vstupních obrázků stará o jejich zobrazení do mapy, případně modifikaci dat. Výsledkem je interaktivní mapa, ve které jsou umístěné mapy hustoty davu.
**Klíčová slova:** Počítání lidí — Počítačové vidění — Konvoluční neuronové sítě — Keras — Leaflet — OpenStreetMap

**Přiložené materiály:** Demonstrační video

*xferen05@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

## 1. Úvod

O co nejpřesnější určení celkového počtu lidí na demonstraci či na veřejné události mají velký zájem jak novináři, tak i policie. Aktuální postupy balancují mezi rychlostí a přesností. Například pomocí "ručního" počítání lze dosáhnout poměrně přesného odhadu, ale za cenu dlouhého úsilí.

Problém je v tom, že při demonstraci nelze celý dav zaznamenat na jednu fotografii. Často jsou lidé také zakryti stánky a budovami. Při získání více fotografií pak může být celý dav pokryt, ale je zde problém se souvislostí jednotlivých fotek mezi sebou. Proto je vhodné do řešení zahrnout mapu a ujasnit si tak rozložení davu v ní. Právě integrace mapy a přesné

analýzy fotek pomocí neuronové sítě může přinést věrohodný obrázek o proběhlé demonstraci.

Existující neuronové sítě jsou schopné v praxi poskytnout dobrý přehled o hustotě v dané oblasti. [1] Používají se ale především na jeden statický pohled, například pro zjištění, zda se konkrétní parkoviště nepřeplní. Většinou jejich využití končí u čísla - počtu lidí a jedné mapy hustoty. V případě využití topologických map se odhad počtu lidí v dosavadních řešeních musí odhadnout z počtu lidí na metr čtverečný a plochy, jako například u aplikace Mapchecking[1].

V mém řešení se kombinuje možnost analýzy obrázku neuronovou sítí s topografickou mapou. Neuronová síť podá údaj o počtu lidí na obrázku. Když se tento údaj vloží do mapy, získáváme z toho druhou zajíma-vou informaci, a to je, na jaké ploše se vyskytuje daný dav. Z toho lze pak odvodit hustotu davu na metr čtverečný a ne na jeden pixel, jak to je u samotného modelu neuronové sítě.

Vzniklá aplikace má především za cíl efektivně využít neuronových sítí pro počítání davu. Uživateli je umožněno pro konkrétní hromadnou akci analyzovat velikost a hustotu davu v topografické mapě.

## 2. Existující řešení v oblasti analýzy davu

V této kapitole popíši konkrétní existující aplikace pro analýzu a počítání davu. Metodám, které tyto aplikace či systémy využívají se pak bud věnovat v kapitole 3.

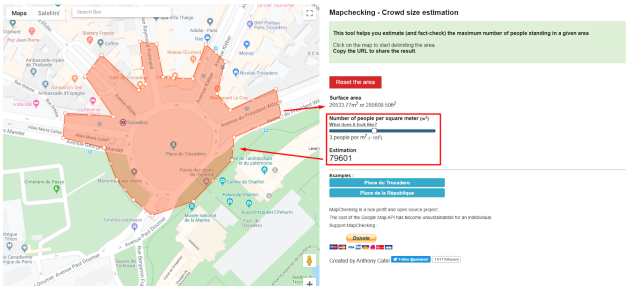
### 2.1 Mapchecking

Webová aplikace mapchecking.com (viz Obr. 1) pro odhad celkového počtu davu používá pouze mapu bez dalších složitých metod. Uživatel zadá polygonem do mapy plochu, kde se dav vyskytoval. Poté musí správně odhadnout, jaká byla hustota daného davu. K tomu by podle autora měly pomoci modelové ukázky nasimulovaných hustot davu dostupné na webu Crowd Safety and Risk Analysis [2], který se zabývá porozuměním bezpečnosti davu. Výhodou aplikace je její jednoduchost a rychlost. Problém je ale v samotné nepřesnosti a nevěrohodnosti výsledků. I když má člověk dostupné fotografie daného davu, musí se na ně pouze podívat a odhadovat, jaká je hustota.

Právě na základě tohohle problému vznikala má myšlenka zapojit do řešení neuronové sítě, které počet a hustotu lidí dokáží odhadnou s dostatečně nízkou chybou.

[1] https://www.mapchecking.com/

[2] http://www.gkstill.com/Support/crowd-density/625sm/Density6.html




**Obrázek 1.** Webová aplikace Mapchecking pro pomoc odhadu počtu lidí na demonstraci. Na obrázku vidíte označené území, kde se dav pravděpodobně nacházel. Na slideru vpravo je nastavena odhadnutá hustota davu na metr čtverečný. Z těchto dvou údajů je odvozen celkový počet lidí na demonstraci.

### 2.2 Systémy na analýzu davu

Například velký zájem o analýzu návštěvníků je v obchodních centrech, kde účelem je jak prevence bezpečnostního rizika a předpověď potřeby evakuace, tak i touha získávat informace o chování nakupujících, jejich pohlaví, atd.

Společenost **iOmniscient** nabízí řešení pro komplexní analýzu chování davu v různých scénářích. Především se soustředí na CCTV bezpečnostní kamery, prevenci přelidnění a realtime monitoring. Kromě početních výstupů nabízí možnost mapy četnosti výskytu lidí a další analytická data.

V rámci projektu LETSCROWD[3] vznikl komplexní nástroj pro dohled nad hromadnými akcemi. Ten sbírá data z bezpečnostních kamer pro předpovídání rizikových situací. Vše je konvertováno na serveru společně s dalšími informacemi jako je analýza příspěvků na sociálních sítích, hlášení policie a organizátorů. Mapa je zde použita především pro zobrazení výskytu kamer, vozidel či strážníků. Vše je zobrazováno v klientské aplikaci tak, aby vyšší autority měly dostatečný přehled o probíhající akci. Toto řešení je nejaktuálnější a stále se vyvíjející.

## 3. Metody počítání a analýzy davu z fotografie a videa

Metody řešící analýzu davu lze rozdělit na klasické a metody využívající neuronových sítí. **Klasické metody** se snaží analyzovat fotku či video pomocí detekce hran, segmentace popředí a pozadí, sledování toku pixelů, shlukováním, či hledají ručně stanovené příznaky. [5] Častým problémem těchto metod je, že jejich přesnost selhává při velmi hustých davech, nebo že potřebují video - sekvenci více po sobě jdoucích snímků ke svému fungování. Tyto problémy se snaží

[3] https://letscrowd.eu/

**Obrázek 2.** Ukázka nejvýznamnějších datesetů pro počítání davu. ShanghaiTech [2], UCF-QNRF [3], UCF_CC_50 [4] a NWPU-Crowd [1]

překonat přístupy založené na konvolučních neuronových sítích. [6]

**Přístup pomocí konvolučních neuronových sítí** je úspěšný, protože s dostatečnými datasety se model dokáže naučit rozpoznávat příznaky v různých situacích, a tak lépe generalizovat. Snižování nákladů a dostupnost zařízení pro pořizování dat ze vzduchu přinesl větší počet dostupných kvalitních obrazových datových sad. Datasety jsou anotované pomocí souřadnic bodů hlav lidí (head dot anotation), výjimečně pomocí boundingboxu. Mapa hustoty se pak generuje pro trénink pomocí konvoluce s gausovou maticí či komplexnějším algoritmem. Na obrázku 2 jsou ukázky nejvýznamnějších z nich. Aktuálně nejnovější dataset NWPU-Crowd [7] byl vydaný na začátku roku 2020, obsahuje 5109 anotovaných obrázků a slouží jako benchmark[4] pro nynější architektury. Je možné říct, že situace s datasety pro počítání davu je velmi přívětivá.

Pro účely počítání davu se ukázalo vhodné upustit od detekce konkrétních jedinců a jejich počítání. [6] Místo toho se ukázalo vhodnější mapovat vstupní data na mapy hustoty. V mapě hustoty k danému obrázku pak hodnota určuje hustotu lidí na pixel. To zlepšilo výsledky především v analýze velmi hustých davů. Pokud se ve fotce davu vyskytuje více jedinců velmi blízko u sebe či za sebou, v mapě se to projeví vyššími hodnotami v daných pixelech a není potřeba se starat o to, že se jedinci vzájemně překrývají a nelze je jednotlivě přesně detekovat.

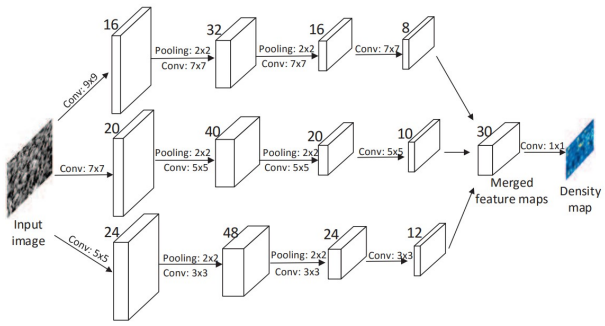
Modely byly nejdříve jednoduché s jedno sloupcovou architekturou [6], kdy jedním sloupcem je myšlena sekvence střídajících se konvolučních filtrů a max poolingu o různých parametrech. Dokázaly klasifikovat dav v daném místě obrazu na stupně hustoty od nízké po vysokou. [8] Problém, který bránil lepším výsledkům, byla perspektiva fotografie. Ta se pak projevovala rozdílnou velikostí lidských hlav, či těl. Řešením bylo přidání více různých sloupců do architektury neuronové sítě. Jejich kombinaci pro lepší výsledek lze pak provést více způsoby. Buď spojením jejich výstupu, nebo jejich přepínáním pro části obrazu.[6]

Multi-Column Convolutional Neural Network [2], zkráceně MCNN, ve své architektuře obsahuje tři sloupce neuronových vrstev (viz Obr. 3), které se liší jen svými parametry konvolučních filtrů. Výstupy sloupců jsou pak spojené a vytváří tak výstup celé neuronové sítě. Vstupní obrázek prochází současně přes všechny tři sloupce. Každý sloupec je citlivý na různou úroveň zvětšení hlav. Tři výstupy sloupců se pak dají dohromady a výsledkem je tedy mapa hustoty, která zohledňuje perspektivu fotografie.

Další možností je Switching Convolutional Neural Network [9], kde autoři natrénovali samostatně jednotlivé sloupce na datech, která jsou přesně v jejich optimálním přiblížení. Poté natrénovali přepínač, klasifikátor, který je schopný rozhodovat, jaký ze samostatně natrénovaných modelů je vhodné pro danou část obrázku použít.

Poslední z populárních architektur je využití multitask learningu, kdy je neuronová síť trénována na více různých pod-úloh. V případě počítání davu pak úlohami může být klasifikace úrovně hustoty a generování mapy hustoty, což použil Sindagi et al. [10].

[4] https://www.crowdbenchmark.com/nwpucrowd.html




**Obrázek 3.** Architektura Multi-Column Convolutional Neural Network představená Yingying Zhangem v roce 2016[2]

Ve své práci jsem použil již natrénovaný model s architekturou MCNN [2] od Priyanka Kasture[5], která model natrénovala v rámci MindSpark Hackathon 2018 a zpřístupnila[6].

### Vyhodnocení přesnosti MCNN

Úspěch neuronových sítí pro počítání davu se hodnotí pomocí dvou metrik: MEA (Mean Average Error) a MSE (Mean Absolut Error),

$$MAE = \frac{1}{N}\sum_{1}^{N}|z_i - \hat{z}_i| \tag{1}$$

$$MSE = \sqrt{\frac{1}{N}\sum_{N}^{1}(z_i - \hat{z}_i)^2} \tag{2}$$

kde N je počet testovacích obrázků, $z_i$ je opravdový počet lidí v $i$tém obrázku, a $\hat{z}_i$ je stanovený počet lidí v $i$tém obrázku. Obecně řečeno $MAE$ vypovídá o přesnosti a $MSE$ indikuje robustnost odhadů. K výpovědní síle této metriky je vždy potřeba dodat, na jakém datasetu podávala jaké výsledky. Je velmi pravděpodobné, že MCNN bude podávat nejlepší výsledky na Shanghaitech datasetu, na který byl designován. Generalizovat pak bude nejlépe na datech podobných jeho tréninku. Podle článku [2] jsem výsledky shrnul do tabulky 1. V tabulce můžete vidět výsledky MCNN v souvislosti s průměrným počtem lidí na jednom obrázku daného datatsetu. Z toho lze odvodit, jaká chyba je pro jak hustý dav očekávatelná.

V závislosti na datasetu NWPU [1] pak MCNN podává následující výsledky v porovnání s předními ostatními architekturami. (viz Tab. 2) Je podstatné říct, že NWPU dataset je aktuálně nejkomplexnější dostupný dataset.

[5]https://github.com/priyanka-kasture/Crowd-Counting-with-MCNNs

[6]https://github.com/priyanka-kasture/Crowd-Counting-with-MCNNs/blob/master/LICENSE

| Dataset | Prům. poč. | MAE | MSE |
|---|---|---|---|
| ShanghaiTech P. A | 501.4 | 110.2 | 173.2 |
| ShanghaiTech P. B | 123.6 | 26.4 | 41.3 |
| WorldExpo | 50.2 | 11.6 | - |
| UCF CC 50 | 1279.5 | 377.6 | 509.1 |
| UCSD | 24.9 | 1.07 | 1.35 |

**Tabulka 1.** Tabulka shrnující výsledky MCNN [2] na různých datasetech.

| Metoda | MAE | MSE |
|---|---|---|
| MCNN [2] | 232.5 | 714.6 |
| C3F-VGG [11] | 127.0 | 439.6 |
| CSRNet [12] | 121.3 | 387.8 |
| CANNet [13] | 106.3 | 386.5 |
| SCAR [14] | 110.0 | 495.3 |
| SFCN+ [15] | 105.7 | 424.1 |

**Tabulka 2.** Tabulka výsledků nejlepších metod na datasetu NWPU [7].

## 4. Architektura aplikace

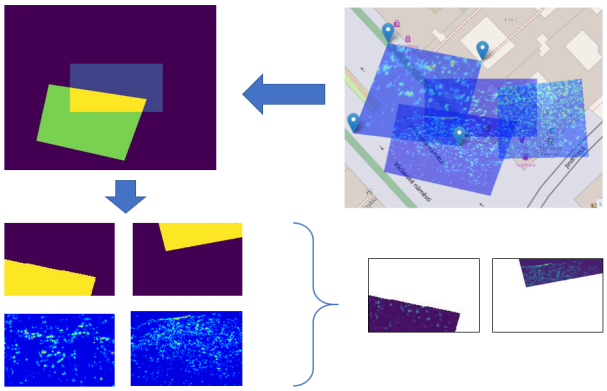
Uživatel chce spočítat celkový počet lidí na demonstraci z fotografií, které pořídil. Aplikace mu umožní nahrát a rozmístit fotografie v mapě a vypočítat hustotu davu v jednotlivých fotografiích. Pro rozmístěné fotografie je provedena korekce počtu u překrývajících se. Výsledkem je vizualizace rozmístění a počet lidí na demonstraci.

Architekturu aplikace tedy navrhuji jako klient-server s využitím webových technologií. Interakce s uživatelem bude realizována na klientovi a vlastní výpočet pomocí CNN na serveru.

Klientská část umožní nahrání fotografií a jejich umístění do mapy. Tato data je třeba synchronizovat se serverem. Ten bude počítat hustotu davu, převádět na počet osob, transformovat fotografie a provádět detekci překryvů fotografií, díky které poté vypočte redukovaný výsledný počet osob. Výsledky pak budou opět synchronizovány s klientskou částí, kde budou uživateli vizualizovány.

Model na serveru odhaduje počet lidí pro celou fotografií vytvořením mapy hustoty. V řadě případů se ale fotografie překrývají a výsledný počet lidí na akci by byl tedy zkreslený, proto je třeba provést korekci počtu osob v místě překryvů. Celkový počet by neměl být jen součet všech hodnot samostatných obrázků, ale musí brát z překryvů poměrnou část.

Je podstatné nalézt masky překryvů pro jednotlivé kombinace fotografií. Server si drží vypočtené mapy hustoty jednotlivých obrázků. Na ty se použije tato získaná maska a je možné tedy vypočítat samostatně počet lidí v místě překryvu. Všechny kombinace těchto vymaskovaných překryvů lze reprezentovat maticí, která je vizualizována na obrázku 5. V matici jsou pak



**Obrázek 4.** Schéma získáni a použití masky pro korekci překryvu.

místo map hustoty již konkrétní počty lidí v místech překryvu. Výsledný počet se pak získá pomocí vzorce:

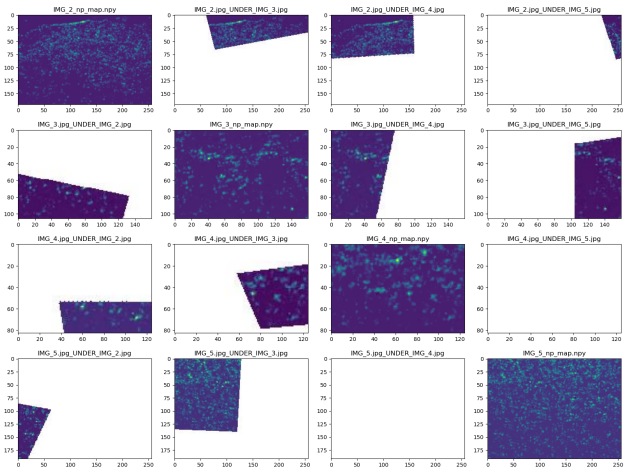
$$SUM = D - \frac{M - D}{2} = \frac{1}{2} \cdot (3 \cdot D - M) \qquad (3)$$

kde $SUM$ je odhad s korekcí, $D$ je součet prvků diagonály (zároveň také odhad před korekcí), $M$ je součet celé matice.

Aby server věděl, jak jsou fotografie umístěné, je třeba zaslat polohu všech obrázků v mapě a s nimi i hranice mapového souřadného systému, na kterém se vyskytují. Nejdříve se vytváří masky pro překryvy transformovaných obrázků. Transformovaná mapa hustoty by ale dávala zkreslený počet. Je tedy třeba provést inverzní transformaci masky tak, aby byla použitelná na originální tvar mapy hustoty. Tento postup je znázorněn na schématu 4.

Server je implementován v jazyce Python. Komunikaci s klientem zajišťuje lehký WSGI[7] webový aplikační framework Flask[8], který splňuje všechny potřeby pro nasazení natrénovaného modelu. Na server přichází požadavky pro přepočítání obrazových vstupů na mapy hustoty. Ty se pak odesílají k zobrazení zpět na klientskou stranu. Výhodou serveru je, že model se nedává na web veřejně přístupný v případě, že by ho bylo třeba udržet v soukromí. (Pokud by toto potřeba nebylo, nabízí se možnost model konvertovat do formátu, se kterým je schopná pracovat knihovna keras.js a aplikace by mohla být tak čistě klientská.) Natrénovaný model pak na serveru běží pomocí knihovny Keras. Další potřebné akce s obrazovými daty jsou pak prováděné pomocí knihovny pro počítačové vidění OpenCV a knihovny numpy.

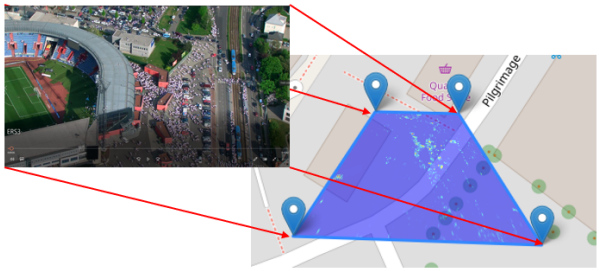
[7] Web Server Gateway Interface
[8] https://pypi.org/project/Flask/



**Obrázek 5.** Vizualizace matice jednotlivých překryvů.

Logika webového klienta je implementována v JavaScriptu. Zobrazení mapy zajišťuje Leafletjs, což je nejrozšířenější open-source JavaScriptová knihovna k tvorbě interaktivních map. Mapu lze posouvat, přibližovat, oddalovat. Je možné zvolit různé zdroje mapového podkaldu (Mapbox, OpenStreetMap, Stamen, ...), já jsem zvolil čistý OpenStreetMap.

Přesouvání obrázku a jeho transformaci v mapě (viz. Obr. 6) jsem vyřešil použitím Leaflet pluginu Leaflet.Path.Drag a pluginu Leaflet.ImageTransform.



**Obrázek 6.** Ukázka perspektivní transformace mapy hustoty davu v obrázku do mapy podle čtyř určujících bodů.

## 5. Uživatelské prostředí

Pro uživatele je zásadní, aby získání celkového počtu lidí bylo co nejsnazší a intuitivní. V této kapitole popíši, jaké akce jsou od uživatele očekávané a kroky, které jsem provedl pro zjednodušení uživatelské činnosti.

Prvním krokem je výběr a nahrání fotografií davu, se kterými se dále pracuje. Poté má uživatel za úkol umístit do mapy jednotlivé obrázky podle 4 bodů, které korespondují se čtyřmi rohovými body obrázku. Obrázky posouvá jako celek, nebo je vhodně posunem krajních bodů perspektivně transformuje. Obrázkem je myšlena fotografie, nebo k ní korespondující mapa hustoty davu. Mezi těmito pohledy uživatel může přepínat.

**Obrázek 7.** Prototyp aplikace před pilotním testem.

Uživatel vidí, kolik je lidí na jednotlivých obrázcích a kolik je tedy celkově lidí na celé akci (suma jednotlivých obrázků s korekcí překryvů).

Tím, že se umístila mapa hustoty do topografické mapy, je možné vypočítat plochu, na které se dav vyskytuje. To může být zajímavý údaj, ze kterého lze vyvodit hustotu vztaženou k metrům čtverečným.

První verze GUI, zobrazená na obrázku 7, reflektovala základní funkcionalitu aplikace. Vznikl tak MVP, se kterým byl proveden pilotní test (muž, 43 let, znalý práce s počítačem). Výstupem experimentu jsou následující požadavky na revizi GUI:

- Je potřeba, aby uživatel poznal, v jakém pořadí a proč má aplikaci ovládat. GUI musí reflektovat uživatelský proces.
- Mapa je klíčový interaktivní prvek (manipulace s fotkami), musí jí být dáno výrazně více prostoru.
- Vhodné zobrazovat vybraný obrázek a jeho mapu hustoty ve větším formátu, než jen náhledový.

Po těchto požadavcích jsem navrhl 5 různých nových layoutů ve formě makety, ty jsem s účastníkem pilotního testu znovu konzultoval a výsledné finální rozvržení implementoval (viz Obr. 8).

GUI je rozděleno na dva sloupce. V levém, který zabírá 60 procent celého prostředí, je zobrazena mapa. V pravém sloupci se nachází ovládací prvky. Od shora je zde umístěn výběr pro načtení fotografií, procházení náhledů načtených fotografií, velký náhled aktuální vybrané fotografie, velký náhled její mapy hustoty a na závěr informace o fotografii a celém projektu. Prvky v ovládání jsou seřazeny od shora dolů tak, jak se s nimi bude popořadě pracovat.

## 6. Závěr

Cílem práce bylo vytvořit aplikaci, která umožní spočítat lidi na hromadné akci s využitím topologické mapy, fotografií a neuronové sítě.

Své řešení jsem navrhl jako klient-server aplikaci. Server umožňuje nasazení natrénovaného modelu neuronové sítě s architekturou MCNN a komunikuje s

**Obrázek 8.** Výsledné GUI po pilotním testu.

klientem pomocí HTTP protokolu. Klient zajišťuje vstup dat, interakci s mapou a vizualizaci výsledku. Aplikace je schopná vzít v potaz překryvy umístěných fotografií.

Vzhledem k tomu, že téma mého projektu vzniklo jako jeden z pod-úkolů fakultního projektu VRASSEO, který se obecně zabývá analýzou davu, je v plánu rozšířit tuto demonstrační aplikaci o další moduly, například detekci anomálie v davu. V budoucnu by se kromě statických fotografií měla do mapy dát vizualizovat především videa z dronů. Takové video je schopné pokrýt větší území a poskytnout mnoho dalších informací, jako je nebezpečné chování, rychlost davu, případně i detekce nebezpečných osob a jejich následná lokalizace v mapě. Bezpečnostní složky by pak měly možnost analyzovat situaci v reálném čase a lépe před-cházet nepříjemným situacím.

## Poděkování

Velmi děkuji panu Ing. Vítězslavu Beranovi Ph.D. za jeho rady, které mi pomohly při tvorbě této práce.

## Literatura

[1] Qi Wang, Junyu Gao, Wei Lin, and Xuelong Li. Nwpu-crowd: A large-scale benchmark for crowd counting. *arXiv preprint arXiv:2001.03360*, 2020.

[2] Yingying Zhang, Desen Zhou, Siqin Chen, Shenghua Gao, and Yi Ma. Single-image crowd counting via multi-column convolutional neural network. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[3] Haroon Idrees, Muhmmad Tayyab, Kishan Athrey, Dong Zhang, Somaya Al-Maadeed, Nasir Rajpoot, and Mubarak Shah. Composition loss

for counting, density map estimation and localization in dense crowds. In *The European Conference on Computer Vision (ECCV)*, September 2018.

[4] Haroon Idrees, Imran Saleemi, Cody Seibert, and Mubarak Shah. Multi-source multi-scale counting in extremely dense crowd images. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013.

[5] Sami Abdulla Mohsen Saleh, Shahrel Azmin Suandi, and Haidi Ibrahim. Recent survey on crowd density estimation and counting for visual surveillance. *Engineering Applications of Artificial Intelligence*, 41:103 – 114, 2015.

[6] Vishwanath A. Sindagi and Vishal M. Patel. A survey of recent advances in cnn-based single image crowd counting and density estimation. *Pattern Recognition Letters*, 107:3 – 16, 2018. Video Surveillance-oriented Biometrics.

[7] Qi Wang, Junyu Gao, Wei Lin, and Xuelong Li. Nwpu-crowd: A large-scale benchmark for crowd counting, 2020.

[8] Min Fu, Pei Xu, Xudong Li, Qihe Liu, Mao Ye, and Ce Zhu. Fast crowd density estimation with convolutional neural networks. *Engineering Applications of Artificial Intelligence*, 43:81 – 88, 2015.

[9] Deepak Babu Sam, Shiv Surya, and R. Venkatesh Babu. Switching convolutional neural network for crowd counting. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[10] Vishwanath A Sindagi and Vishal M Patel. Cnn-based cascaded multi-task learning of high-level prior and density estimation for crowd counting. In *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 1–6. IEEE, 2017.

[11] Junyu Gao, Wei Lin, Bin Zhao, Dong Wang, Chenyu Gao, and Jun Wen. C^ 3 framework: An open-source pytorch code for crowd counting. *arXiv preprint arXiv:1907.02724*, 2019.

[12] Yuhong Li, Xiaofan Zhang, and Deming Chen. Csrnet: Dilated convolutional neural networks for understanding the highly congested scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1091–1100, 2018.

[13] Weizhe Liu, Mathieu Salzmann, and Pascal Fua. Context-aware crowd counting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5099–5108, 2019.

[14] Junyu Gao, Qi Wang, and Yuan Yuan. Scar: Spatial-/channel-wise attention regression networks for crowd counting. *Neurocomputing*, 363:1–8, 2019.

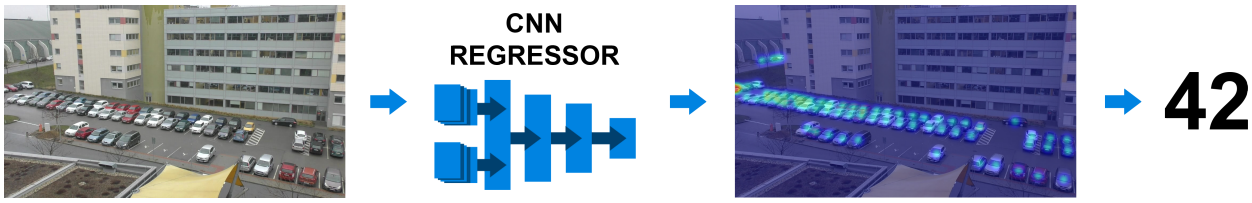
[15] Qi Wang, Junyu Gao, Wei Lin, and Yuan Yuan. Learning from synthetic data for crowd counting in the wild. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8198–8207, 2019.

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

# Counting Vehicles in Static Images

Ondřej Zemánek




**Abstract**
This paper addresses the problem of counting vehicles in static images with no geometric information of the scene. Four convolutional neural network architectures were studied, implemented and trained as a main part of this work. Also, a dataset that consists of 19 310 images in total from 12 views that captures 7 different scenes were taken as part of this work. The trained networks map the appearance of the input sample to its corresponding vehicles density map, which can be easily translated to the vehicle count with keeping the localization of the vehicles in the input image. The main contribution of this work is in an application and a comparison of the state-of-the-art solutions to the problem of object counting. Most of them were mainly designed to count pedestrians in crowded scenes or for medicine images, so the major goal was to adapt these solutions for vehicle counting task. The implemented models were trained on TRANCOS dataset which is a popular benchmark for counting vehicles on annotated low quality highway pictures. Their performance is compared and the results are discussed.

**Keywords:** visual counting, vehicle counting in static images, car park dataset

**Supplementary Material:** *N/A*

*xzeman53@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

## 1. Introduction

Visual counting that aims to accurately estimate the number of vehicles is a hard problem. But its potential is huge in many applications across many industries.

For instance, it can help truck drivers, who need to plan their next break, by monitoring parking capacity near highways. Another application can be long-term analysis of a traffic density on main city roads and highways, so road closures, detours or road expansion can be planned easily and smoothly. Also, solving the vehicle counting problem can bring a cheaper solution for monitoring shopping center parking lots, so instead of using a physical sensor for each parking space, a few cameras can be used to monitor the parking lot.

The most recent state-of-the-art solutions are based mainly on the convolutional neural network model. Therefore, this work is focused only on these approaches. The best approaches can be divided into objects detection approaches and density map regression based approaches.

The first group uses classification of individual objects in YOLO-like (You Look Only Once) [1] style to detect and count the objects in the input image. Although these approaches can be fast and reliable in trivial cases, in very dense and overcrowded scenes like images with overlapping objects, low-resolution, partly visible objects, images with slightly unseen perspective, the overall performance of these models is limited.

The other group of solutions that reaches much better results in the target scenarios is based on a different idea. Instead of solving this problem by detection of

each object in the image, these solutions transform the visual counting task into object density map estimation from the input image. In other words, they are using convolutional neural networks to transform an input image appearance into an object density map in a certain resolution. From the output of this transformation, the object count can be easily estimated by the output density map integration even with keeping the information of the objects localization.

The main contribution of this work is an application and a comparison of the existing solutions on parking lot dataset. To achieve this, I had to analyse the existing convolutional neural architectures, adapt these models to vehicle counting problem, create a large and diverse dataset for training, train them in with various parameters, and finally, evaluate them.

## 2. Implemented Architectures

The state-of-the-art approaches for visual counting are mainly built upon one of two concepts: density map regression and detection. Thus, the chosen architectures are designed in this way.

First three solutions are based on Density map regression. So, the input image is regressed into density map that represents spatial distribution of objects. Then, this map is integrated into a object count which corresponds to the input sample.

The last studied approach is somewhere in the middle of these two concepts. It combines density map regression with classification. More in subsection 2.4.

Tensorflow open source platform was used to implement these convolutional neural networks. The final models of the Counting CNN, the Hydra CNN and Spatial Division and Conquer Network were inspired by the original authors implementations, that were implemented in the Caffe framework and the PyTorch platform respectively. The authors of Stacked Hourglass model provides only brief description of the implementation, so the network implementation is based on this description only.

Despite the fact that the authors shares the implementation details, the training process implementation is tied its application and the dataset. Thus, the training process for each used network has to be created manually based on the deep analysis of the architecture.

### 2.1 From Human Pose Estimation to Visual Counting

The Stacked Hourglass architecture proposed by Newell at al. [2] is a composition of multiple modules called hourglass. Each Hourglass module processes

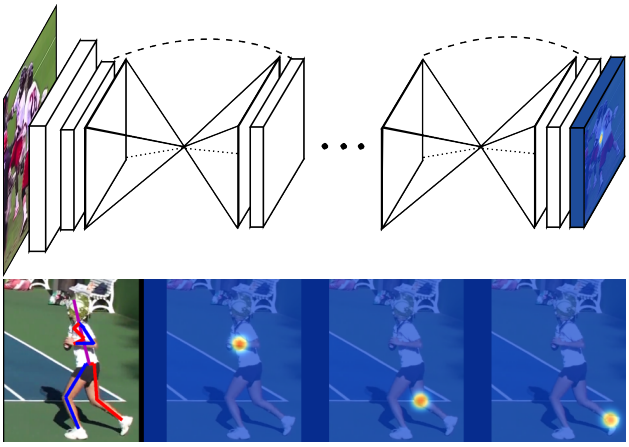


**Figure 1. top:** Stacked hourglass concept – input image is passed through multiple hourglass modules. Each hourglass model creates an intermediate prediction to improve the final result. **bottom:** Output of the multiple-level hourglass model for Human pose estimation problem [2].

the input features on multiple scales and consolidates them to best capture the object landmarks. This is done by repeated up-down/bottom-up process. Also, it uses an intermediate supervision process, i.e. the architecture uses intermediate prediction to improve the next prediction. This is done by skipping layers. This model (Figure 1) aims on the problem of human pose estimation, but as it is shown in the achieved results (Sec. 5), it also shows good results in the visual counting problem.

### 2.2 Single-Pipeline CNN with Great Results

The Counting convolutional neural network created by Oñoro et al. [3] is a simple sequence of 6 convolutional layers and two max-pooling layers, as can be seen in Figure 2. The input image patch with size $72 \times 72$ pixels is processed by this sequence and it returns a density map with $18 \times 18$ pixels size which represents the object spatial distribution in the image. Finally, the object count is gathered by integrating the density map.

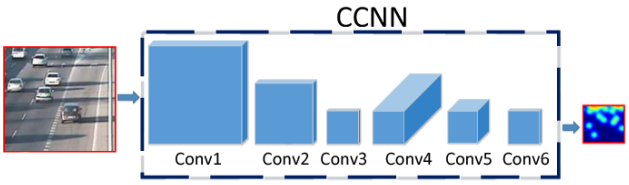



**Figure 2.** Counting CNN architecture – sequence of 6 convolutional layers in an up-down order [3].

### 2.3 Even Better Results with Multi-Scale Inputs

Next implemented architecture is called Hydra CNN, like the mythological creature Hydra with nine heads

and a big body. It is mainly based on multiple Counting CNN's that are used as the input heads of the creature as illustrated in Figure 3. Each head processes the input sample on a different scale, so the final architecture is a scale-aware solution for visual counting. The patch scale (crop ratio of the original sample) for head $H_i$ is defined as follows,

$$H_s = 1 - \frac{1}{C} \cdot H_i, \tag{1}$$

where $C$ is the number of heads. In case of the first level head, the input patch corresponds to the original sample. The heads' intermediate outputs are fused by three fully connected layers, so even the input array contains the image in multiple scales, the output is a single density map like in the case of Counting CNN. The concept can be easily extended by adding a new head or simplified by removing one. The results are much better than in the case of the simple counting CNN.

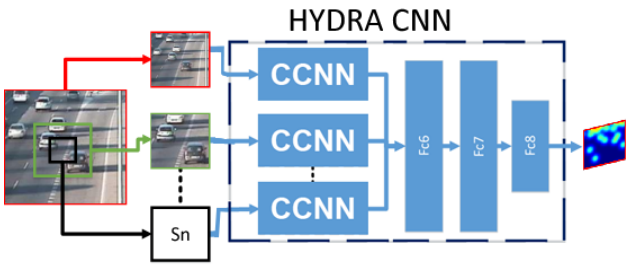



**Figure 3.** Hydra CNN scheme – input sample is up-sampled multiple times and processed by the Counting CNNs. The sub-results are merged into a single output map by fully connected layers.

## 2.4 Open-Set to Closed-Set Transformation as the State of the Art

Previously described methods are modeled in a regression manner. Xiong et al. [4] proposed a new different approach with their Spatial Divide-and-Conquer Network (S-DCNet) that is more complex and it uses multiple modules with different purposes. The main idea is spatial division of the input image into small regions, each with a closed set of a defined range, so they can transform quantity to intervals which the network can easily classify.

It is based on the main idea that the visual counting problem is an open-set problem by nature. But only limited and closed set labeled counts can be observed in reality. So the goal is to transform the original problem into a close-set one.

This is done by spatial division of the image until the count of every part is in a specified range. For example, the experimented range for vehicle counting I

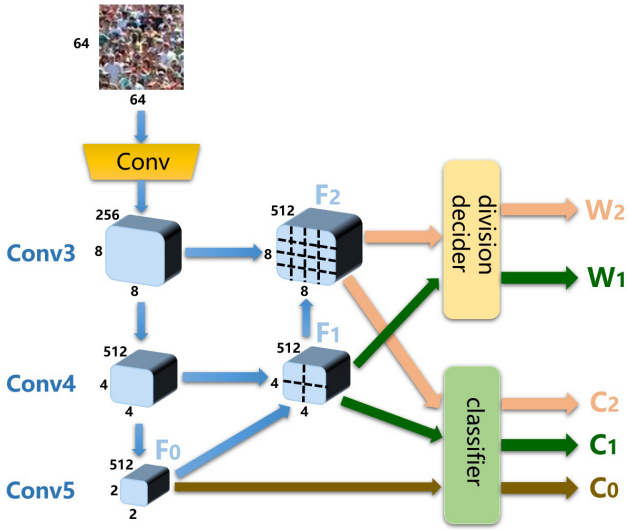



**Figure 4.** The architecture of Spatial division-and-conquer network. The first 5 layers are convolutional and follow the VGG16 concept [5]. These layers are than connected to the classifier and division decider straight forward or preceded by fusion with another layer. Finally, the output weights and classification are processed to get the object density map.

am using is $[0,5]$. That means the architecture spatially divides the input image, until every part reaches 5 object at max.

The division decider in Figure 4 is trained to decide whether it is necessary to divide the input or not. It returns $W_i$ weights in range $[0,1]$, where greater value means higher need for division. The weight is than used to compute the division result $DIV_i$. So, higher resolution count maps $C_i$ are applied if the division weight $W_i$ is higher.

Next, the classifier module predicts the object counts $C_i$ for each output feature map. The classification is done on specified intervals in the closed-set range, i.e. for each feature map, a single object count is obtained.

After the model prediction, following post-process is applied:

$$DIV_i = (1 - W_i) \circ avg(C_{i-1}) + W_i \circ C_i, \tag{2}$$

where "$\circ$" denotes the Hadamard product and $avg$ is an averaging re-distribution operator. Finally, predicted object count is represented by last $DIV_i$ map. The $i$ level corresponds to max division time which is the value that limits the input image division. So, if the $i$ is 2, than the model takes into account up-to 2 times divided input image.

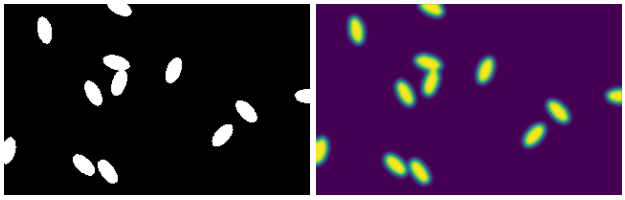


**Figure 5.** Toy dataset example of input sample (left) and output density map (right).

## 3. Training Details

The authors of the proposed solutions are proving their network performance with pretrained model that has been successfully trained on available benchmark dataset. Unfortunately, they do not always share training details.

For instance, the paper about proposed Spatial division-and-conquer architecture contains very vague information about the training process. Therefore, to train this model, it was necessary to understand the architecture in depth and experiment with model parameters.

### 3.1 Iterative Training Process With a Toy Dataset

To train the models on large and diverse dataset, it is necessary to know how the models performs with different training settings and parameters. As the fastest approach to make the training process work in the way how it was designed is training on a simplified, "toy" dataset.

Toy dataset, as can be seen in Figure 5 is a custom generated dataset which can be easily created in a short time. Its parameters, like Gaussian blur variance $\sigma$, resolution and object shape, can be modified for each convolutional neural network architecture. The main benefit of this simple dataset is that the training process is much faster than training on real images.

### 3.2 Output Activation and Loss Functions

Even with a fast training process, there is another important factor influencing training success. That is the combination of the last activation function and the loss function. The last activation function gives the transformation of the linear output value and the loss function is used to compute the trained model error.

Unfortunately, there is no general-purpose combination of these functions. So, as the implemented architectures do not have the same output format, it was necessary to understand the model pipeline and decide which combination is the best for each output.

For classification problem is common to use sigmoid or soft-max function as the last activation function and some type of cross-entropy function. Also, There are multiple cross-entropy loss functions for different purpose, like multi-class or binary classification.

For regression problem, it is typical to use linear function as the activation function and use L1-norm (mean absolute error) or L2-norm (mean square error) as loss functions (density map). Otherwise, the activation function can be set to sigmoid with use of quadratic loss function so the regressed value is in range $[0, 1]$ (weight, normalized output)[1].

The Counting CNN, Hydra-CNN model and the Stacked Hourglass model are using a combination of linear activation function and mean square error loss function.

The Spatial Division-and-Conquer model is much more complex. The network output consist of the division weights and quantity interval classifications. The division weight is a regression to values between 1 and 0, so the combination of sigmoid activation function and mean square error loss function are implemented. The count interval problem is a multi-class classification, so the soft-max activation function with mean absolute error loss function is applied.

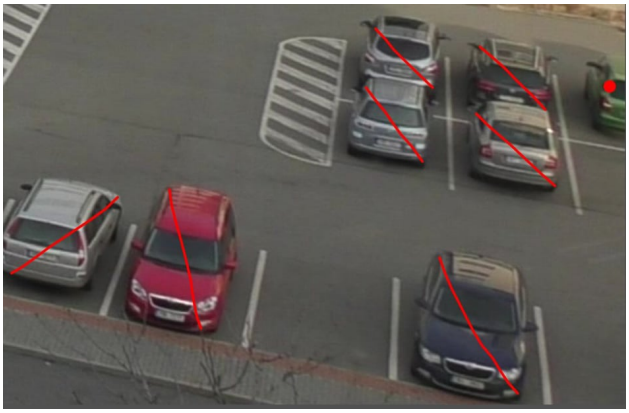


**Figure 6.** Example of the labeling style. Scribbles are used for high-resolution vehicles and dots for partially visible cars or vehicles in distant.

### 3.3 Ground Truth Labeling

For ground truth labels is used dotted annotation blurred by 2D Gaussian function. So, the vehicles in images are labeled by only single dot in first step. Then the dotted map is then blurred with Gaussian blur. Even after blurring the dots out, an integration of the blurred map still corresponds to the vehicle count. During the training process this map is used as the ground truth density map.

## 4. Diverse and Robust Training Dataset

To train the implemented network to count vehicles in images, it is crucial to have a big enough dataset

[1] https://medium.com/@phuctrt/189815343d3f

**Figure 7.** Custom dataset samples.

of images with similar parameters. As the parameters depend mainly on the final application, it is necessary to define it.

Parking lot occupancy monitoring was chosen as the main application. Therefore, the dataset images should capture parking areas or similar places like highways or streets with cars.

### 4.1 Existing Datasets

Currently, there are three suitable datasets for this application. The first one is the TRaffic ANd COngestionS (TRANCOS) [6] dataset with more than thousand labeled low-quality images from highway. The next dataset is called CNRPark+EXT[2] and it captures occupancy of parking lots with roughly 4 300 labeled images. Lastly, the CARPK[3] is a collection of drone images of huge parking lots with 1 500 annotated images where only a part of it is suitable for our application.

### 4.2 Newly Created Dataset

As the main goal of this work is an robust real-world application for vehicle counting problem, we need much more diverse and robust dataset to train the networks. Therefore, a new and more diverse parking lot dataset was collected as part of this work. Figure 7 shows examples of this custom dataset. It consist of 19 310 images in total from 12 views that capture 7 different scenes.

[2]http://www.cnrpark.it/
[3]https://lafi.github.io/LPN/

Each location was captured from a similar angle to the ground to simulate the common monitoring cam position. The recording process took place from September to March, so diverse weather and lighting conditions were captured. Also, three online webcams were recorded as part of the custom dataset. This adds another few thousands of images to this dataset.

### 4.3 Annotation

The training cannot be done without the ground truth labels for the dataset pictures. Several annotation styles were tested to label the images, like bounding box, silhouette, scribble. Although the bounding box annotation is common approach for object detection and silhouette annotation is even more precise, these two label styles are too time-consuming and unnecessary for our application. Thus, the faster and sufficient scribble and dots labeling styles were chosen as can be seen in Figure 6.

So far, more than 3 500 images were annotated as part of this work and the labeling process still continues.

## 5. Achieved Results

The presented networks were trained on the TRANCOS dataset to demonstrate their performance so far. The dataset contains training, validation and test sets, so the results can be accurately compared on samples that were not used for training. Visual comparison of the trained models on TRANCOS dataset can be seen in Figure 8.

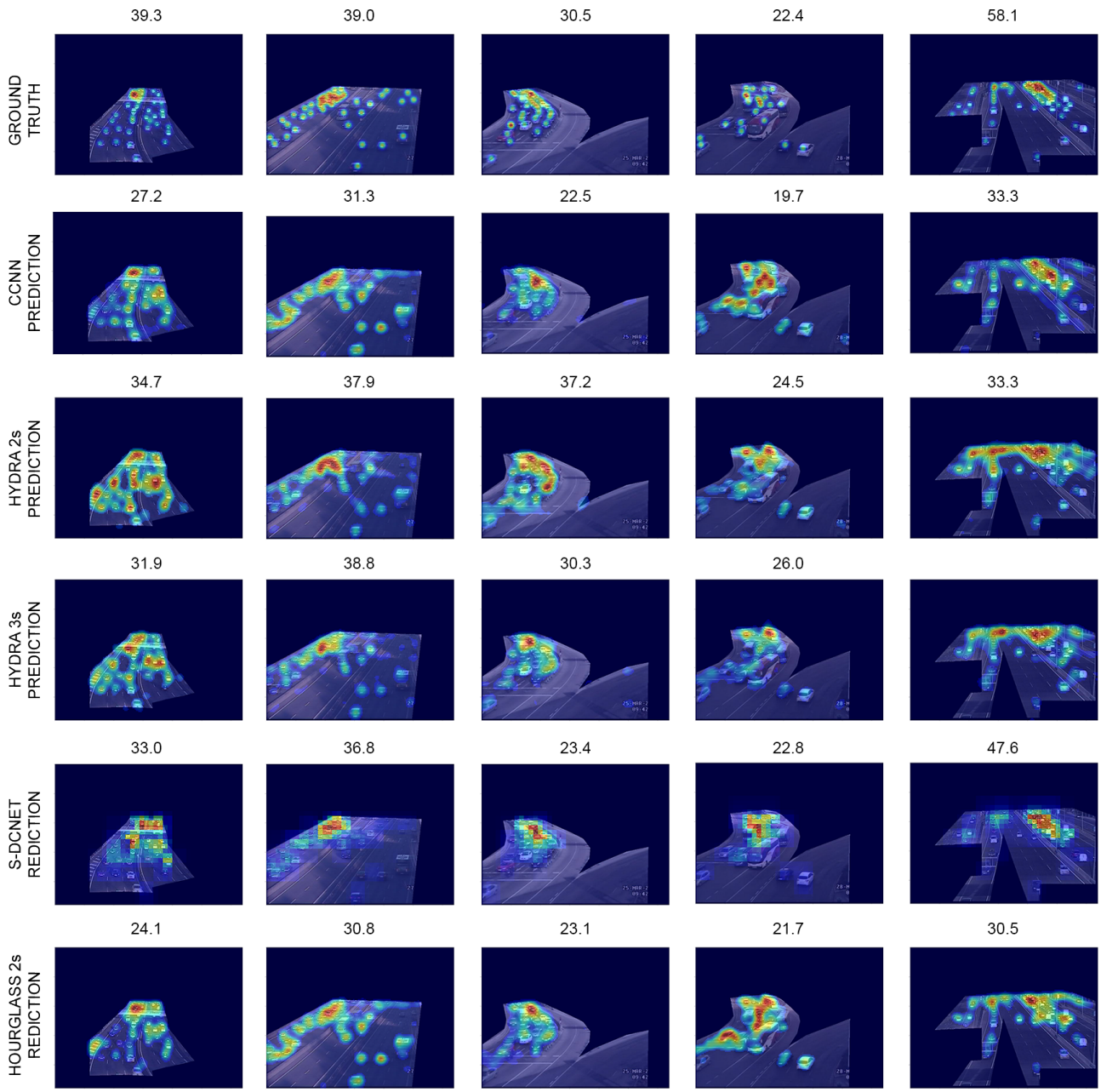



**Figure 8.** Five test samples from TRANCOS dataset with trained models predictions. Top row corresponds to the target image with ground-truth. "Hydra 2s", "Hydra 3s", "Hourglass 2s" stands for the Hydra CNN with 2 heads, 3 heads and the Stacked Hourglass model with 2 stacks respectively. The ground-truth counts are slightly different because Gaussian blur was applied to the label maps.

The comparison of the used architectures on full TRANCOS dataset can be seen in table 1. The evaluation was done with Grid Average Mean Absolute Error (GAME) [7] metric defined by

$$GAME(L) = \frac{1}{N}\sum_{n=1}^{N}(\sum_{l=1}^{4^L}|C^l_{pre} - C^l_{gt}|), \qquad (3)$$

where N denotes the number of images, $C^l_{pre}$ and $C^l_{gt}$ are the predicted and ground-truth count of the L-th subregion, respectively.

The Counting CNN achieves good results in total count prediction, but in higher levels on GAME

| Method | GAME 0 | GAME 1 | GAME 2 | GAME 3 |
|---|---|---|---|---|
| CCNN | 12.18 | 16.44 | 20.35 | 22.97 |
| Hydra 2s | 10.77 | 14.28 | 17.69 | 21.13 |
| Hydra 3s | 11.02 | 14.37 | 17.01 | 20.64 |
| SHG 2s | 14.30 | 15.84 | 18.23 | 22.81 |
| S-DCNet | **8.56** | **9.357** | **10.40** | **11.83** |

**Table 1.** Trained model evaluation with GAME metric on the TRANCOS dataset. The best performance is in boldface. "SHG 2s" stands for the Stacked Hourglass model with 2 stacks

metric shows some false prediction. In case of the Hydra CNN the results are better but the problem with noisy prediction is remains. The best predictions gives the Spatial Divide-and-Conquer Network, which has accurate prediction across all GAME levels, so the spatial prediction is very precise. Lastly, the Stacked hourglass model with 2 stacks shows great results in spatial prediction, but the total density of the predicted map is lower than ground-truth.

Next step in comparison of these architectures is the custom dataset evaluation with the GAME metric. However, the training process is still in progress and I don't want to present temporally results.

## 6. Conclusions

In this paper, I have shown four different architectures for visual counting that has been implemented, described the training details of these models. Also, The custom car park dataset with 19 300 images and 3 500 already labeled pictures have been presented.

The architectures were evaluated on the popular TRANCOS dataset and the results were shown in last chapter.

The newly created dataset is being continually updated with new locations and more importantly it is being labeled.

Also extended version of the Spatial division and conquer network has been recently released by the authors and I am finishing the implementation of this network.

Finally, all the presented networks are being trained on the custom dataset and will be evaluated.

## Acknowledgements

I would like to thank my supervisor prof. Ing. Adam Herout, Ph.D. for great support and constructive criticism during my work on this paper. Also, I would like to thank to Ing. Jakub Špaňhel for his advice and help with a dataset acquisition.

Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015-042), is greatly appreciated.

## References

[1] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[2] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *European conference on computer vision*, pages 483–499. Springer, 2016.

[3] Daniel Onoro-Rubio and Roberto J López-Sastre. Towards perspective-free object counting with deep learning. In *European Conference on Computer Vision*, pages 615–629. Springer, 2016.

[4] Haipeng Xiong, Hao Lu, Chengxin Liu, Liang Liu, Zhiguo Cao, and Chunhua Shen. From open set to closed set: Counting objects by spatial divide-and-conquer. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 8362–8371, 2019.

[5] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[6] Roberto López-Sastre Saturnino Maldonado Bascón Ricardo Guerrero-Gómez-Olmedo, Beatriz Torre-Jiménez and Daniel Oñoro-Rubio. Extremely overlapping vehicle counting. In *Iberian Conference on Pattern Recognition and Image Analysis (IbPRIA)*, 2015.

[7] Ricardo Guerrero-Gómez-Olmedo, Beatriz Torre-Jiménez, Roberto López-Sastre, Saturnino Maldonado-Bascón, and Daniel Onoro-Rubio. Extremely overlapping vehicle counting. In *Iberian Conference on Pattern Recognition and Image Analysis*, pages 423–431. Springer, 2015.

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

# Inclusion of Regular Expressions with Counting

David Mikšaník*

**Abstract**
We present an algorithm solving the inclusion problem for regular expressions with the counting operator limited to character classes, the so-called extended regular expressions (eREs), which are common in practice. Such regular expressions do not extend expressiveness beyond regularity, but allow one to succinctly express repeated patterns. Our algorithm is based on the transformation eREs into monadic counting automata (MCAs), i.e., finite automata with counting loops on character class where each counter is bounded. Similarly to the classical algorithm, we transform eREs into automata, but now we use MCAs instead of nondeterministic finite automata (NFAs). Following by building the product of MCAs and searching for a final state in the product. MCAs are compact representation of eREs because the number of states in MCAs does not depend on the bounds used in the counting operator, in contrast to NFAs where the number of states grows linearly. These bounds can be large in practice, thus MCAs are often significantly smaller than NFAs. We provide several examples for which the classical algorithm working with NFAs does not terminate in a reasonable amount of time, but our algorithm does. We also hope that our algorithm outperforms the classical algorithm in general, especially if the bounds of the counting operators are large.

**Keywords:** regular expressions — language inclusion — finite automata — counting automata

**Supplementary Material:** *N/A*

*xmiksa05@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

## 1. Introduction

Regular expressions with the counting operator limited to character classes, we called them *extended regular expressions* (eREs), have the same expressive power as the standard regular expressions (REs). The purpose of eREs is a succinct expression of some regular expressions such as `[abc]{5,10}` representing all strings of the length 5–10 where each symbol of the string is `a`, `b`, or `c`. Usually the eREs also contain other well-known operators, e.g., `+` or `?`, but such operators do not bring the succinctness as the counting operator. Sometimes the restriction of appearing counting operator only in the character classes is omitted (e.g., `(abc){5,10}` denoting all strings where `abc` appears 5–10 times). In [1], it was observed that in practice the regular expressions mostly use only counting operator limited to character classes (e.g., in the Snort rules [2] used for finding attacks in network traffic; or in the RegExLib library [3], which collects expressions for recognizing URIs, markup code to name a few). For these reasons, we limit ourselves only to eREs with counting operator limited to character classes.

Although there are several automata models that are capable of representing regular expressions in as succinct way as the eREs (e.g., [1, 4, 5]), for the best of our knowledge all algorithms solving the inclusion problem for eREs work with NFAs—eREs are transforming to NFAs directly or from any of the model above. In each case the advantage of the compact representation is lost. In this work we develop an algorithm that avoids such transformation to NFAs.

The inclusion problem for REs consists of two inputs REs $r_1$, $r_2$ and the question whether the language that $r_1$ denotes is included in the language that $r_2$ denotes, symbolically we need to decide whether $\mathcal{L}(r_1) \subseteq \mathcal{L}(r_2)$. The language of regular expressions is defined as usual. The inclusion problem for eREs is the same as the problem above except that $r_1, r_2$ can contain the counting operator limited to character

classes. This problem can arise in any Intrusion Detection System (IDS). Suppose the following scenario. Let $\alpha$ be eRE that representing all packets in network traffic that are possible dangerous (or interesting). But you want to detect more packets that are dangerous. Thus you construct an eRE $\beta$. Because you want to be sure that $\beta$ is as strong as $\alpha$, you need test whether $\mathcal{L}(\alpha) \subseteq \mathcal{L}(\beta)$. If it is true, then you know that the replacement $\alpha$ by $\beta$ is safe.

General solutions to the inclusion problem for eREs start with constructing NFAs $N_1, N_2$ for each of the input eRE. Then is used the algorithm for testing language inclusion for the NFAs. How was proposed in [6], these algorithms can be categorized into two types: (1) those based on *subset construction* build a product automaton $N_1 \times \overline{N_2}$ of $N_1$ and the complement of $N_2$ and search for a final state; (2) those based on *simulation* (e.g., [7]) first compute a simulation relation on $N_1 \cup N_2$ and then check if all initial states of $N_1$ can be simulated by some initial state of $N_2$. Usually (2) is better than (1) because the computation of simulation is done in polynomial time. But the main drawback of (2) is that it is incomplete—simulation implies language inclusion, but not vice-versa. Contrary, the methods based on the subset construction are complete, but the complementation of $N_2$ requires that $N_2$ is deterministic, which can easily exponentially blow up in the number of states.

As was mentioned above, there are several automata models for succinct representation of eREs. From these models we choose one, called *monadic counting automata* (MCAs), which are introduced in [1, section 4.1]. Informally, they are finite automata with bounded counters. Our algorithm for the inclusion problem for eREs works similar as the methods based on the subset construction. First, we transform the input eREs into MCAs $M_1, M_2$. Second, we create $\overline{M_2}$, the complement of $M_2$, by determinizing and completing $M_2$ and then complementing the accepting condition. Third, we build the product automaton $M_1 \times \overline{M_2}$ and search for a final state. We note that [1] provide the algorithm for determinization of MCAs. The problem is how to efficiently test whether a state is reachable from the initial states. Such a test is not as easy as for NFAs, because the next move of the MCA does not depend only on the input symbol, but also on the values of counters. The main contribution of this paper is the algorithm for testing reachability of states in the product automaton (see Section 3).

The advantages of our algorithm are showing when the eREs contain a lot of counting operators and the upper bound of the counting operator is large. For example, if $r_1 = \texttt{.*a.\{k\}}, r_2 = \texttt{.*a.\{k-1,k+1\}}$ the methods based on NFAs do not terminate in a reasonable time for relatively small values of $k$, but if we use MCAs, then the problem is solvable for such values of $k$. Eventually, our algorithm converges to the methods based on the subset construction if the eREs do not contain any counting operator. In this case, it is appropriate to use some method based on simulation relation.

## 2. Counting Automata

A finite, non-empty set $\Sigma$ of *symbols* is called an *alphabet*. A *string* is a sequence of symbols $a_1 a_2 \ldots a_n$ where $a_i \in \Sigma$, for $1 \leq i \leq n$. The *length* of $w$ is defined as $|w| = n$. We use $\varepsilon \notin \Sigma$ to denote the *empty string*, so $|\varepsilon| = 0$. The set of all strings over the alphabet $\Sigma$ is denoted by $\Sigma^*$.

The abstract syntax of the extended regular expressions (eREs) is the following:

$$R ::= \emptyset \mid \varepsilon \mid \sigma \mid R_1 R_2 \mid R_1 + R_2 \mid R^* \mid \sigma\{m,n\}$$

where $\sigma$ is a predicate denoting a set of alphabet symbols, and $n, m \in \mathbb{N}$. The semantics is defined as in the standard regular expressions, with $\sigma\{n,m\}$ denoting a string $w$ with $n \leq |w| \leq m$, where $n \leq m$, symbols each of them satisfying $\sigma$.

Monadic counting automata (MCAs) naturally arise from eREs (see Figure 1). They are a restriction of a more general concept called counting automata (CAs). In this section, we provide necessary theory background and give the definitions of CAs and MCAs, following [1].

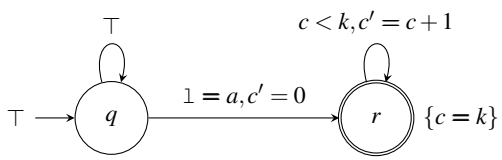



**Figure 1.** A CA for the eRE $\texttt{.*a.\{k\}}$, which is also an MCA with $k \in \mathbb{N}$, $I : \texttt{s} = q$, $F : \texttt{s} = r \wedge c = k$, and $\Delta : q \xrightarrow{\{\top,\top,\top\}} q \vee q \xrightarrow{\{1=a,\top,c'=0\}} r \vee r \xrightarrow{\{\top,c<k,c'=c+1\}} r$.

### 2.1 Logical background

Let $V$ be a set of variables $V$, and let $Q$ be a set of constants (disjoint with the set of natural numbers $\mathbb{N}$ including zero), which will correspond to the set of states in CAs. We define a *Q-formula over V* to be a quantifier-free formula $\varphi$ of Presburger arithmetic [8, Section 3.3 ] extended with constants from $Q$ and $\Sigma$, i.e., a Boolean combination of (in-)equalities $t_1 = t_2$ or $t_1 \leq t_2$ where $t_1$ and $t_2$ are constructed using $+, \mathbb{N}$, and $V$, and predicates of the form $x = a$ or $x = q$ for $x \in V, a \in \Sigma$, and $q \in Q$.

An assignment $M$ to free variables of $\varphi$ is a *model* of $\varphi$, denoted as $M \models \varphi$, if it makes $\varphi$ true. We use $sat(\varphi)$ to denote that $\varphi$ has a model and we say that $\varphi$ is *satisfiable*. The *semantics* of a formula $\varphi$ is the set $[\![\varphi]\!]$ of all possible tuples of the free variables in $\varphi$ which make $\varphi$ true. Suppose that $C = \{c_1, \ldots, c_n\}$ are the free variables in $\varphi$. The *projection* of $\varphi$ on $C$ is the formula $\exists c_1, \ldots, c_n : \varphi$.

## 2.2 Labelled Transitions Systems

A *labelled transition system* (LTS) over $\Sigma$ is a five-tuple $T = (Q, V, I, F, \Delta)$ where

- $Q$ is a finite set of *control states*,
- $V$ is a finite set of *configuration variables*,
- $I$ is the *initial Q-formula* over $V$,
- $F$ is the *final Q-formula* over $V$, and
- $\Delta$ is the *transition Q-formula* over $V \cup V' \cup \{\mathtt{l}\}$ with $V' = \{x' \mid x \in V\}, V \cap V' = \emptyset$ , and $\mathtt{l} \notin V$.

We call $\mathtt{l}$ the symbol variable and allow it as the only term that can occur with a predicate $\mathtt{l} = a$ for $a \in \Sigma$, called an *atomic symbol guard*. Moreover, $\mathtt{l}$ is also not allowed to occur in any other predicates in $\Delta$.

A *configuration* of LTS $T$ is a function $\alpha : V \to \mathbb{N} \cup Q$ that maps every configuration variable to a number from $\mathbb{N}$ or a state from $Q$. We will denote by $\mathcal{C}$ the set of all configuration of the LTS $T$. The transition relation $[\![\Delta]\!] \subseteq \mathcal{C} \times \Sigma \times \mathcal{C}$ is encoded by the transition formula $\Delta$ as follows $(\alpha, a, \alpha') \in [\![\Delta]\!]$ iff $\alpha \cup \{x' \mapsto k \mid \alpha'(x) = k\} \cup \{\mathtt{l} \mapsto a\} \models \Delta$. For a string $w \in \Sigma^*$, we define inductively that a configuration $\alpha'$ is a *w-successor* of $\alpha$, written $\alpha \xrightarrow{w} \alpha'$, such that $\alpha \xrightarrow{\varepsilon} \alpha$ for all $\alpha \in \mathcal{C}$, and $\alpha \xrightarrow{av}$ iff $\alpha \xrightarrow{a} \overline{\alpha} \xrightarrow{v} \alpha'$ for some configuration $\overline{\alpha}$, $a \in \Sigma$, and $v \in \Sigma^*$. A configuration $\alpha$ is initial if $\alpha \models I$, and final if $\alpha \models F$. The outcome of $T$ on a word $w$ is the set $out_T(w)$ of all $w$-successors of the initial configurations, and $w$ is accepted by $T$ if $out_T(w)$ contains a final configuration. The *language* $\mathcal{L}(T)$ of $T$ is the set of all words that $T$ accepts.

## 2.3 Counting Automata

A *(nondeterministic) counting automaton* (CA) is a five-tuple $N = (Q, C, I, F, \Delta)$ such that $(Q, V, I, F, \Delta)$ is an LTS with the following properties:

1. The set of configuration variables $V = C \cup \{\mathtt{s}\}$ consists of a set of counters $C$ and a single control state variable $\mathtt{s}$ such that $\mathtt{s} \notin C$.
2. The transition formula $\Delta$ is a disjunction of *transitions*, which are conjunctions of the form $(\mathtt{s} = q) \wedge \sigma \wedge g \wedge f \wedge (\mathtt{s}' = r)$, denoted by $q \text{-}\{\sigma, g, f\}\!\rightarrow r$, where $q, r \in Q$, $\sigma$ is the transition's *guard formula* over $\{\mathtt{l}\}$, $g$ is the transition's *guard formula* over $V$, and $f$ is the transition's *counter assignment formula*, a conjunction of atomic assignments to counters in which every counter is assigned at most once.
3. There is a constant $\boldsymbol{max}_N \in \mathbb{N}$ such that no counter can ever grow above that value.

Moreover, for every transition $\varphi = q \text{-}\{\sigma, g, f\}\!\rightarrow r \in \Delta$, we define the function *sym* that returns the transition's guard formula over $\{\mathtt{l}\}$, that is $sym(\varphi) := \sigma$.

A *deterministic counting automaton* (DCA) is a CA $N$ where $I$ has at most one model and, for every $a \in \Sigma$, every reachable configuration $\alpha$ has at most one $a$-successor. A CA is *complete* if for any configuration $\alpha \in \mathcal{C}$ and every symbol $a \in \Sigma$ the $a$-successor $\alpha' \in \mathcal{C}$.

## 2.4 Monadic Counting Automata

A (nondeterministic) *monadic counting automaton* (MCA) is a CA $M = (Q, C, I, F, \Delta)$ where the following holds:

1. The set of control states is $Q = Q_s \uplus Q_c$, where $Q_s$ is a set of *simple states* and $Q_c$ is a set of *counting states*.
2. The set of counters $C = \{c_q \mid q \in Q_c\}$ consists of a unique counter $c_q$ for every counting state $q \in Q_c$.
3. All transitions containing counter guards or updates must be incident with a counting state in the following manner. Every counting state $q \in Q_c$ has a single *increment transition*, a self-loop $q \text{-}\{\sigma, c_q < \boldsymbol{max}_q, c_q' = c_q + 1\}\!\rightarrow q$ with the value of $c_q$ limited by the bound $\boldsymbol{max}_q$ of $q$, and possibly several *entry transitions* of the form $r \text{-}\{\sigma, \top, c_q' = 0\}\!\rightarrow q$, which set $c_q$ to 0. As for *exit transitions*, every counting states is either *exact* or *range* where exact counting states have exit transitions of the form $q \text{-}\{\sigma, c_q = \boldsymbol{max}_q, \top\}\!\rightarrow s$ and range counting state have exit transitions of the from $q \text{-}\{\sigma, \top, \top\}\!\rightarrow s$ with $s \in Q$ such that $s \neq q$.
4. The initial condition $I$ is of the form
$$I : \bigvee_{q \in Q_s^I} \mathtt{s} = q \vee \bigvee_{q \in Q_c^I} (\mathtt{s} = q \wedge c_q = 0)$$
for some sets of initial simple and counting states $Q_s^I \subseteq Q_s$ and $Q_c^I \subseteq Q_c$, respectively.
5. The final condition $F$ is of the form
$$F : \bigvee_{q \in Q_s^F \cup Q_r^F} \mathtt{s} = q \vee \bigvee_{q \in Q_e^F} (\mathtt{s} = q \wedge c_q = max_q)$$
where $Q_s^F \subseteq Q_s$ is a set of simple final states, $Q_r^F \subseteq Q_r$ is a set o final range counting states, and $Q_e^F \subseteq Q_e$ is a set of final exact counting states.

## 3. The Inclusion Problem for Extended Regular Expressions

Let $r_1, r_2$ be the input eREs of the inclusion problem. Our solution starts with creating MCAs $M_1, M_2$ such that $\mathcal{L}(r_1) = \mathcal{L}(M_1)$ and $\mathcal{L}(r_2) = \mathcal{L}(M_2)$. We need to decide whether $\mathcal{L}(M_1) \subseteq \mathcal{L}(M_2)$, or equivalently $\mathcal{L}(M_1) \cap \overline{\mathcal{L}(M_2)} = \mathcal{L}(M_1) \cap \mathcal{L}(\overline{M_2}) = \emptyset$. For convenience, the automaton that recognize such a language is called the *product automaton*, written $M_1 \times \overline{M_2}$.

Traditionally, to complement $M_2$ we need first determinize $M_2$. Since the determinization of an MCA is not again an MCA, in Section 3.1 following [1], we give a brief overview how the states in such automaton are represented. In Sections 3.2–3.4, we present our algorithm for the inclusion problem for eREs, namely how to build $M_1 \times \overline{M_2}$ and how to test whether $\mathcal{L}(M_1) \cap \mathcal{L}(\overline{M_2}) = \emptyset$.

### 3.1 Determinization of Monadic Counting Automata

The crucial step in building the product automaton is the determinization of $M_2$. Fortunately, [1, Section 4.2] provides the algorithm for determinization of MCAs. We note that the result automaton of the algorithm is not again an MCA, but the automaton has still somewhat restricted structure as we show in Section 3.4. For that reason, all MCAs that are determinized by the algorithm in [1, Section 4.2] are called *determinized* MCAs (DMCAs).

Each state of a DMCA is represented by the notion of *sphere* [1]:

$$\Psi := \bigvee_{q \in Q'_s} \mathtt{s} = q \vee \bigvee_{q \in Q'_c} \left(\mathtt{s} = q \wedge \bigvee_{0 \le i \le \boldsymbol{max}'_q} c_q = c_q[i]\right) \tag{1}$$

for some $Q'_s \subseteq Q_s$, $Q'_c \subseteq Q_c$, and $\boldsymbol{max}'_q \le \boldsymbol{max}_q$. That is, a sphere $\Psi$ records which states may be reached in the original MCA when $\Psi$ is reached in the determinized MCA and also which variants of the counter $c_q$ may record the value of $c_q$ when $q$ is reached.

The spheres can be also represented by a multiset of states. By a slight abuse of notation, we use $\Psi$ for the sphere itself as well as for its multiset representation $\Psi : Q \to \mathbb{N}$. The fact that $\Psi(q) > 0$ means that $q$ is present in the sphere, i.e., $\mathtt{s} = q$ is a predicate in (1), and for a counting state $q$, the counters $c_q[0], \ldots, c_q[\Psi(q) - 1]$ are the $\Psi(q)$ variants $c_q$ tracked in the sphere. i.e., $\boldsymbol{max}'_q = \Psi(q)$ in (1).

### 3.2 Product Construction of Monadic Counting Automata

First, we need to compute $\overline{M_2}$, the complement of $M_2$. Since we know how to determinize MCAs, it remains to complete the determinization of MCA. Let $D = (Q^D, C^D, I^D, F^D, \Delta^D)$ be a DMCA with the same language as $M_2$. The completion of D is the following: we add a new non-final state $q_{sink}$ and the transition $q_{sink} \text{-}\{\top,\top,\top\}\!\!\to q_{sink}$. For every state $q$, let $P_q = \{\varphi \mid q \text{-}\{\varphi\}\!\!\to r \in \Delta^D\}$. Then, for every state $q \neq q_{sink}$ we add a new transition $q \text{-}\{\psi\}\!\!\to q_{sink}$ where $\psi = \wedge_{\varphi \in P} \neg\varphi$. Intuitively, if no outgoing transition from $q$ can be executed, then we can use this new added one. For this reason the procedure preserves the determinism. Also the langauge of $D$ is preserved because $q_{sink}$ is not a final state. To finish to complementation of $M_2$ it is sufficient to complement the final condition $F^D$ of $D$ by simply putting $\neg F^D$. We define the function *complement* that takes an MCA and produces a DMCA as described above.

Second, we build the product automaton $N = M_1 \times \overline{M_2}$ that recognizes the language $\mathcal{L}(M_1) \cap \overline{\mathcal{L}(M_2)}$ similarly as for NFAs. The states of $N = (Q, C, I, F, \Delta)$ are pairs $(q, R)$, where $q \in Q_1$ and $R \in Q^D$, i.e., $Q \subseteq Q_1 \times Q^D$. The set of counters of $N$ is $C \subseteq C_1 \cup C^D$ (some counter might not be needed if its corresponding state is not reachable, see below the function *ground*). The initial formula $I$ of $N$ labels pairs of states as initial if both states are also initial in $M_1$ and $\overline{M_2}$, respectively. Formally, we transform $I = I_1 \wedge I^D$ into disjunctive normal formal such that each part of disjuncts of the form $\mathtt{s} = q \wedge \mathtt{s} = R$ is replaced by $\mathtt{s} = (q, R)$. The initial value of counters are then the combinations of initial values of $M_1$ and $\overline{M_2}$. This transformation is denoted by $dnf$, so $I = dnf(I_1 \wedge I^D)$. The final formula $F$ of $N$ is computed analogously. The transition formula $\Delta$ of $N$ is computed as follows: let $(q, R)$ be a reachable state (at the start all initial states are reachable). We combine the outgoing transitions from $q$ and $R$, that is if $\varphi = q \text{-}\{\omega\}\!\!\to q' \in \Delta_1$, $\psi = R \text{-}\{\Omega\}\!\!\to R' \in \Delta^D$ and $sym(\varphi) \wedge sym(\psi)$ is satisfiable, then we add a new transition $(q, R) \text{-}\{\omega \wedge \Omega\}\!\!\to (q', R')$ to $\Delta$ and the pair $(q', R')$ is marked as reachable. After we process all combinations of outgoing transitions, we pick other reachable state, but each state is process at most once, thus the algorithm terminates. We note that the generated states might not be reachable in real (see Section 3.3), because we do not only combine the transitions $\varphi \in \Delta_1$ and $\psi \in \Delta^D$ for which $sym(\varphi) \wedge sym(\psi)$ is unsatisfiable and completely ignore the guards on the counters, which can cause that the transitions are unsatisfiable—this is purpose of the next section.

The whole procedure of the product construction is summarized in Algorithm 1, where the function *ground* on Line 14 removes from $C_1 \cup \neg C^D$ all counters that do not appear in any guards of $I$ and $\Delta$ and the function *ground* on Line 15 removes predicates from $dnf(F_1 \wedge \neg F^D)$ that contain states that are not in $Q$ or counters that are not in $C$.

**Algorithm 1:** Product of MCAs

**Input** : MCAs $M_1 = (Q_1, C_1, I_1, F_1, \Delta_1)$, $M_2 = (Q_2, C_2, I_2, F_2, \Delta_2)$ with $Q_1 \cap Q_2 = C_1 \cap C_2 = \emptyset$

**Output** : An CA $N = M_1 \times \overline{M_2}$ such that $\mathcal{L}(N) = \mathcal{L}(M_1) \cap \overline{\mathcal{L}(M_2)}$

1 $(Q^D, C^D, I^D, \neg F^D, \Delta^D) \leftarrow complement(M_2)$;
2 $I \leftarrow dnf(I_1 \wedge I^D)$; $\Delta \leftarrow \emptyset$;
3 $Q \leftarrow \{(q,R) \mid \mathrm{s} = (q,R) \text{ appears in } I\}$;
4 $W \leftarrow Q$;
5 **while** $W \neq \emptyset$ **do**
6 pick and remove $(q,R)$ from $W$;
7 **foreach** $q \text{-}\{\varphi\}\!\rightarrow q' \in \Delta_1$, $R \text{-}\{\psi\}\!\rightarrow R' \in \Delta^D$ **do**
8 Let $\sigma_1 = sym(q \text{-}\{\varphi\}\!\rightarrow q')$;
9 Let $\sigma_2 = sym(R \text{-}\{\psi\}\!\rightarrow R')$;
10 **if** $sat(\sigma_1 \wedge \sigma_2)$ **then**
11 **if** $(q', R') \notin Q$ **then**
12 $Q \leftarrow Q \cup \{(q', R')\}$;
13 $W \leftarrow W \cup \{(q', R')\}$;
14 $\Delta \leftarrow \Delta \cup \{(q,R) \text{-}\{\varphi \wedge \psi\}\!\rightarrow (q', R')\}$;
15 $C \leftarrow ground(C_1 \cup C^D)$;
16 $F \leftarrow ground(dnf(F_1 \wedge \neg F^D))$;
17 **return** $(Q, C, I, F, \Delta)$;

### 3.3 Reachability of Final States

In principle, we apply breath-first search on the product automaton $M_1 \times \overline{M_2}$ where the starting points are the initial states. If we encounter a final state, then we need to check whether its final condition is reachable. If so, we stop and know that $\mathcal{L}(M_1) \cap \overline{\mathcal{L}(M_2)} \neq \emptyset$, otherwise we continue in searching.

In detail, for every state $q$ in $M_1 \times \overline{M_2}$ the formula $\beta_q$ denotes the possible values of counters if $q$ is reached. We start from the initial states where the initial values of counters are given by the initial condition. These initial states are pushed in the list *Worklist*. For each state $q$ that is not initial, we set $\beta_q$ to $\perp$. Until *Worklist* is empty, we take a state $q$ from *Worklist*. If $q$ has self-loops, i.e., $q \text{-}\{\alpha\}\!\rightarrow q \in \Delta$ for some $\alpha$, then the possible values of counters represented by $\beta_q$ can be changed by executing the self-loops (the details are provided in Section 3.4, see function *accelerate*), thus suppose that $\beta_q$ is updated accordingly to the self-loops of $q$. Then for all states $r \neq q$ such that $q \text{-}\{\varphi\}\!\rightarrow r \in \Delta$, we test whether $\beta_q \wedge \varphi$ is satisfiable. If so, then we compute new values of counters of the states $r$. This is done by the projection $\beta_q \wedge \varphi$ of the unprimed counters used in the formula, followed by an application of the unprime function. The *unprime* function replaces every occurrence of a primed counter $c'$ by its corresponding unprimed counter $c$. Let $\psi = unprime(projection(\beta_q \wedge \varphi))$. If $[\![\psi]\!] \subseteq [\![\beta_r]\!]$, then we do not add the state $q$ to *Worklist* because we do not get any new information—if some transition from $r$ is unsatisfiable, then it will still be unsatisfiable if $\beta_r$ is modified by the new values represented by $\psi$. Otherwise $\beta_r := \beta_r \vee \psi$ and we push $r$ in *Worklist*. To test whether a state $q$ is final, we test whether $\beta_q \wedge F$ is satisfiable.

We note that some states $q$ can appear in *Worklist* more than once. But always the semantics of the formula $\beta_q$ got larger if the state appears again in the *Worklist*. Since the number of possible configurations of the product automaton, i.e., the number of possible values of counters, is finite (due to bounded counters), the state $q$ cannot be added to *Worklist* infinite number of times. Therefore, the algorithm always terminates.

### 3.4 Acceleration of Self-loops

We have built the product automaton $N = M_1 \times \overline{M_2}$ in Section 3.2. We also provide the test of reachability of states in the product automaton in Section 3.3. However, we do not say how the formula $\beta_q$ is updated if a state $q$ has a self-loop, which we fix in this section.

Let $(q,R)$ be a state of $N$. The state $(q,R)$ has a self-loop in $N$ if and only if $q$ has a self-loop in $M_1$ and $R$ has a self-loop in $\overline{M_2}$. Since $M_1$ is an MCA, we know that each state of $M_1$ has at most one self-loop by the definition. This is not true in general in the DMCA $\overline{M_2}$. It follows that the state $(q,R)$ in the product automaton can have more than one self-loop.

The trivial solution to update $\beta_q$ is that we do not distinguish between the self-loops and the outgoing transitions of the state $q$ in Section 3.3. If such a solution is used, then the algorithm for the inclusion problem for eREs runs as slowly as the methods based on NFAs. First, we give a small example showing why

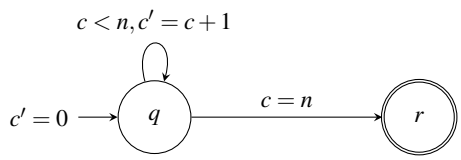



**Figure 2.** Example of an MCA $M$ with $n \in \mathbb{N}$ where $q$ is an exact counting state. The language of $M$ consists of all strings of length $n$.

**Table 1.** Possible forms of the self-loops on state $q$ in a DMCA.

| Name | Form of the label of a self-loop in DMCAs |
|---|---|
| (I) | $c_q[0] < \boldsymbol{max}_q \wedge c_q[0]' = 0$ |
| (II) | $c_q[\Psi(q)-1] \neq \boldsymbol{max}_q \wedge \bigwedge_{i=0}^{\Psi(q)-1} c_q[i]' = c_q[i]+1$ |
| (III) | $c_q[\Psi(q)-1] = \boldsymbol{max}_q \wedge c_q[0]' = 0 \wedge \bigwedge_{i=0}^{\Psi(q)-2} c_q[i+1]' = c_q[i]+1$ |

this solution is strongly inefficient, and then we present an idea of how the self-loops can be accelerated.

Suppose that we have an MCA $M_1$ as in Figure 2 and assume that $M_2$ is a one-state MCA such that $\mathcal{L}(M_2) = \Sigma^*$. Then the product automaton $M_1 \times \overline{M_2}$ has the same structure as the MCA $M_1$. We simulate the procedure from Section 3.3 with the trivial solution for updating $\beta_q$. The initial formula gives $\beta_q = (c = 0)$. Next, we check whether the self-loop or the exit transition of $q$ can be executed, in this case only the self-loop on $q$ can be executed. Note that $unprime(projection(\beta_q \wedge c < n \wedge c' = c+1))$ is equal to $c = 1$. Because $[\![c = 1]\!] \not\subseteq [\![c = 0]\!]$, we set $\beta_q = (c = 0 \vee c = 1)$. And $q$ is pushed to *Worklist*. We continue in a similar manner and after $n$ steps we obtain $\beta_q = (c = 0 \vee \cdots \vee c = n)$. The next step $n+1$ produces the formula $unprime(projection(\beta_q \wedge c < n \wedge c' = c+1))$ whose semantics is already included in $[\![\beta_q]\!]$, thus we stop executing this self-loop.

From this simple example, we see that the algorithm based on the trivial solution spends the most time in updating $\beta_q$ of states $q$ with self-loops (in practice $n$ can be large). In the example above, we can use the advantage of that the product automaton is an MCA. We use the following facts: (1) if we enter a counting state $q$, then $c_q$ is set to zero; (2) if $q$ is a counting state, then there is only one possible self-loop, called incremental, which have a fixed form. In our example, the self-loop can be accelerated by replacing the guard and update of the self-loop by $\exists k : (0 \leq k \leq n \wedge c' = c+k \wedge c' \leq n)$. Such a formula is called an *acceleration* of the self-loop. By using the same methods as above, we can obtain the same result after one iteration instead of $n$. In general, for any counting state $p$, the self-loop of $p$ can be replaced by the acceleration formula $\exists k : (0 \leq k \leq \boldsymbol{max}_q \wedge c'_q = c_q + k \wedge c'_q \leq \boldsymbol{max}_q)$.

Now, suppose that $M_1$ is a one-state such that $\mathcal{L}(M_1) = \Sigma^*$.[1] Then the product automaton $M_1 \times \overline{M_2}$ is a DMCA. As we said, the structure of DMCAs is still somewhat restricted, but more complicated than the structure of MCAs. There are two main differences: (1) each state can have several self-loops; (2) the counter guards and updates of a transition consists of several different variants of counters. Exactly as for MCAs, the key for acceleration of the self-loops in DMCAs is to find the forms of the self-loops in DMCAs. Suppose that the DMCA is the determinization of the MCA, which using only one counter. Then from the determinization algorithm [1, Section 4.2] follows that there are three different forms of self-loops (see Table 1).

We assume that $\boldsymbol{max}_q > 0$, otherwise the state is not counting. Thus the self-loops of the form (I) do not have to be accelerated, i.e., the acceleration formula looks the same as the form. The self-loops of the form (II) have a similar form as incremental self-loops in MCAs, except that we need to update more variants of counters. Since the highest variant of $c_q$ has the highest value, the acceleration formula is $\exists k : \big(0 \leq k \leq \boldsymbol{max}_q \wedge \bigwedge_{i=0}^{\Psi(q)-1} c_q[i]' = c_q[i]+k \wedge c_q[i]' \leq \boldsymbol{max}_q\big)$. The self-loops of the form (III) do not have to be accelerated if $\Psi(q)-1 = \boldsymbol{max}_q$. Otherwise we must have $\Psi(q) < \boldsymbol{max}_q$. In this case there is some largest number $k < \Psi(q)-1$ such that $c_q[k]+1 < c_q[k+1]$, or, equivalently. $c_q[\Psi(q)-1-k]+k \neq \boldsymbol{max}_q$. If there is no such $k$, then $c_q[0] > 0$. It follows that after $\Psi(q)-k$ iterations the self-loops do not add any new information. Thus the acceleration formula must need find to such a number $k$. So the acceleration formula is $\exists k : \big(0 \leq k \leq \boldsymbol{max}_q \wedge c_q[\Psi(q)-1-k]+k = \boldsymbol{max}_q \wedge \bigwedge_{i=0}^{k} c_q[i]' = i \wedge \bigwedge_{i=k+1}^{\Psi(q)-2} c_q[i]' = c_q[i-k]+k\big)$.

Suppose that we have a general product automaton. From the preceding paragraphs, we know how to accelerate a self-loop with counter variants of a single counter. But what would happen if the self-loop has counter variants of different counters? The counter guards and updates of the self-loops can be divided into $\varphi = \alpha_1 \wedge \cdots \wedge \alpha_n$ such that $\alpha_i$ contains only variants of single counter $c_i$. Each of the individual $\alpha_i$ is of the form (I)–(III), or incremental self-loop. For each of the forms we know its corresponding acceleration formula $\psi_i$, but the acceleration of $\varphi$ is not equal to the conjunction of its corresponding acceleration formulae $\varphi_i$. We need to ensure that the $k$ in each part of the counter update is the same. Replace the variable $k$ in the counter guard of $\psi$ by $k_i$. Now the acceleration formula of $\varphi$ is equal to the conjunction of $\bigwedge_{i=0}^{n} \psi_i$ and $k_1 = k_2 = \cdots = k_n$, which ensures that the counters are

[1] Our algorithm can be also used for the *universality* problem of eRE. That is, we check whether $\Sigma^* \subseteq \overline{\mathcal{L}(M_2)}$.

incremented by the same value.

The state $q$ in the product automaton can have more than one self-loop. In this case we proceed as follows. Let $\varphi_1, \ldots, \varphi_n$ be self-loops of $q$ in an arbitrary, but fixed, order. For each self-loop $\varphi_i$ we know the acceleration formula $\psi_i$. Thus we also know how $\beta_q$ is updated. If $[\![\psi_i]\!] \subseteq [\![\beta_q]\!]$, then $\beta_q$ is not updated. The function *accelerate* works as follows: we update $\beta_q$ by processing the self-loops $\varphi_1, \ldots, \varphi_n$ repeatedly until last $n$ acceleration formulae do not update $\beta_q$.

Finally, we note that it is not necessary to build the whole product automaton and after that search for a reachable final state (with a reachable final condition). In practice, we can build the product automaton on the fly and if we encounter a reachable final state (with a reachable final condition), then we can stop. In this step, we have $\mathcal{L}(M_1) \cap \overline{\mathcal{L}(M_2)} \neq \emptyset$, or equivalently $\mathcal{L}(M_1) \not\subseteq \overline{\mathcal{L}(M_2)}$. Eventually, we stop if we build the whole product automaton.

## 4. Advantage of our algorithm over the methods based on NFAs

In this section, we give a comparison between our algorithm, which is introduced in Section 3, and the methods based on NFAs. We define the *size* of the automaton (classical or counting), written $|N|$, as the number of its states.

We have given eREs $r_1$ and $r_2$. Both approaches build the product automaton $N_1 \times \overline{N_2}$ where $N_1, N_2$ are NFAs or MCAs such that $\mathcal{L}(r_1) = \mathcal{L}(N_1)$ and $\mathcal{L}(r_2) = \mathcal{L}(N_2)$. In the following, $N_i$ denotes an NFA and $M_i$ denotes an MCA for $i \in \{1, 2\}$.

The size of the product automata is $\mathcal{O}(|N_1| \cdot |\overline{N_2}|)$ and $\mathcal{O}(|M_1| \cdot |\overline{M_2}|)$, respectively. Suppose that $r_1$ is a fixed eRE except that the bounds of the counting operator are parameters. If the bounds are increased, the size of $M_1$ does not change in contrast to the size of $N_1$ where the size grows linearly. Although $\mathcal{O}(|N_1|) = \mathcal{O}(|M_1|)$, for any value of the bounds in the counting operators we have $|M_1| \leq |N_1|$. The hidden constant in $\mathcal{O}$ may be large. Suppose that in $r_1$ the counting operator occurs $k > 0$ times and the smallest value of all lower bounds of the counting operators is $\ell > 1$. Because the size of $N_1$ grows linearly we know that $|M_1| \cdot k\ell \leq |N_1|$.

From [1] we know that the complemenation of MCAs may not have be always smaller than th unfolding MCAs to NFAs and complementing NFAs. But in many cases we have $|\overline{M_2}| \leq |\overline{N_2}|$ (cf. [1]). Therefore in many cases we have $|M_1| \cdot |\overline{M_2}| \cdot k\ell \leq |N_1| \cdot |\overline{N_2}|$. Thus the advantage of our approach is that we work with smaller automata. An open question is whether the reachability test will be efficient enough so that we outperform the methods based on NFAs.

At least one example was given in the introduction. Recall, let $r_1 =$ `.*a.{k}`, $r_2 =$ `.*a.{k-1,k+1}` for $k \geq 1$. $N_1$ representing $r_1$ has $k+1$ states, $M_1$ has always two states regardless of the values of $k$ (see Figure 1). $N_2$ representing $r_2$ has $k+2$ states, the complement of $N_2$ has $2^{k+2}+1$ states, $M_2$ has always two states, the complement of $M_2$ has $(k+2)+1$ states. Note that we need to count the sink state of the complements. From the example we see that even for relatively small values of $k$ the methods based on NFAs fail.

## 5. Conclusions

In this paper, we presented the algorithm for the inclusion problem of extended regular expressions (eREs). The algorithm is based on the transformation of eREs into monadic counting automata (MCAs). The MCAs are compact representations of eREs because the number of states in MCAs does not depend on the bounds in the counting operator. Thus the product of MCAs is often significantly smaller than if eREs are transformed into nondeterministic finite automata (NFAs).

We have shown that we can solve inclusion problem for new sets of eREs if the eREs are transformed into MCAs instead of NFAs, because the solution working with NFAs fail due to exponential blow up in the number of states. Moreover, we note that the advantage of our algorithm grows if the bounds and occurrences of the counting get larger.

Although the eREs are common in practice, there are still some regular expressions that do not satisfy our abstract syntax of eREs. The possible extension is to avoid the restriction that counting operator is limited to character classes, e.g., `(abc){5,10}` denoting all strings where `abc` appears 5–10 times.

## Acknowledgements

I would like to thank my supervisor Ing. Ondřej Lengál, Ph.D. for his help.

## References

[1] Lukáš Holík, Ondřej Lengál, Olli Saarikivi, Lenka Turoňová, Margus Veanes, and Tomáš Vojnar. Succinct determinisation of counting automata via sphere construction. In *In Proc. of 17th Asian Symposium on Programming Languages and Systems - APLAS'19*, number 11893, pages 468–489. Springer Verlag, 2019.

[2] Marty Roesch. Snort: A Network Intrusion Detection and Prevention System. https://www.snort.org/.

[3] RegExLib.com: The Internet's First Regular Expression Library. https://regexlib.com/.

[4] Dag Hovland. Regular expressions with numerical constraints and automata with counters. In *Proceedings of the 6th International Colloquium on Theoretical Aspects of Computing*, ICTAC '09, page 231–245, Berlin, Heidelberg, 2009. Springer-Verlag.

[5] R. Smith, C. Estan, S. Jha, and I. Siahaan. Fast signature matching using extended finite automaton (XFA). In *Proceedings of the 4th International Conference on Information Systems Security*, ICISS '08, page 158–172, Berlin, Heidelberg, 2008. Springer-Verlag.

[6] Parosh Aziz Abdulla, Yu-Fang Chen, Lukáš Holík, Richard Mayr, and Tomáš Vojnar. When simulation meets antichains: On checking language inclusion of nondeterministic finite (tree) automata. In *Proceedings of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'10, page 158–174, Berlin, Heidelberg, 2010. Springer-Verlag.

[7] David L. Dill, Alan J. Hu, and Howard Wong-Toi. Checking for language inclusion using simulation preorders. In Kim G. Larsen and Arne Skou, editors, *Computer Aided Verification*, pages 255–265, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

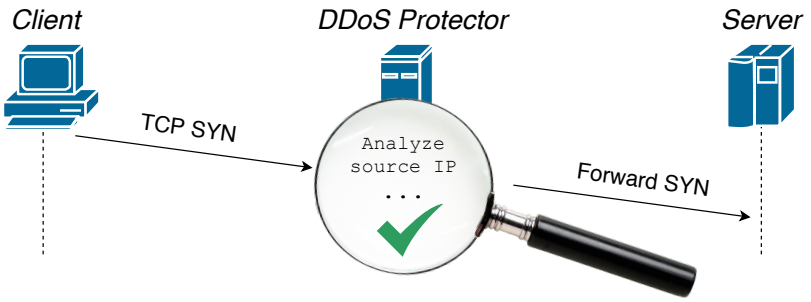
[8] Aaron R. Bradley and Zohar Manna. *The Calculus of Computation: Decision Procedures with Applications to Verification*. Springer Publishing Company, Incorporated, 1st edition, 2010.

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

# Adaptive SYN Flood Mitigation Based on Attack Vector Detection and Mitigation Process Monitoring

Patrik Goldschmidt*




**Abstract**
TCP SYN Flood is one of the most widespread DoS attack types performed on computer networks nowadays. The attack comes in many possible forms and several different mitigation methods to deflect it also exist. This paper discusses mentioned security incidents, various mitigation approaches, and presents a mechanism able to choose the most suitable method to mitigate the attack. The suggestion is made according to network traffic and the properties of mitigation methods. After the suggested method is deployed, the algorithm also monitors its behavior and may suggest a different strategy when the one currently in use proves to be ineffective. Our experiments have shown that the mechanism is able to successfully detect several attack variants and suggest a suitable method to deflect them while trying to minimize the impact on the end-user as much as possible. On the other hand, the suggestion accuracy is heavily dependent on available mitigation methods and their properties, which need to be set manually before the system can be used.

**Keywords:** TCP SYN Flood — DDoS Mitigation — Adaptive DoS Protection

**Supplementary Material:** *N/A*

*xgolds00@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

*Note that Sections 1 and 2 are partially modified versions of my Excel@FIT 2019 paper [1], which also discussed SYN Flood problematics, and thus its introduction and theoretical background are relevant for this article as well.*

## 1. Introduction

Transmission control protocol (TCP) is an integral part of the Internet protocol suite. As its importance is fundamental for the operation of the Internet, it is often misused to cause various cybersecurity threats. Data from the past several years show a strong trend towards TCP abuse to perform Distributed Denial of Service (DDoS) attacks. The report from Q4 2019 by Kaspersky Lab states that the most frequent target of a denial of service attacks was TCP, targeted by 90.5% of all the attacks [2] (Figure 1). According to Cisco, the number of DDoS attacks will double to 14.5 million p.a. by 2022 [3].

Figure 1 also shows that the majority of attacks on TCP are performed by SYN flooding, which poses a significant threat to many web services such as websites or e-mails. The purpose of this paper is to describe details of this weakness and its attack variants, present existing solutions to prevent it, and most im-

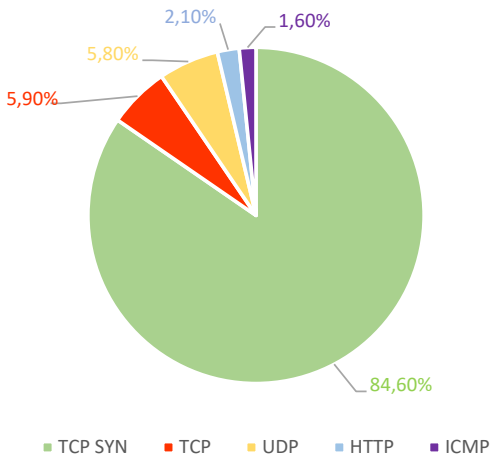



**Figure 1.** Distribution of DDoS attacks by type, Q4 2019 by Kaspersky Lab. [2]

portantly, introduce a system able to switch between these mitigation solutions in real-time according to network traffic patterns and effectivity of the currently employed mitigation strategy.

The mechanism was developed as a part of the CESNET's research project *DDoS Protector*[1], which currently contains 3 SYN Flood mitigation methods and requires a system to switch between them to enhance its mitigation capabilities. At the time of writing this paper, the mechanism is already implemented and tested, currently waiting for integration into the main system. However, results obtained so far are mostly experimental, and its fine-tuning is still in progress.

## 2. TCP Security Considerations

### 2.1 Known Vulnerabilities and Attacks

Threats that abuse TCP weaknesses can be classified as either flood or injection attack types. Flood attacks typically target a single host or a network. They aim to exhaust the target's resources by flooding bogus packets, making it inaccessible for regular clients, thus creating a denial of service. On the other hand, injection attacks are based on eavesdropping and injection of crafted segments into a TCP session. Injected data may contain malicious code, compromise the user's privacy [4], or reset the session [5].

The functionality of the most popular TCP attack – SYN flood depends on the process called TCP three-way-handshake, which is used to create a reliable channel for communication between two hosts. During this process, the server receives a request to set up a connection (SYN segment), responds with a confirming message (SYN-ACK segment), and waits until the

[1] https://www.liberouter.org/technologies/ddos-protector/

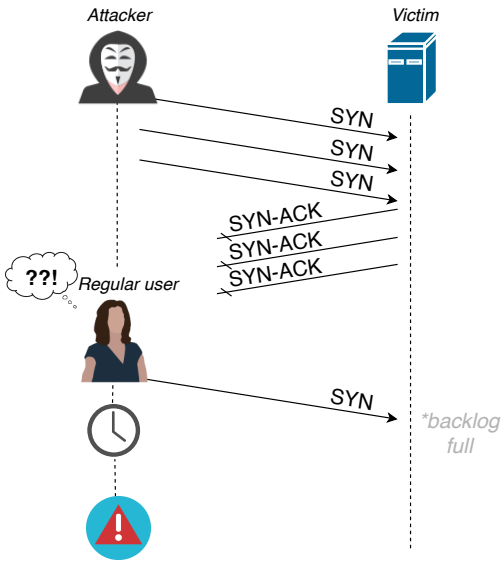



**Figure 2.** TCP SYN flood attack.

client that originated the previous SYN confirms an establishment of the connection. The rationale behind a successful DoS assumes that the server (victim) allocates a new state for every received SYN segment, and that there is a limit of such states that can be stored. These are described in RFC 793 as Transmission Control Block (TCB) data structures. TCB structures are used to store necessary state information for an individual connection. They may be implemented differently among the operating systems, but the key concept is that a new memory chunk needs to be allocated upon every new TCP connection [6].

Operating system kernels typically try to protect host memory from getting exhausted by implementing a limit of contemporary TCB structures called backlog. When the backlog limit is reached, either incoming SYN segments are ignored, or uncompleted connections in the backlog are replaced. As illustrated in Figure 2, the primary goal of SYN flooding is to exhaust the target's backlog with half-open connections so that legitimate clients will be unable to connect. For this purpose, spoofed IP addresses that do not generate a reply to SYN-ACKs are often used.

More sophisticated variants of this attack include SYN/SYN-ACK floods, Fake session (SYN + SYN-ACK + FIN flood), which flood the server with various types of TCP segments to simulate the traffic of a real client. This is done to disguise their malicious intentions since pure SYN attacks are more likely to be detected. Another special technique – Session attack utilizes a botnet to create a lot of valid TCP connections at once and stretch them as long as possible, making the victim server inaccessible [7].

Some less sophisticated TCP attacks may rely on pure strength, typically flooding SYN-ACK, frag-

mented ACK, PSH-ACK, RST or FIN segments in enormous numbers. Regular stateless firewalls typically do not analyze and filter this type of traffic, therefore, it can reach its victim more easily than pure SYN flooding. From now on, only the SYN Flood attack and its mitigation will be considered.

### 2.2 SYN Flood Mitigation Techniques

Modern operating systems provide relatively large backlogs, thus being protected from big spikes in regular TCP traffic. However, this is not sufficient to cover flood attacks described in 2.1, so specialized methods are still required. Linux kernels historically provided robust security by implementing two end-host countermeasures – *SYN cookies* and *SYN caching* [8].

SYN cache method utilizes hashing to store a lightweight fingerprint of the IP address, port number and a secret for every incoming TCP connection. This way, the operating system does not need to allocate the whole TCB, but only a fragment of the original memory required. A device implementing this method is, therefore, able to queue more requests, becoming harder to exhaust. In the BSD kernel from 2002, this optimization reduced the size of the per-connection data by 78 % while allowing up to 15359 entries [8].

In contrast to SYN cache, SYN cookies method does not need to store any state information at all, requiring no memory per connection. Essential data defining the connection alongside with a timestamp and a secret are hashed into a 32-bit value representing the *SEQ* number of the SYN-ACK segment. Upon ACK response receipt, the server can reconstruct original SYN parameters and successfully establish a connection. The method is often highly effective, but its nature denies SYN-ACK retransmission and also restricts TCP options usage [9].

Other interesting approaches, like *TCP Random Drop* [10], which aims to replace a random half-open connection with another, also exist. However, the usage of end-host mitigation techniques is sometimes undesired (e.g., to preserve system resources), unsuitable in certain environments, or insufficient against some types of attacks. Therefore, other specialized methods need to be employed as well.

Various TCP extensions and modifications with anti-DoS capabilities like TCP Cookie Transactions [11] and TCP Fast Open [12] also exist. However, none of them is globally used mainly due to a lack of support from software companies and hardware vendors.

Other potential ways for TCP DoS protection include network-based countermeasure techniques. The simplest of them is *traffic filtering* [13] and its improved variant *reverse-path forwarding* [14]. Their fundamental idea is to deny all incoming traffic from IP addresses that do not match their source network prefix (packets intentionally crafted with false IP). This process would allow discarding all traffic from spoofed IP addresses, but is not reliable, because filtering policies would need to be deployed by a majority of Internet service providers, which cannot be generally relied on (2006) [15].

SYN flood attacks were historically mitigated by firewalls, proxies, or IDS/IPS systems, which usually used initiator SYN-ACK spoofing or listener ACK spoofing techniques [15] [16]. These mitigation practices are mostly present to this day, though many security solutions also utilize cloud technologies with virtualized IDS/IPS systems instead of traditional per-network defense [17].

In real-world situations, both end-host and network-based techniques are frequently employed, and they generally do not interfere when used in combination [15]. Newer trends in DDoS mitigation also employ artificial intelligence and machine learning principles such as [18]. These approaches are generally able to protect against a wider range of attacks but suffer from a poorer performance when compared to their previously mentioned counterparts.

As may be seen, the presented methods have different behavior and consume various amounts of system resources. They also act differently from the side of the end-user. Therefore, a switching mechanism between these methods able to turn on "the best" is desired. This algorithm should choose a strategy that is able to successfully deflect the attack and affect the end-user as little as possible.

## 3. CESNET's DDoS Protector Environment

As mentioned in Section 1, this project was developed as a part of the larger-scale research – *CESNET's DDoS Protector*. This system aims to provide DDoS protection for high-speed networks, using proprietary network interface cards based on FPGA technology with a custom firmware and drivers. The Protector was firstly designed to mitigate reflective and amplification DDoS attacks but gradually evolved into a system with different capabilities, such as a TCP SYN Flood mitigation. Currently, the following SYN Flood network-based mitigation strategies are supported:

- TCP SYN Drop
- TCP Reset Cookies
- TCP ACK Spoofing

Traditionally, the mitigation methods were strictly bound to the protected network's prefix by rules, which

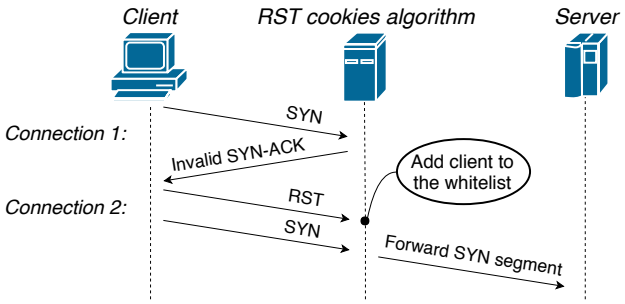



**Figure 3.** RST Cookies functionality.

had to be specified by the system's user. Therefore, the same mitigation principles were applied regardless of the properties of the attack. Also, changing the method required manual intervention as well. For this reason, a mechanism of *Adaptive SYN Flood Mitigation Modules*, as described in Section 4, was designed and developed. This section will further analyze the mentioned available mitigation methods.

### 3.1 TCP SYN Drop

TCP SYN Drop is a proprietary method developed especially for CESNET's DDoS Protector. Its functionality depends on soft and hard thresholds, which are used to limit the maximum throughput of SYN data allowed from a single client. Each client has an associated counter based on its IP address stored in the module's internal structures. If the client has not sent any ACK segment yet, the maximum number of SYN segments it can send is given by the soft threshold. If at least one ACK from the particular IP is received, SYNs are limited by the higher hard threshold. This mechanism effectively limits the number of connection requests from a single IP but can be easily fooled by injecting a random ACK segment into the session or by constantly spoofing source addresses [19].

### 3.2 TCP Reset Cookies

TCP Reset Cookies method utilizes the TCP three-way handshake mechanism to establish a security association with a client before forwarding its SYN data. As outlined in Figure 3, the association is established by intentionally crafting an invalid SYN-ACK response to the first SYN received from a client. According to the RFC 793, the client is expected to respond with the RST segment containing a specific sequence number (SEQ). The mechanism compares expected SEQ with the received one and adds IP address on the whitelist if they match. Connection requests from whitelisted clients are forwarded, whereas ones from non-whitelisted addresses are dropped. With the incorporated functionality of SYN Drop mechanism, this variant provides significantly better security, though

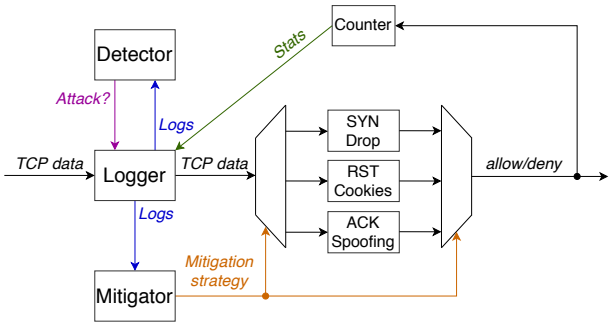



**Figure 4.** Adaptive SYN Flood Mitigation Modules overview.

at the cost of higher system resources utilization and bigger impact on the end-client [1].

### 3.3 TCP ACK Spoofing

The goal of the TCP ACK Spoofing approach is to prevent the exhaustion of the protected device's backlog by denying it to get overfilled. This process works on the principle of sending spoofed ACK segments to finish every half-open session and complete the three-way handshake. If the client does not generate an ACK segment within the specified timeout period, the ACK spoofing mechanism terminates the connection with an RST segment. If the expected ACK is received, the algorithm marks the connection as valid and does not interfere in the future TCP communication between the nodes. The method prolongs the server's ability to serve clients but does not mitigate an attack by itself. Therefore, it is not suitable against excessive volumes of SYN traffic, but can be worth a try when all other methods prove to be ineffective [19].

## 4. Adaptive SYN Flood Mitigation Modules

As outlined before, the mechanism for dynamic method switching requires to provide several features to work as desired. This functionality is provided by the following 3 separate submodules, collectively named as *Adaptive SYN Flood Mitigation Modules*:

- SYN Flood Detector
- SYN Flood Logger
- SYN Flood Mitigator

In short, these modules monitor live traffic (Logger), compare it to different thresholds and search for patterns (Detector), and choose the proper SYN flood mitigation method according to them (Mitigator) (Figure 4). Each of these submodules is discussed in more detail in the following subsections.

### 4.1 SYN Flood Logger

The main purpose of the SYN Flood Logger submodule is to gather and store statistics of the ongoing

TCP traffic for further analysis. This is achieved by directly processing IP and TCP headers of SYN, ACK and RST segments and incrementing their associated counters. To estimate the potential usage of whitelists and other data structures, the information about unique IP addresses needs to be collected as well. Since precious IP counting would be too expensive, the submodule utilizes the HyperLogLog (HLL) probabilistic data structure. Probabilistic data structures offer an ability to store vast amounts of data in relatively small structures (KiB to MiB) at the cost of precious results for approximate values. For our purposes, 512B big 9-bit HLL with a standard error of 4.6% was used.

Other statistics required to determine successful mitigation, such as the number of allowed SYNs, the number of denied SYNs, and the status of the attack (active/inactive) can not be collected directly, but the user or other submodules are responsible to provide them by appropriate function calls.

Collecting and monitoring the network statistics with respect to time is allowed by splitting the data collecting process into blocks with equal duration. Each block (time window) is represented by a single log structure, which contains all the statistics described above and adds a context about when its data have been collected. For mitigation purposes, the absolute time is not necessary, so we are working with the relative time considered from the current moment up to the $N$ time windows to the past. When a new window is created, the logs are rotated, so the oldest log is automatically replaced. The recommended length of the window is several seconds, so the mechanism stays flexible and quick to react to potential attack vector changes.

Real-time environments such as CESNET's DDoS Protector are expected to have minimum packet delays and be able to handle tens of gigabits of traffic per second. For this reason, the traffic logging process needs to be parallelized into several threads, each running its own Logger instance. Results from all instances are then merged into a master-logger, which is used for querying and further data analysis. Since the process of merging logs takes some time, each thread needs to have 2 independent logger instances and support a mechanism to write only to Loggers currently not being merged. This way, the statistics can be collected and evaluated in mutex-free manner without data races.

### 4.2 SYN Flood Detector

SYN Flood Detector is a stateless submodule used to find out whether the SYN Flood attack is present or not. The current implementation of the Detector consists of the set of triggers and thresholds, which are fired when certain patterns and conditions are found in the statistics structure collected by the Logger submodule. Currently, active detection triggers include simple SYN messages count in the current time window, but also more advanced patterns, such as ratios between unique SYN addresses and unique pure ACK addresses and also historical thresholds, such as if the SYN ratio was consistently high in the past several time windows but the hard limit in the current time window was not reached, the possible attack may get reported anyway. As displayed in Figure 4, the results from the Detector are used to update window statistics in the Logger submodule. These data are then passed to the Mitigator instance for further analysis.

### 4.3 SYN Flood Mitigator

SYN Flood Mitigator submodule is responsible for mitigation methods management and decision-making which one is the most suitable to deflect the ongoing attack. Its principle is based on maintaining a list of available mitigation methods, choosing one of them according to their properties, and then determining whether its employment is effective against the attack or not. If the mitigation proves to be ineffective, the method is marked not to be used again during this particular attack, and another available strategy is tried.

The list of available methods is created by registration. This process involves specification of a mitigation method strategy structure, which is supposed to describe properties of the mitigation method, such as how much computer memory does it need, how many hashing functions it computes during the segment processing, or which types of TCP traffic does it process. There are currently 15 of these method parameters, aiming to provide a very detailed description of methods, but be as general as possible, so new methods may be easily added in the future. In other words, the whole mechanism is not limited only to 3 mitigation methods supported by the DDoS Protector.

Internally, the registered mitigation methods are stored in an array ordered based on their rating. The rating describes an impact the mitigation method has on the end-user. Processing frequently used TCP segments, such as ACKs, hashing, dropping SYNs, requiring retransmission, etc. negatively impacts the end-user, and hence increases the rating value. Mitigation methods with the lowest ratings are placed at the beginning of this array, so the method determination process considers them first. More heavyweight methods reside at the end, so they are not used until more serious conditions occur.

The optimal method is determined according to discovered traffic patterns – such as if the method can be effective and its whitelists will not get overfilled

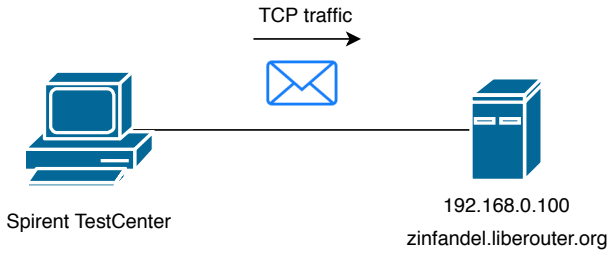



**Figure 5.** Experimental configuration testing environment scheme.

by many unique IP addresses. If more than one suitable method is found, the mechanism always chooses the one having a lower rating, thus having a lesser impact on the end-user. After the method is chosen, it is not changed for several time windows to gather enough mitigation statistics and prevent frequent method switching. If the method is determined to be ineffective (e.g., not dropping any SYN segments while the attack is ongoing), it is marked as unsuitable and not taken into account during the decision-making process for the several following time windows.

## 5. Evaluation and Results

According to the modular and highly configurable design, all of the submodules described in Section 4 can be customized and fine-tuned during both the compilation and execution stages using various macros and configuration structures. For example, the user may specify the sensitivity of the Detector thresholds, defining that an attack is triggered sooner or later. Weights for methods ratings can also be altered, so the user is in full control over which mitigation method will get prioritized when searching for the best match. As the reader may have noticed, the functionality and mitigation capabilities of the mechanism are particularly dependent on the configuration it is initialized with.

At first, an initial experimental configuration was created for mechanism evaluation purposes. This setup then was iteratively developed based on its performance in the performed tests. Our test environment consisted of a simple point-to-point network (Figure 5), established in CESNET's Liberouter project private network. The program using *Adaptive SYN Flood Mitigation Modules* was deployed on a server connected to a machine running *Spirent TestCenter* application[2]. This program provides a feature to forge custom packets at excessively high speeds, thus being an ideal choice to simulate a SYN Flood DDoS attack.

The current implementation of the SYN Flood detection in the CESNET's DDoS Protector is based only on a single threshold. Our tests have shown that adaptive approach described in this paper is much more sensitive and can react on wider variety of possible attack vectors. Apart from regular threshold detection, our mechanism successfully reports events like a sudden increase of the SYN traffic. For example, if the volume of SYN traffic increases by 20% over the past 30 time windows (e.g., 30 seconds), and at least 80% of the SYN threshold is reached, a possible attack is reported. This may, of course, produce a number of false positives such as in the morning when employees are arriving to work. Again, all these facts showcase the importance of the configuration.

| Rank | Method | Rating |
|---|---|---|
| 1. | TCP SYN Drop | 127.014 |
| 2. | TCP RST Cookies | 149.100 |
| 3. | TCP ACK Spoofing | 491.927 |

**Table 1.** Mitigation methods Mitigator ratings

There are many more detection patterns, such as the ratio between unique IP addresses sending (pure) ACK segments and unique addresses sending SYNs, which may also signal a possible attack. Currently, a value we use for this threshold is 1.20, but again, it is highly dependable on the environment.

After initially filling the properties of the mitigation methods from Section 3, the ratings shown in Table 1 were obtained. This corresponds to our expectations since the processing of ACK segments is rather costly. This property results from the high share of ACK segments in regular TCP traffic (more than 90% according to our measurement between CESNET and ACONET networks in 2018). After running the algorithm with this setup upon regular pure SYN Flood attack, we observed a behavior which followed its theoretical functionality described in Section 4. The method with the lowest mitigation rating (SYN Drop) was chosen, and the mechanism did not change it until the pure SYN attack was in progress. As we begin mixing SYNs with several fake ACKs, the SYN Drop mitigation started to be ineffective, so the mechanism switched to RST Cookies and again, kept it active until we managed to fool it with regular TCP stack as well. In this case, the last resort method – TCP ACK Spoofing was employed. When we completely turned off the SYN generation, ACK Spoofing was held for several more time windows before stopping the mitigation completely. Several similar tests like this were also conducted with expected results corresponding to the module configuration.

As a result of these findings, we deduce that the mechanism is a step ahead from the current DDoS Pro-

[2] spirent.com/products/testcenter

tector's state and may work fine if configured properly. However, the trend of setting everything manually is unscalable and unsuitable in general. For this reason, we are trying to find a "good" universal configuration, that will work fine until the setup process could be done in a more automatized manner. The author of this document is already working on TCP segment classification using machine learning techniques, which may complement or replace the Detector's functionality in the future.

## 6. Conclusions

This paper has focused on the analysis of the TCP SYN flood attack and discussed several techniques for its mitigation. All of the examined methods have certain advantages and disadvantages, making them suitable for different variants of the attack. To tackle this issue, the paper has presented the *Adaptive SYN Flood Mitigation* mechanism, which aims to provide a way to determine the most viable mitigation method based on detected traffic patterns and mitigation statistics.

Our experimental results have proved that the mechanism can detect regular and more sophisticated SYN floods that the current DDoS Protector triggers were not able to react on. However, its mitigation capabilities are heavily dependent on the configuration, which is also partially dependent on the protected network properties and the environment the algorithm is deployed in. The "universal" configuration able to facilitate most of the environments has not been found yet, so we are continuing to tweak and experiment with the mechanism to achieve better performance and mitigation capabilities.

Although the described approach is officially finished, further improvements can still be made. Apart from finding its optimal and universal configurations, there is a huge potential in improving the detection capabilities and the mitigation method suggestion principles. The Logger submodule may also be slightly modified and reused for feature extraction for the DDoS mitigation machine learning project, which the author of this article is currently starting to work on.

## Acknowledgements

I would like to thank my supervisor *Ing. Jan Kučera* for his friendly nature and support. Many thanks also to *Ing. Martin Žádník Ph.D.*, who introduced me to the CESNET's security research projects back in 2017.

*This project was developed as a part of the research "Adaptive protection against DDoS attacks*[3]*" granted by Ministry of Interior, Czech Republic under the auspices of CESNET a.l.e. and its contractual research for students, in cooperation with Faculty of Information Technology at Brno University of Technology.*

[3] starfos.tacr.cz/en/project/VI20192022137

## References

[1] P. Goldschmidt. TCP Reset Cookies – a heuristic method for TCP SYN Flood mitigation. In *Excel@FIT 2019*, Apr 2019.

[2] Kupreev, Badovskaya, and Gutnikov. Ddos attacks in q4 2019. Technical report, Kaspersky Lab, February 2020. Available at: https://securelist.com/ddos-report-q4-2019/96154/.

[3] Cisco Systems. Cisco Visual Networking Index: Forecast and Trends, 2017–2022 White Paper. Technical report, Jan 2017.

[4] B. Harris and R. Hunt. TCP/IP security threats and attack methods. *Computer Communications*, 22(10), 1999.

[5] Paul A. Watson. Slipping in the Window: TCP Reset Attacks. Jan 2004.

[6] W. Eddy. TCP SYN Flooding Attacks and Common Mitigations. RFC 4987 (Informational), August 2007.

[7] Riorey, Inc. Taxonomy of DDoS Attacks. Accessed: 2019-08-26.

[8] Jonathan Lemon. Resisting SYN Flood DoS Attacks with a SYN Cache. In *USENIX Conference on the BSD operating system*, 2002.

[9] Daniel J. Bernstein and Eric Schenk. SYN Cookies proposal, Sept 1996. Accessed: 2019-08-26.

[10] L. Ricciulli, P. Lincoln, et al. TCP SYN Flooding Defense, 1999.

[11] W. Simpson. TCP Cookie Transactions. RFC 6013, Jan 2011.

[12] Y. Cheng, J. Chu, S. Radhakrishnan, and A. Jain. TCP Fast Open. RFC 7413, Dec 2014.

[13] P. Fergusson and D. Senie. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2827, May 2000.

[14] Baker, F. and Savola, P. Ingress Filtering for Multihomed Networks. RFC 3704, Mar 2004.

[15] Wesley M. Eddy. Defenses Against TCP SYN Flooding Attacks. *The Internet Protocol Journal*, 9(4), Dec 2006.

[16] Christoph L. Schuba, Ivan V. Krsul, Markus G. Kuhn, Eugene H. Spafford, Aurobindo Sundaram, and Diego Zamboni. Analysis of a Denial of Service Attack on TCP. In *Proceedings of the IEEE Symposium on Security and Privacy*, 1997.

[17] Opeyemi Osanaiye et al. Distributed denial of service (DDoS) resilience in cloud: Review and conceptual cloud DDoS mitigation framework. *Journal of Network and Computer Applications*, 67, 2016.

[18] Chuanhuang Li, Yan Wu, Xiaoyong Yuan, Zhengjun Sun, Weiming Wang, et al. Detection and defense of DDoS attack-based on deep learning in OpenFlow-based SDN. *International Journal of Communication Systems*, 31(5), 2018.

[19] P. Goldschmidt. Heuristic Methods for the Mitigation of DDoS Attacks That Abuse TCP Protocol, May 2019. Bachelor thesis. Faculty of Information Technology, Brno University of Technology.

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

# Automated human recognition from image data

Lukáš Dobiš

**Abstract**
This paper describes an approach for automated human recognition by using convolutional neural networks (CNN) to perform facial analysis of persons face in image data. The predicted biometric indicators are following: age, gender, facial landmarks and facial expression. CNN architectures with pretrained weights for each task are described. Age estimation CNN has new weights trained and freezed, then has added new LSTM layers into its architecture. New LSTM layers are trained and tested on newly created video data set. Solution for human recognition inference with single image and time series variants, in form of script with interconnected CNNs is explained and its inference speed performance supports further proposed expansion plans for live video inference.

**Keywords:** Human classification — Computer vision — Deep learning

**Supplementary Material:** GitHub repository

*xdobis01@stud.feec.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

## 1. Introduction

Automated human recognition as a technology is used widely in human-computer interaction, the security and surveillance industry, demographic research, commercial development, mobile application, service robotics and video games [44]. Use of computer vision for human recognition, has greatly increased with introduction of deep learning methods based on CNN.

State-of-art neural networks often focus only on one task, so combining multiple of them using their strengths, leads to composition of networks each responsible for different task. This way used networks can have their architecture or weights changed at will. So when better architecture for one task is discovered it can be used as replacement without need to retrain whole composition of networks. With this approach it is also easier to train, test and evaluate networks, since independent tasks have often their own benchmark data sets.

This paper applies multiple pretrained convolutional neural networks to detect face, its landmarks and to further predict person age, gender and facial expression. These are important descriptors in automated human recognition. Since each human has unique appearance and leads different lifestyle, these descriptors precise prediction creates a challenge for current algorithms. Aim of this work is to create composition of CNNs oriented on human recognition, which will be capable of real-time inference in video, with improvement in prediction by utilizing temporal information.

Age estimation being hardest task is main focus. So in addition to pretrained weights has used age prediction network new weights trained and freezed. Then is modified with additional LSTM layers which are independently trained and tested on new data set created for this purpose. Results show that adding LSTM layers has improved age prediction performance.

Human recognition is therefore solved with distinct CNNs interconnected to create automated human face detection and subsequent age, gender and emotion classification from images. Overall lightweight single image inference time on GPU is 0.1038 second. Modified solution for video inference, needs 20 images for age prediction and has inference time 0.1181s.

By using several lightweight precise CNNs, the proposed solution is fast, accurate and highly modular for further improvements.

## 2. Background and related works

### 2.1 Object detection

Most specialized detectors like face detectors are based on general purpose object detection architectures [38],

which use feature extraction networks pretrained on large data sets like ImageNet [12], PASCAL2007 [14] or MS COCO [23].

As an object detection consist of two steps. First step is finding parameters of so called bounding box which fits over detected object and second detected object classification. To well known approaches belong: Faster R-CNN[31], Deconvolutional Single Shot Detector (DSSD)[16], You Only Look Once detector v.3 (YOLOv3) [30], RetinaNet[24] and CenterNet[41]. Approaches are described in order of their first appearance.

Faster R-CNN [31] is fourth iteration on region proposing CNN approach. Its one of two stage methods, which use two separate parts of network, one to propose regions for object location and second to classify object. This approach is the most accurate but large number of region proposals affects inference speed. Other mentioned approaches are one-stage methods so localizing and classifying is done at once by same parts of network. This greatly improves inference speed but at cost of accuracy mainly for smaller objects.

Deconvolutional SSD [16] approach draws feature maps from feature extractor at differing depth. Bounding boxes have spatial geometric parameters regressed and class probabilities estimated at each drawn depth level.

YOLOv3 [30] divides image into grid of cells, where for each cell all possible bounding boxes are predicted. All this takes place in form of fully convolutional network, so YOLO is extremely fast and invariant to the input image size.

Relatively new is RetinaNet [24] approach, its innovation lies in using different type of loss function for classification rather than classic cross entropy. Hard to classify samples of rare class are often in low number and are overwhelmed by easy samples contribution to loss, which then dominates the gradient. RetinaNet loss function called Focal loss solves this by increasing importance of correcting misclassified samples. In terms of speed and accuracy RetinaNet stands in the middle, Figure 1.

New modern approach CenterNet [41] treats objects as a single point, center point of its bounding box. It uses keypoint estimation to find object center points and regresses other object properties like size, orientation and pose. CenterNet architecture claims to be simpler, faster and more accurate then other bounding box detectors. But currently to authors knowledge CenterNet based facial detectors outperformed RetinaNet based architectures only in terms of speed.

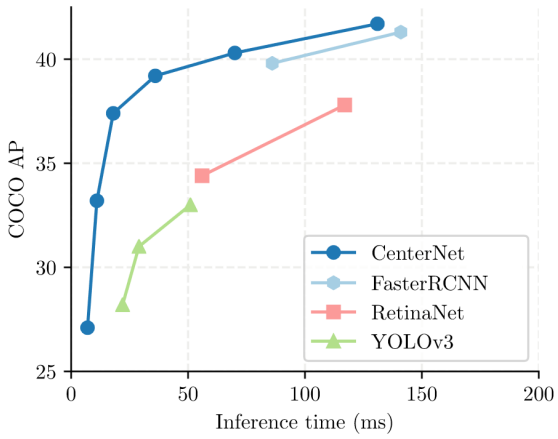



**Figure 1.** Speed-accuracy trade-off on COCO validation for some of the state-of-the-art real-time detectors [41].

Since this work should be capable of real-time inference, the most accurate approach Faster R-CNN was not chosen. Therefore facial detector used in this work is based on second most accurate approach RetinaNet. This facial detector is called RetinaFace[13] and is examined in subsection (3.1).

## 2.2 Age estimation

Current approaches treat age as regression metric or as ordinal property to be classified. Ordinal regression (OR) CNNs have been proven to outperform metric regression CNN [25].

In the field of machine learning OR has given a way to extend classification algorithms to solve regression tasks like age estimation, by reformulating problem to utilize multiple binary classification tasks. Where binary tasks represent range of values which output can be classified. First attempts used perceptrons and support vector machines [11][10]. Then came unifying general reduction framework from OR to binary classification [22]. Later this framework has been used for CNN estimating age and had proven to be at state-of-art level [25]. It portrayed OR problem with $K$ ranks as $K-1$ binary classification problems. Where $k$ ($k \in (1,2,..,K-1)$) task predicted whether age label of a face on image is greater than tasks rank $r_k$. All tasks share intermediate layers but have distinct weights in output layer. While successful at the time this approach had flaw in its classifier inconsistency. There was no assurance that tasks do not contradict each other. For example one task predicts age not above 20 and higher rank task predicts age above 30.

Modification of [25] called Ranking-CNN trains a series of CNNs and aggregates their output to predict age label of a given face image. This improves the predictive performance in comparison with a single CNN

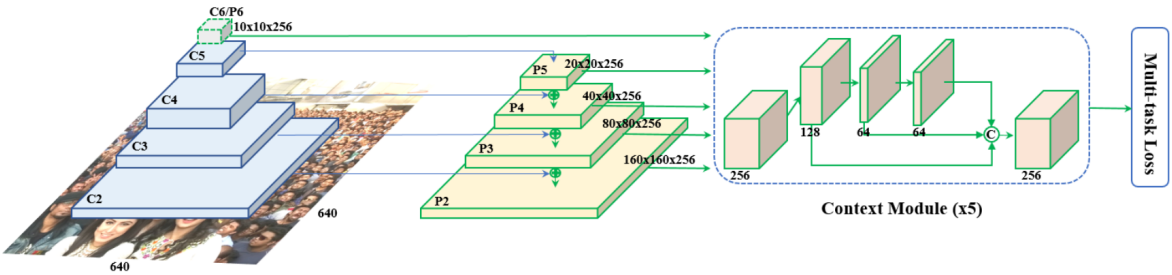



**Figure 2.** The RetinaFace neural network architecture [24].

with multiple binary outputs. But there is disadvantage in form of considerable increase in training complexity and still does not address classifier inconsistency [8].

Another approach utilizing binary classifiers for OR is siamese CNN architecture. With only single output neuron, comparisons are made between input image and multiple meticulously selected anchor images. From this comparisons an output rank label is computed, this is one of the solutions to class inconsistency [28].

Example of non OR approach is All-in-one CNN, training for more general face attributes analysis tasks (face detection,gender prediction, age estimation, etc.) improves the overall performance of metric regression, by sharing lower-layer parameters [29].

Different non OR approach CNN with cascades was designed to classify face images into age groups followed by metric regression modules for more accurate age estimation [7].

Last mentioned method is OR Consistent Rank Logits (CORAL) framework. This methods solves class inconsistency problem and addresses training data set uneven class frequency. Method will be thoroughly examined in subsection (3.2), as it is used in this paper [5].

### 2.3 Gender classification

There are various method to classify gender, this survey will mention methods using RGB images only. First methods used Support vector machine classifiers on image intensities [37]. Same concept using image intensities, used Adaboost instead [2], or applied Weber's Local texture descriptor [36]. Other methods used image intensity together with shape and texture features in combination with mutual information [27]. Another combination was Local binary pattern (LBP) and an Adaboost classifier [34]. First well established method using CNN was [20], others followed [3], [4]. Last mentioned is used by this paper, and described in subsection (3.4).

### 2.4 Emotion classification

Overview of emotion recognition methods can be found in [19]. Some of the often cited works are [21] and [4]. Latter is used in this paper. Temporal information has also been proved to help better classify emotions [15].

## 3. Used architectures

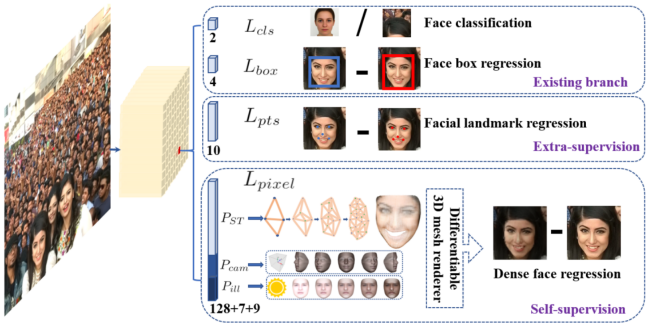
There are 4 CNNs used by this paper, first CNN is the most crucial as its output is input to all other CNNs. Its role is to detect persons face. Other 3 CNNs use this face as input to predict their own predictions.

### 3.1 RetinaFace network

RetinaFace is face detector based on RetinaNet architecture [13]. It broadens goals of face detection from just traditional bounding box parameters prediction to include also prediction of 5 facial landmarks. As loss function it uses multitask focal loss function (1),which takes into account face alignment, pixel-wise face parsing and 3D dense correspondence regression , Figure 3.

$$L = L_{cls}(p_i, p_i^*) + \lambda_1 p_i^* L_{box}(t_i, t_i^*) + \lambda_2 p_i^* L_{pts}(l_i, l_i^*) + \lambda_3 p_i^* L_{pixel} \quad (1)$$

**Figure 3.** RetinaFace one-stage pixel-wise face localization employs extra-supervised and self-supervised multi-task learning in parallel with the box classification and regression branches [13]




First element $L_{cls}(p_i, p_i^*)$ is face classification binary softmax loss, with $p_i$ being probability of anchor

box having a face and $p_i^*$ equal to 1 for positive anchor and 0 for negative anchor. Second element $L_{box}(t_i, t_i^*)$ is box regression loss equal to $\mathrm{R}(t_i - t_i^*)$ where R is $smooth - L_1$ from [31], parameters $t_i = [t_x, t_y, t_w, t_h]$, $t_i^* = [t_x^*, t_y^*, t_w^*, t_h^*]$ are predicted and ground truth bounding box geometric transformation parameters associated with positive anchor. Third element is $L_{pts}(l_i, l_i^*)$ is facial landmark regression loss with $l_x$ and $l_y$ coordinates parameters for 5 landmarks and each predicted $l_i$ and ground truth $l_i^*$ landmarks. Their transformation and loss is computed same as with box regression loss. Fourth element $L_{pixel}$ is Dense Regression Loss and has equation (2). Dense Regression Loss is pixel-wise difference of rendered 2D face $\mathscr{R}(\mathscr{D}_{P_{ST}}, P_{cam}, P_{ill})$ and ground truth face. $W$ and $H$ are width and height of anchor crop $I_{i,j}^*$. $\mathscr{R}$ is 3D mesh renderer used with shape and texture parameters $P_{ST}$ to project a colouredmesh $\mathscr{D}_{P_{ST}}$ onto 2D image plane with camera parameters $P_{cam}$ and illumination parameters $P_{ill}$.

$$L_{pixel} = \frac{1}{W * H} \sum_{i}^{W} \sum_{j}^{H} ||\mathscr{R}(\mathscr{D}_{P_{ST}}, P_{cam}, P_{ill})_{i,j} - I_{i,j}^*||_1 \tag{2}$$

Last loss-balancing parameters $\lambda_1$,$\lambda_2$,$\lambda_3$ are weights with values 0.25,0.1 and 0.01 this means that bounding box and landmark loss is deemed more important than dense correspondence regression loss, Figure 3.

Mathematical apparatus to predict $P_{ST}$ uses mesh decoder with graph convolution method based on fast localised spectral filtering formulated as Chebyshev polynomial truncated at order K, with each order evaluated at scaled Laplacian $\hat{L}$. This filtering operation uses sparse matrices and is extremely fast but its thorough explanation is above extent of this paper so for full mesh decoder explanation one should read RetinaFace paper [13].

RetinaFace architecture uses feature pyramid network with its outputs flowing through context modules to shared loss head, Figure 2.

Independent context module is applied to each of the feature pyramid levels to increase the receptive field and enhance the rigid context modeling power. In lateral connections the deformable convolution network strengthens the non-rigid context modeling capacity.

With confidence threshold of 0.05 are most of the bounding box predictions filtered out. Non maximum suppression is applied with intersection over union threshold equal to 0.4 on top 400 confidence score bounding boxes. Each overlapping bounding box contribution to final prediction location is weighted by his confidence score.

### 3.2 CORAL model

Consistent Rank Logits model is CNN with ordinal responses. Its authors provide theoretical guarantees of class consistency, well defined generalization bounds, task-specific importance weighting, while accomplishing lower number of parameters to be trained in comparison to other state-of-art CNNs. These guarantees are for full review in original paper [5].

Age rank label $y_i$ is extended into $K-1$ binary labels $y_i^{(1)}, \ldots, y_i^{K-1}$, where $y_i^{(k)} \in \{0,1\}$ indicates if $y_i$ exceeds rank $r_k$ (3). This indication is for latter purposes, defined as indicator function $\mathbb{1}\{\cdot\}$ which if inner condition is true returns 1, otherwise returns 0.

$$y_i^{(k)} = \begin{cases} 1 & y_i > r_k \\ 0 & y_i \leq r_k \end{cases} \tag{3}$$

So instead of single ordinal age label, has CORAL anotation form of binary vector where ones start at index of original age label ordinal value 4. This allows for training simple CNN with $K-1$ binary classifiers in output layer. They share weight parameter but have independent bias which effectively deals with class inconsistency and lowers number of parameters.

$$h(\mathbf{x_i}) = r_q \quad (4) \qquad q = 1 + \sum_{k=1}^{K-1} f_k(\mathbf{x_i}) \quad (5)$$

From binary classifier tasks responses a predicted rank $r_q$ is obtained in equation (4). And binary label vector index $q$ is equal to one plus sum of predictions $f_k(x_i) \in \{0,1\}$ where $k$ is index of binary classifier in output layer (5). All predictions $f_k$ must reflect ordinal information and at same time be rank-monotonic. Rank-monotonic rule for ordinal values can be simply explained for age case as moving one ordinal value to higher index must increase its actual non ordinal metric value and same must hold for moving in opposite direction (6) [5].

$$f_1(x_i) \geq f_2(x_i) \geq \ldots \geq f_{K-2}(x_i) \geq f_{K-1}(x_i) \tag{6}$$

Let $\mathbf{W}$ denote weight parameters excluding the bias units of output layer. Let penultimate layer, whose output is denoted as $g(\mathbf{x}_i, \mathbf{W})$ , shares a single weight with all neurons of output layer. To $g(x_i, \mathbf{W})$ are then added $K-1$ independent bias units. Together $\{g(\mathbf{x}_i, \mathbf{W}) + b_k\}_{k=1}^{K-1}$ create inputs to the corresponding binary classifiers in the output layer. Then predicted empirical probability for binary classifier task $k$ is defined in equation (7). Where $s$ is logistic sigmoid function (8).

$$\hat{P}(y_i^{(k)} = 1) = s(g(\mathbf{x}_i, \mathbf{W}) + b_k) \tag{7}$$

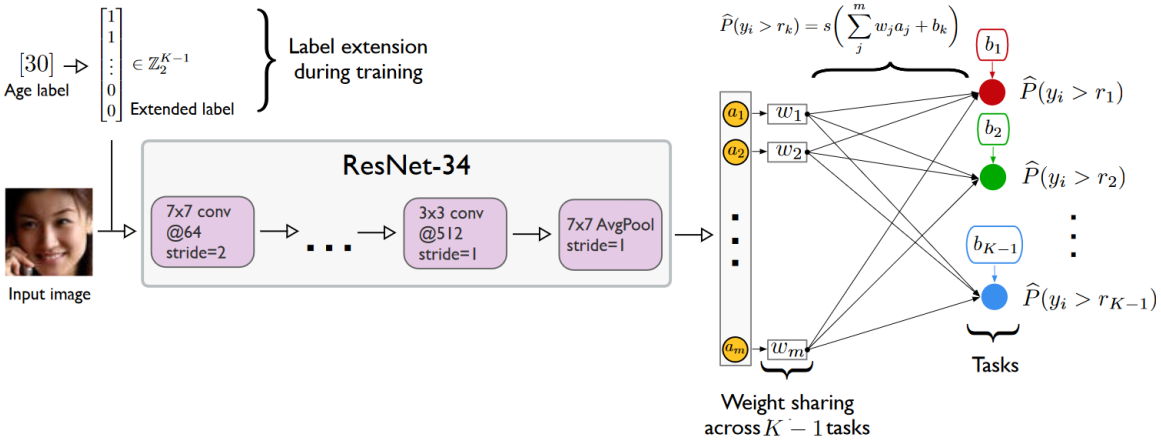



**Figure 4.** CORAL model architecture [5]

$$s(z) = \frac{1}{(1+exp(-z))} \tag{8}$$

In training CORAL minimized loss function is weighted CE of $K-1$ binary classifiers (9). Where $\lambda^{(k)}$ denotes the weight of loss associated with $k$-th classifier. For rank prediction, binary labels $f_k(\mathbf{x}_i)$ are obtained from equation (10) [5].

$$L(\mathbf{W},\mathbf{b}) = -\sum_{i=1}^{N}\sum_{k=1}^{K-1}\lambda^k[\log(s(g(\mathbf{x}_i,\mathbf{W})+b_k))y_i^{(k)} + \log(1-s(g(\mathbf{x}_i,\mathbf{W})+b_k))(1-y_i^{(k)})] \tag{9}$$

$$f_k(\mathbf{x}_i) = \mathbb{1}\{\hat{P}(y_i^{(k)}=1) > 0.5\} \tag{10}$$

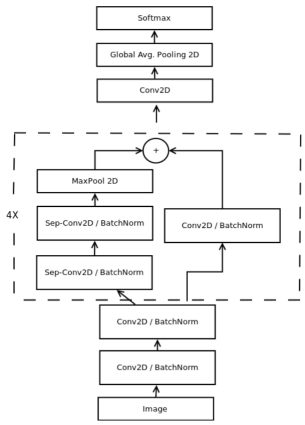
### 3.3 Proposed modification of CORAL model

Long-short time memory cell (LSTM) is a recurring neural network architecture used for time sequences of data [17]. It has very high rate of success and is one of main drivers of current AI research growth [45]. By adding LSTM layers to current CORAL model, network should be able to learn to extract temporal information to improve its performance. In CORAL architecture in Figure 4, are features after propagating through ResNet-34, concatenated into 2048 feature long 1D tensor. This tensor then acts as input to last linear layer which predicts results of binary classification tasks. By adding two LSTM layers to first process a time series of concatenated 1D tensors of multiple images, final LSTM output 2048 feature long tensor should be enriched with temporal information to provide provide better prediction in last linear layer.

### 3.4 Gender and emotion classification

Gender classification network [4] classifies into woman and man labels. For this it utilizes fully-convolutional neural network for binary classification architecture. Emotion is classified similarly, but using same architecture for multi-class classification. It can classify emotion into angry, disgust, fear, happy, sad, surprise and neutral labels [4]. Both networks take greater crop then bounding box face because networks were trained to utilize semantic information about hair to deliver more accurate prediction, mainly for gender. Architecture components are in Figure 5.

**Figure 5.** Architecture mini-Xception, used in gender,emotion classification networks [4]




## 4. Implementation

Implementation of CNNs has been written in programming language Python. All networks except one modified CORAL variant, had their weights and necessary code for network initialization downloaded from github repositories of CNNs authors. RetinaFace and CORAL networks were implemented using PyTorch framework[26], while gender and emotion networks were implemented with Keras API[9] using TensorFlow framework[1].

### 4.1 Script initialization

Main script is Recognition-net.py that takes as input directory of images to be analyzed and directory to save analyzed images with prediction metadata. Other optional inputs are RetinaFace base architecture

and weights (default ResNet-50 or MobileNet-0.25), CORAL weights training data set (AFAD [25],CACD[6], default MORPH[32],UTK[40], new UTK (1-100)) and processing unit (default CPU,GPU).

Script contains functions and classes from authors scripts which are used to create network model and load weights. Script starts with reading user inputs and based on user input initializes each CNN.

### 4.2 Face detection

Image is passed to RetinaFace network which returns bounding boxes in form of top left and bottom right coordinates and their probability scores. Third output are parametrized landmarks that are decoded into image coordinates from image size. Boxes with low probability are rejected and overlapping boxes are united using non-maximum suppression.

### 4.3 Facial analysis

Information about top left and bottom right corner is used to crop raw image. Cropped image of face is resized into 128x128 and centered. In this form is image passed to CORAL model which returns vector of age probabilities, this vector is thresholded by 0.5 (10) and then is summed into estimated age label. Bottom age limit of picked data set of CORAL weights is added to estimated age label. This is the final age prediction.

Next bounding box location is used again to crop out face, but this time is used area larger than detected bounding box. It takes 50% wider and 66% longer face area than original bounding box. This cropped face image is converted to gray scale. Gray and RGB image variants are then normalized and RGB variant is resized into 48x48 size and gray variant into 64x64 size. RGB face image goes into gender classification network and gray face image is input to emotion classification network. Both networks output a vector of label probabilities where prediction is label with maximum probability.

All recognized faces with their predictions are drawn onto raw image and saved to inputted save directory. Along with analyzed image the following data is stored in .txt file with same name: recognized faces bounding boxes metadata with predicted age, gender and emotion.

### 4.4 CORAL LSTM modification

Videos of subjects with age range 0-100 were downloaded from Youtube channel SoulPancake [46]. Each video had around around 50 or more unique individuals with their age displayed in bottom left corner. Each person had their scene extracted and first 20 images of scene were used to create one sequence per person. Training data set has 68 sequences, and testing data set has 35 sequences. Majority of individuals are not present in both data sets. In few cases where individual is present in both data sets, there is age difference of around 3-4 years.

Age estimation CORAL network had new weights trained on UTK data set[40]. Using same setting as authors with 200 epochs, Adam optimizer and learning rate of 0.0005 [5]. These weights were freezed, then two LSTM layers were inserted between last two layers of architecture, Figure 4. These layers were trained and tested on new video data set. Training setting was same as with original weights training.

In Recognition-LSTM net.py is all essential functionality same as in Recognition-net.py, but directory to be analyzed is expected to have time sequence of images with filename containing their index in time. Option to choose CORAL weights is not available since weights for LSTM modified CORAL architecture have only one variant. Output is same but first 19 images lack age prediction, because for age inference there is requirement of 20 images.

## 5. Results

As CPU was used AMD Ryzen 5 2600 Six-Core 3.4 GHz Processor and for GPU NVIDIA GeForce GTX 1060 6GB. Inference speed of each CNN is in Table 1. Author of this paper used mostly PyTorch and had not installed driver requirements needed for GPU use for Keras and TensorFlow frameworks, therefore only CPU was used for gender and emotion inference. Average time of forward pass through LSTM layers was 1.642 ms (GPU).

Face detection AP (Area under curve AUC) was tested using both trained available RetinaFace architectures and weights in Table 2. Age prediction accuracy of CORAL network for each available weights

**Table 1.** CNN average inference speed, tested on FDDB data set [18] (1-4 faces). Inference speed for one image and LSTM modified version, used lightweight RetinaFace MobileNet-0.25 architecture

| CNN network | CPU speed | GPU speed |
|---|---|---|
| RetinaFace Resnet-50 | 1.8668s | 0.1539s |
| RetinaFace MobileNet-0.25 | 0.1399s | 0.0243s |
| CORAL | 0.1407s | 0.0157s |
| CORAL with LSTM | 1.1025s | 0.0202s |
| Gender model | 0.0048s | - |
| Emotion model | 0.0032s | - |
| One image inference | 0.4816s | 0.1038s |
| Modified one image inference | 1.3810s | 0.1181s |

**Figure 6.** Script output showing different age predictions for CORAL weights trained on different data sets (AFAD - top left, CACD - top right, MORPH - bottom left, UTK - bottom right)

was tested on UTK data set, Table 3. All weights were trained with different age ranges AFAD(15-40), CACD(14-62), MORPH(16-70), UTK(21-60), so to evaluate their performance, only images with persons in their age range intersection (21-40) were used for evaluation. Last compared weights were trained by author of this paper, using whole age (1-100) range of UTK data set. Metrics used to evaluate performance of age estimation were mean absolute error and root mean square error.

Table 4 shows comparison between CORAL model trained by author of paper on UTK data set and same model trained with additional LSTM layers. Since LSTM needs images in time sequence, new video data set was used for evaluation.

## 6. Conclusions

This paper describes basic scalable groundwork for further additional facial analysis.

Each network performance in terms of accuracy was satisfactory: RetinaFace predicts reliably as can be seen on face detection data set WIDER FACE on Table 2, gender network has 85.33% accuracy on IMDB celebrity data set[33], emotion network has accuracy 73.14% on fer2013 emotion data set [35]. Age estimation performance can be seen in Table 3, these values are not as good as authors had in their paper [5], this can be accredited to CORAL being trained on different sized crop than RetinaFace bounding box. This can be fine-tuned in the future. Overall the most underperforming prediction is age estimation but that is also the most difficult task. All data sets yielded good and bad results depending on person reflecting their data set nuances, this can be seen on Figure 6. CORAL Weights trained by author had yielded similar results to weights of original authors. Finally Table 4 and Figure 7, shows proof of improvement in CNN performance by introducing LSTM element, to harness temporal information. This result could be improved by enlarging video data set and changes to training.

**Table 2.** RetinaFace AP on validation data set WIDER FACE [39]

| Architecture | Easy | Medium | Hard |
|---|---|---|---|
| Resnet-50 | 95.482% | 94.046% | 84.430% |
| MobileNet-0.25 | 90.708% | 88.165% | 73.827% |

**Table 3.** CORAL network prediction error on section of UTK data set [40] in age range 21-40 (13147 images), for all training data sets options

| Training data set | MAE | RMSE |
|---|---|---|
| AFAD | 5.6004 | 7.0212 |
| CACD | 9.5130 | 11.9372 |
| MORPH | 7.4501 | 9.0592 |
| UTK | 6.3496 | 8.7732 |
| UTK(1-100) | 6.3516 | 8.7429 |

**Table 4.** Comparison between original and LSTM modified CORAL model, evaluated on new test video data set, 35 sequences - 35 unique individuals

| CORAL Architecture | MAE | RMSE |
|---|---|---|
| Original | 10.43 | 13.87 |
| Modified | 9.26 | 11.76 |

**Figure 7.** Script output showing different age predictions for original (left) and modified (right) CORAL architecture (trained on UTK(1-100), modified LSTM additionally trained on new video data set))

Inference speed of individual parts of image evaluation is on Table 1. Lightweight image analysis is around 0.1 s, but this highly depends on number of faces per image, so for more people in image prediction it could be problematic for real time use. Inference with CPU for more than 2 people is not usable and modified variant is not usable at all. This need for speed can be solved by further optimizing code between CNN passes or better hardware resources allocation. This can be done by not using neural frameworks by loading models as weights in ONNX format into performance-focused inference engine like ONNX Runtime [42], which can run atop of NVIDIA TensorRT Inference Accelerator [43].

Solution with LSTM, is now usable only for one person per image, because input needs time sequence of 20 bounding boxes of one person. So code should be updated to solve this issue for multiple persons.

This work in future expects to optimize inference speed and experiment with LSTM layer of CORAL network. Goal is to achieve live video feed inference performance with accuracy focused on age estimation.

## Acknowledgements

I would like to thank my supervisor doc. Ing. Radim Kolář Ph.D. for his help.

## References

[1] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M. and Kudlur, M., 2016. *Tensorflow: A system for large-scale machine learning*. In 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16) (pp. 265-283).

[2] Afifi, M. and Abdelhamed, A., 2019. *AFIF4: deep gender classification based on AdaBoost-based fusion of isolated facial features and foggy faces*. Journal of Visual Communication and Image representation, 62, pp.77-86.

[3] Antipov, G., Baccouche, M., Berrani, S.A. and Dugelay, J.L., 2017. *Effective training of convolutional neural networks for face-based gender and age prediction*. Pattern Recognition, 72, pp.15-26.

[4] Arriaga, O., Valdenegro-Toro, M. and Plöger, P., 2017. *Real-time convolutional neural networks for emotion and gender classification*. arXiv preprint arXiv:1710.07557..

[5] Cao, W., Mirjalili, V. and Raschka, S., 2019. *Consistent rank logits for ordinal regression with convolutional neural networks*. arXiv preprint arXiv:1901.07884.

[6] Chen, B.C., Chen, C.S. and Hsu, W.H., 2014, September. *Cross-age reference coding for age-invariant face recognition and retrieval*. In European conference on computer vision (pp. 768-783). Springer, Cham.

[7] Chen, J.C., Kumar, A., Ranjan, R., Patel, V.M., Alavi, A. and Chellappa, R., 2016, September. *A cascaded convolutional neural network for age estimation of unconstrained faces*. In 2016 IEEE 8th International Conference on Biometrics Theory, Applications and Systems (BTAS) (pp. 1-8). IEEE.

[8] S. Chen, C. Zhang, M. Dong, J. Le and M. Rao, "*Using Ranking-CNN for Age Estimation*," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, 2017, pp. 742-751. doi: 10.1109/CVPR.2017.86

[9] François Chollet, 2015 *Keras* GitHub, GitHub repository, [Online],

https://github.com/fchollet/keras, commit = 5bcac37

[10] Chu, W. and Keerthi, S.S., 2005, August. *New approaches to support vector ordinal regression*. In Proceedings of the 22nd international conference on Machine learning (pp. 145-152). ACM.

[11] Crammer, K. and Singer, Y., 2002. *Pranking with ranking*. In Advances in neural information processing systems (pp. 641-647).

[12] Deng, J., Dong, W., Socher, R., Li, L.J., Li, K. and Fei-Fei, L., 2009, June. *Imagenet: A large-scale hierarchical image database*. In 2009 IEEE conference on computer vision and pattern recognition (pp. 248-255). Ieee.

[13] Deng, J., Guo, J., Zhou, Y., Yu, J., Kotsia, I. and Zafeiriou, S., 2019. *RetinaFace: Single-stage Dense Face Localisation in the Wild*. arXiv preprint arXiv:1905.00641.

[14] Everingham, M., Van Gool, L., Williams, C.K., Winn, J. and Zisserman, A., 2007. *The PASCAL visual object classes challenge 2007 (VOC2007) results*.

[15] Fan, Y., Lu, X., Li, D. and Liu, Y., 2016, October. Video-based emotion recognition using CNN-RNN and C3D hybrid networks. In Proceedings of the 18th ACM International Conference on Multimodal Interaction (pp. 445-450).

[16] FU, Cheng-Yang, et al. *Dssd: Deconvolutional single shot detector*. arXiv preprint arXiv:1701.06659, 2017.

[17] HOCHREITER, Sepp; SCHMIDHUBER, Jürgen. *Long short-term memory*. Neural computation, 1997, 9.8: 1735-1780.

[18] Jain, V. and Learned-Miller, E., 2010. *Fddb: A benchmark for face detection in unconstrained settings*.

[19] Ko, B.C., 2018. *A brief review of facial emotion recognition based on visual information*. sensors, 18(2), p.401.

[20] Levi, G. and Hassner, T., 2015. *Age and gender classification using convolutional neural networks*. In Proceedings of the IEEE conference on computer vision and pattern recognition workshops (pp. 34-42).

[21] Levi, G. and Hassner, T., 2015, November. *Emotion recognition in the wild via convolutional neural networks and mapped binary patterns*. In Proceedings of the 2015 ACM on international conference on multimodal interaction (pp. 503-510).

[22] Li, L. and Lin, H.T., 2007. *Ordinal regression by extended binary classification*. In Advances in neural information processing systems (pp. 865-872).

[23] Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P. and Zitnick, C.L., 2014, September. *Microsoft coco: Common objects in context*. In European conference on computer vision (pp. 740-755). Springer, Cham.

[24] Lin TY, Goyal P, Girshick R, He K, Dollár P. *Focal loss for dense object detection*. InProceedings of the IEEE international conference on computer vision 2017 (pp. 2980-2988).

[25] Niu, Z., Zhou, M., Wang, L., Gao, X. and Hua, G., 2016. *Ordinal regression with multiple output cnn for age estimation*. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4920-4928).

[26] PASZKE, A.; GROSS, S.; CHINTALA, S.; et al.: *PyTorch*. [online]. [Cit. 2019-05-10]. Retrieved from: https://pytorch.org

[27] Perez, C., Tapia, J., Estévez, P. and Held, C., 2012. Gender classification from face images using mutual information and feature fusion. International Journal of Optomechatronics, 6(1), pp.92-119.

[28] Polania, L., Fung, G. and Wang, D., 2019, January. *Ordinal Regression using Noisy Pairwise Comparisons for Body Mass Index Range Estimation*. In 2019 IEEE Winter Conference on Applications of Computer Vision (WACV) (pp. 782-790). IEEE.

[29] Ranjan, R., Sankaranarayanan, S., Castillo, C. and Chellappa, R., University of Maryland and College Park, 2019. *All-in-one convolutional neural network for face analysis*. U.S. Patent Application 16/340,859.

[30] REDMON, Joseph; FARHADI, Ali. *Yolov3: An incremental improvement*. arXiv preprint arXiv:1804.02767, 2018.

[31] REN, Shaoqing, et al. *Faster r-cnn: Towards real-time object detection with region proposal networks*. In: Advances in neural information processing systems. 2015. p. 91-99

[32] Ricanek, K. and Tesafaye, T., 2006, April. *Morph: A longitudinal image database of normal adult age-progression*. In 7th International Conference on Automatic Face and Gesture Recognition (FGR06) (pp. 341-345). IEEE.

[33] Rothe, R., Timofte, R. and Van Gool, L., 2018. *Deep expectation of real and apparent age from a single image without facial landmarks*. International Journal of Computer Vision, 126(2-4), pp.144-157.

[34] Shan, C., 2012. *Learning local binary patterns for gender classification on real-world face images*. Pattern recognition letters, 33(4), pp.431-437.

[35] Wolfram Research ,*The Facial Expression Recognition 2013 (FER-2013) Dataset*. From the Wolfram Data Repository (2018).

[36] Ullah, I., Hussain, M., Muhammad, G., Aboalsamh, H., Bebis, G. and Mirza, A.M., 2012, April. *Gender recognition from face images with local wild descriptor*. In 2012 19th International Conference on Systems, Signals and Image Processing (IWSSIP) (pp. 417-420). IEEE.

[37] Uricár, M., Timofte, R., Rothe, R., Matas, J. and Van Gool, L., 2016. *Structured output svm prediction of apparent age, gender and smile from deep features*. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (pp. 25-33).

[38] XU, Yuanyuan, et al. *CenterFace: Joint Face Detection and Alignment Using Face as Point*. arXiv preprint arXiv:1911.03599, 2019.

[39] Yang, S., Luo, P., Loy, C.C. and Tang, X., 2016. *Wider face: A face detection benchmark*. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 5525-5533).

[40] Zhang, Z., Song, Y. and Qi, H., 2017. *Age progression/regression by conditional adversarial autoencoder*. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 5810-5818).

[41] ZHOU, X.,WANG, D., KRÄHENBÜHL, Philipp. *Objects as Points*. arXiv preprint arXiv:1904.07850, 2019.

[42] *ONNX Runtime: cross-platform, high performance scoring engine for ML models* [Online] at https://github.com/microsoft/onnxruntime

[43] *NVIDIA TensorRT™ - SDK for high-performance deep learning inference* [Online] at https://developer.nvidia.com/tensorrt

[44] *Facial recognition: top 7 trends (tech, vendors, markets, use cases and latest news)* [Online] at https://www.gemalto.com/govt/biometrics/facial-recognition

[45] *Google Scholar - Internet search engine for scientific papers* [Online] at https://scholar.google.com/

[46] *0-100 series, SoulPancace,Youtube* [Online] at https://www.youtube.com/playlist?list=PLzvRx-johoA-7DXw5EThWpQlVYH0hm1aC

Fakulta informačních technologií
Vysoké učení technické v Brně

FIT

2020

Božetěchova 1/2, Brno 612 66