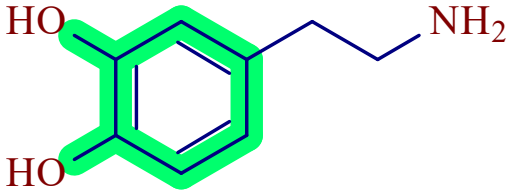
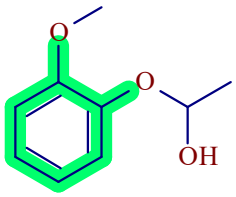
BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2018
http://excel.fit.vutbr.cz

Podštruktúrne vyhľadávanie v databázach chemických látok

Ivan Ševčík*





Abstrakt

V tomto článku je predstavený spôsob, pomocou ktorého je možné vyhľadávať zadaný vzor ako podštruktúru v databázach chemických látok obsahujúcich desiatky miliónov záznamov. Jedná sa o výpočetne náročný problém, ktorý je potrebné riešiť na niekoľkých úrovniach. Metóda prezentovaná v tomto článku využíva kombináciu topologických indexov, tzv. molekulárnych odtlačkov, a rýchleho algoritmu pre hľadanie izomorfného podgrafu. Práca porovnáva niekoľko algoritmov, ako aj spôsobov uloženia molekúl v systéme. Najlepším riešením sa ukázalo byť použitie algoritmu zvaného RI v kombinácii s uložením serializovaných dát do databázy MongoDB. Riešenie umožňuje užívateľom vyhľadávať v databázach v reálnom čase, do niekoľkých sekúnd. Cieľom tohto článku je oboznámiť čitateľa s problematikou vyhľadávania v chemických databázach a ako je možné vytvoriť takýto systém pre vyhľadávanie s ohľadom na súčasný výskum. Výsledky a myšlienky práce by mohli byť využité v príbuzných oblastiach vyžadujúcich vyhľadávanie vzorov v databázach, ako je napríklad počítačové videnie.

Kľúčové slová: Databáza — Vyhľadávanie vzorov — Grafový izomorfizmus

Priložené materiály: Dodatočné zdroje

*xsevci50@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Úvod

Práca sa zaoberá popisom systému, ktorý umožňuje vyhľadávať vzory ako podštruktúry v databázach chemických látok v prijateľnom čase. Podštruktúrne vyhľadávanie má široké uplatnenie. Pomáha nájsť biologicky aktívne alebo toxické látky, urýchľuje výskum nových liečiv a umožňuje identifikovať látky s chemicky podobnými vlastnosťami. Táto úloha je pomerne dobre preskúmaná a je známy obecný prístup k jej riešeniu. Pri implementácii je však potrebné zvoliť konkrétne metódy a algoritmy tak, aby sa dosiahlo čo najlepších výsledkov. Súčasťou práce je preto okrem predstavenia návrhu samotného systému umožňujúceho podštruktúrne vyhľadávanie aj vyhodnotenie a porovnanie dostupných metód s ohľadom na súčasný stav výskumu.

Vyhľadanie vzoru v chemickej štruktúre je náročná operácia a problém tiež predstavuje neustále narastajúca veľkosť databáz. Chemické štruktúry sú najčastejšie reprezentované formou neorientovaného grafu a vyhľadanie vzoru odpovedá nájdeniu izomorfného podgrafu. Jedná sa o NP-úplný problém. Výsledkom vyhľadávania je množina všetkých látok uložených v databáze, ktoré obsahujú zadaný vzor ako podgraf. Pretože je táto operácia často súčasťou rôznych výpočtov a predikcií, je nutné aby bola dokončená v čo najkratšom čase. V databáze obsahujúcej sto miliónov záznamov by jeden takýto dotaz nemal trvať dlhšie ako

niekoľko sekúnd. K tomu je nutné nájsť nielen rýchly algoritmus, ale aj vhodný topologický index, ktorý umožní predvýber kandidátov, a taktiež optimalizovať spôsob uloženia a načítania molekulárnych dát.

Existuje niekoľko online chemických databáz, ktoré podporujú podštruktúrne vyhľadávanie. Medzi jednu z najväčších patrí databáza PubChem[1]. Jedná sa o uzavreté systémy, ktoré poskytujú požadovanú funkcionalitu, ale sú naviazané na konkrétnu databázu a ich implementácia nie je verejne dostupná. Hlavnou nevýhodou sú obmedzenia na využívanie týchto databáz. Napríklad spomínaná databáza PubChem má obmedzenie na svoje REST API na maximálne päť požiadavkov za sekundu, 400 za minútu, a spoločný výpočetný čas operácií nesmie presiahnuť 300 sekúnd za minútu, čo môže nastať pri paralelnom behu požiadavkov [1].

Okrem online sú dôležité aj offline databázy, ktoré sú zdieľané napríklad vo forme súboru. Jedná sa prevažne o súkromné databázy uložené zväčša vo formáte SDF. K týmto databázam neexistuje odpovedajúci vyhľadávací nástroj. Riešením môže byť použitie jednej z chemoinformatických knižníc, pomocou ktorých je možné takýto nástroj vytvoriť. Populárna knižnica, od ktorej mnohé ďalšie preberajú koncepty alebo implementáciu, je knižnica RDKit[2]. Zatiaľ čo vytvorenie takéhoto nástroja pre osobné použitie je vďaka knižnici pomerne jednoduché, problém nastáva, ak je potrebné systém prispôsobiť pre konkrétne potreby alebo výrazne škálovať. Napriek tomu predstavuje RDKit vhodnú východiskovú pozíciu. Okrem samotného podštruktúrneho vyhľadávania, ktoré je založené na algoritme VF2, poskytuje knižnica taktiež zásuvný modul pre PostgreSQL databázu, ktorá slúži pre uloženie štruktúrnych dát a ďalších informácií o látkach. Tento modul sprístupňuje vyhľadávanie na vysokej abstraktnej úrovni pomocou SQL dotazov a samotné vyhľadanie potom prebieha priamo v procese databázového servera. To ale taktiež znamená, že pri veľkom počte dotazov hrozí preťaženie tohto jedného servera.

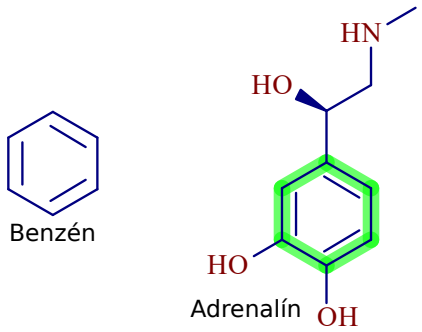
V tomto článku je predstavené modulárne riešenie, v ktorom je každá časť systému zodpovedná za špecifickú úlohu, čo dáva možnosť jednoduchšie škálovať podľa konkrétnej potreby. Tento prístup tiež umožňuje systém oveľa lepšie optimalizovať a zvoliť nástroje, ktoré sú pre daný problém najvhodnejšie. Samotné vyhľadávanie používa rovnaký topologický index ako knižnica RDKit. Pre nájdenie izomorfného podgrafu je však použitý algoritmus RI, ktorý je v porovnaní s algoritmom VF2 schopný nájsť riešenie až v trojnásobne kratšom čase.

[1] https://pubchem.ncbi.nlm.nih.gov/
[2] http://www.rdkit.org/

Výsledkom práce je demonštrácia systému, ktorý umožňuje užívateľom zakresliť požadovaný vzor a vyhľadať ho v databáze. Dôležitou súčasťou je množina meraní, ktoré umožňujú zodpovedať otázky ohľadom voľby algoritmov a technológií.

2. Podštruktúrne vyhľadávanie

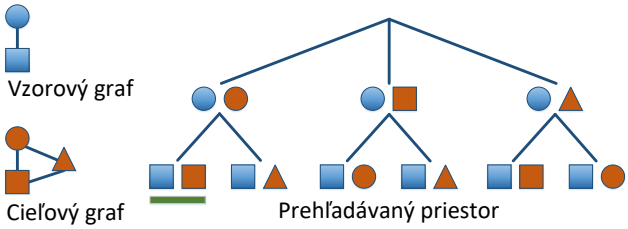
Ako bolo spomenuté v úvode, chemické štruktúry zvyknú byť reprezentované formou dvojrozmerného neorientovaného grafu, kde vrcholy predstavujú atómy a hrany väzby medzi atómami. Bežnú definíciu grafu je nutné navyše rozšíriť tak, aby každý vrchol a hrana mohli niesť množinu atribútov a hodnôt, ktoré popisujú ich vlastnosti. Atómy sú totiž rôzne prvky, zatiaľ čo väzby medzi atómami majú rôznu násobnosť. Na obrázku 1 je možné vidieť chemickú štruktúru pre adrenalín.



Obrázok 1. Vpravo: Chemická štruktúra adrenalínu obsahujúca atómy rôznych prvkov, aromatické väzby ako aj stereochémiu. Vľavo: Štruktúra benzénu. Zeleným je zvýraznený jeho výskyt v adrenalíne ako podštruktúra.

Úlohou podštruktúrneho vyhľadávania je zistiť, či daná chemická štruktúra obsahuje zadaný vzor. Keďže sú chemické štruktúry reprezentované ako grafy, táto úloha odpovedá nájdeniu izomorfného podgrafu. Napriek tomu, že sa jedná o NP-úplny problém, existujú algoritmy, ktoré ho umožňujú prakticky riešiť. Ullmann ako prvý navrhol takýto algoritmus [2]. Jeho princípom je prerezávanie stromu predstavujúceho prehľadávaný priestor (obrázok 2) na základe splnenia určitej podmienky. Ak dôjde k jej porušeniu, algoritmus už nemusí skúšať priradenie zvyšných vrcholov, čím výrazne redukuje počet porovnaní.

Ullmannov algoritmus bol dlho jeden z najpoužívanejších, ale medzi jeho nevýhody patrí hlavne vysoká pamäťová náročnosť. V roku 1999 bol v článku [4] predstavený algoritmus VF, a následne v roku 2004 jeho vylepšená verzia VF2 [5], ktoré zlepšujú vlastnosti Ullmanovho algoritmu a riešia aj pamäťovú náročnosť. Tento algoritmus sa stal viac-menej štandardnou implementáciou pre hľadanie izomorfných

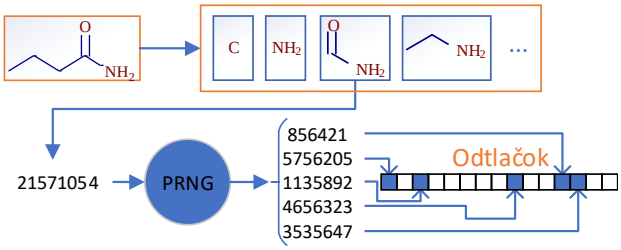


Obrázok 2. Prehľadávaný priestor ako strom. Zelené podtrhnutie označuje vetvu, v ktorej dôjde k nájdeniu izomorfizmu. [3]

podgrafov v chemických aplikáciách, a je možné ho nájsť použitý v populárnych knižniciach ako sú RDKit alebo OpenBabel. Výskum však naďalej pokračoval a boli vyvinuté mnohé ďalšie algoritmy, ako napr. SND, LAD alebo FocusSearch. Zaujímavý je ale predovšetkým algoritmus RI predstavený v roku 2011, ktorý je podľa meraní v kontexte biochemických aplikácií lepší než všetky uvedené algoritmy [3]. Jeho princípom je zostavenie vyhľadávacej stratégie na základe vzorového grafu v kombinácii s jednoduchými a rýchlymi obmedzujúcimi podmienkami. Rozhodli sme sa preto porovnať algoritmus RI so zaužívaným VF2 priamo pri probléme vyhľadávania v databázach chemických látok.

3. Molekulárne odtlačky

Aj rýchlemu algoritmu pre hľadanie izomorfného podgrafu by trvalo veľmi dlhú dobu porovnať všetky záznamy v databáze. Ak budeme uvažovať, že by jedna inštancia problému trvala priemerne 10 µs, pre jeden dotaz v databáze o veľkosti 10 miliónov záznamov by iba samotný beh algoritmu v najlepšom prípade spotreboval 100 sekúnd výpočetného času. S narastajúcou databázou by potom tento čas ďalej lineárne narastal. Riešením tohto problému je použitie molekulárnych odtlačkov. Často sa jedná o bitové polia, kde na jednotlivých bitoch sú zakódované rôzne dôležité topologické vlastnosti molekúl. Existuje mnoho druhov odtlačkov, no len niektoré z nich sú vhodné pre podštruktúrne vyhľadávanie. Jeden z najrozšírenejších typov odtlačkov popísala firma Daylight. Jeho podstatou bolo kódovanie ciest v grafe postupne narastajúcej dĺžky do odtlačku [6]. Keďže sa ale jedná o komerčný produkt, jeho implementácia nebola voľne dostupná. Mnohé knižnice si preto vytvorili vlastné odtlačky tohto typu. Odtlačok pre podštruktúrne vyhľadávanie v knižnici RDKit, `PatternFingerprint`, pracuje na podobnom princípe, ktorý zachytáva obrázok 3. Ako prvé sa vytvorí odtlačok, čiže bitové pole požadovanej dĺžky a všetky bity sa inicializujú na hodnotu 0. Pomocou algoritmu sa nájdu všetky výskyty niekoľkých preddefinovaných vzorov, ktoré značia významné topológie. Následne sa každý výskyt spolu s vlastnosťami atómov a väzieb zahašuje na 32-bitové číslo, ktoré sa použije pre inicializáciu pseudonáhodného generátora čísel. Ten následne vygeneruje určený počet čísel, ktoré po modulu s dĺžkou odtlačku slúžia ako indexy do bitového pola a do bitov na týchto pozíciach sa zapíše hodnota 1.



Obrázok 3. Princíp vytvárania podštruktúrneho odtlačku v knižnici RDKit. Preddefinované vzory sa vyhľadajú v štruktúre a každý výskyt sa zahašuje na 32-bitové číslo, ktoré inicializuje PRNG. Ten vygeneruje určený počet čísel slúžiacich ako indexy do bitového poľa, kde dôjde k zmene hodnoty bitu.

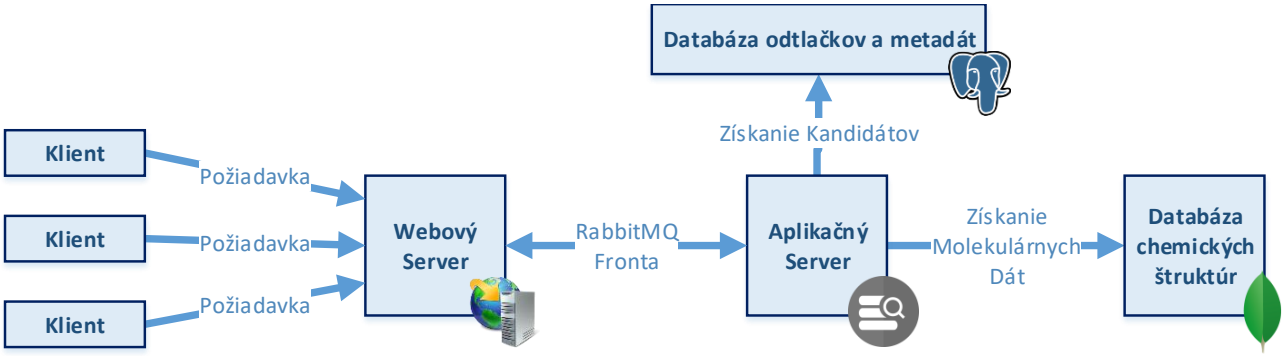
Samotný princíp použitia odtlačkov je pomerne priamočiary. Pre vzor sa určí odtlačok a následne sa porovnáva s tými uloženými v databáze. Iba ak všetky bity s hodnotou 1 v odtlačku vzoru sú nastavené na 1 aj v odtlačku chemickej štruktúry, má zmysel použiť algoritmus pre vyhľadanie vzoru. V opačnom prípade sa vzor v štruktúre určite nenachádza. Takto je možné z veľkej databázy získať pomerne malú množinu kandidátov, a tým výrazne obmedziť počet spustení algoritmu pre vyhľadanie podgrafu.

Presnosť odtlačku z knižnice RDKit je približne 60%, čo znamená, že 40% kandidátov neobsahuje hľadaný vzor [7]. Ak by však databáza mala desať miliónov záznamov a počet skutočných výsledkov by bol rádovo v jednotkách tisícov, použitie odtlačku by zredukovalo počet prehľadávaných štruktúr o viac ako 99%. V takom prípade by aj pomerne zlý odtlačok s presnosťou iba 5% dokázal výrazne zmenšiť prehľadávanú množinu.

4. Spôsob uloženia molekúl

Chemické látky sú naprieč chemoinformatickými systémami zdieľané najčastejšie pomocou textového formátu MDL. Ten najskôr popisuje vlastnosti každého atómu a následne jednotlivé väzby ako indexy koncových atómov spolu s vlastnosťami väzby [8]. Viacero molekúl v tomto formáte je potom možné zoskupiť do jedného súboru SDF. Jeho následnou kompresiou, napríklad metódou DEFLATE, je možné dosiahnuť uspokojivej veľkosti jedného záznamu.

K molekulám je však nutné v procese vyhľadávania



Obrázok 4. Architektúra systému. Webový server prijíma požiadavky na vyhľadávanie od klientov a plní nimi RabbitMQ frontu. Aplikačný server vyberie požiadavku z fronty a začne s jej spracovaním. V prvom kroku získa pomocou odtlačkov množinu kandidátov z PostgreSQL databáze. Následne týchto kandidátov, pokiaľ nie sú v pamäti, načíta z MongoDB databáze chemických štruktúr. Pomocou algoritmu pre hľadanie izomorfného podgrafu skontroluje výskyt vzoru v kandidátoch a výslednú množinu uloží k požiadavke. Pri nasledovnom dotazovaní klienta na výsledok operácie je mu táto množina predaná.

rýchlo pristupovať. Dátové štruktúry reprezentujúce molekuly naprieč systémom sú pamäťovo náročné, pretože bývajú optimalizované na rýchlosť a jednoduchosť použitia. Udržiavať takúto reprezentáciu v pamäti pre všetky záznamy by kládlo príliš veľké nároky na jej veľkosť. Je preto zrejmé, že bude nutné molekuly komprimovať, a až v prípade potreby vytvoriť odpovedajúcu dátovú štruktúru. Uvedený formát pre zdieľanie molekúl nie je na tieto účely vhodný, pretože jeho textová reprezentácia vyžaduje parsovanie textových reťazcov. Lepším riešením je serializovať dáta do binárneho formátu, akým je napríklad MessagePack. V prostredí .Net existuje pre tento účel knižnica MessagePack-CSharp[3], ktorá navyše podporuje následnú kompresiu algoritmom LZ4.

Okrem voľby formátu je tiež nutné uvažovať, akým spôsobom dáta serializovať, aby doba spätného získania plnej reprezentácie ako aj veľkosť po serializácii boli čo najmenšie. Z tohto dôvodu bolo vyskúšaných niekoľko rôznych prístupov, ktoré je možné nájsť v repozitári odkazovanom v abstrakte práce. Vyhodnotenie spolu s porovnaním s komprimovaným formátom SDF je možné nájsť v sekcii 6.

Aj keď sa vďaka kompresii výrazne znížia pamäťové nároky, pri priemernej veľkosti 500 B na záznam by systémy s databázami dosahujúcimi sto miliónov záznamov museli mať desiatky gigabajtov dostupnej pamäte iba na uloženie molekúl. Je preto nutné uvažovať s možnosťou, že nie všetky záznamy budú vždy v pamäti. V tom prípade je potrebné mať čo najrýchlejší spôsob prístupu k nim. V práci bolo vykonané porovnanie uloženia záznamov priamo v súborovom systéme, kde každý súbor odpovedal jednej molekule, a v NoSQL databáze MongoDB, kde sú molekuly uložené ako samostatné dokumenty.

[3] https://github.com/neuecc/MessagePack-CSharp

5. Celkový návrh systému

Systém sa celkovo skladá zo štyroch častí, ktoré je možné vidieť na obrázku 4. Webový server sprístupňuje systém užívateľom vo forme webových stránok a poskytuje API, cez ktoré klient spúšťa vyhľadávanie a dotazuje sa na jeho výsledok. Aplikačný server je zodpovedný za samotné vyhľadávanie v databáze. Pre požadovaný vzor získa množinu kandidátov, v ktorej následne skontroluje výskyt vzoru pomocou algoritmu pre hľadanie izomorfného podgrafu. Odtlačky a metadáta o chemických látkach sú uložené v PostgreSQL databáze. Tá je zodpovedná aj za určenie množiny kandidátov, ktorú vráti ako zoznam identifikátorov štruktúr. K tomu bolo potrebné implementovať zásuvný modul, ktorý podporuje operáciu porovnania odtlačkov. V prípade, že sa potrebné chemické štruktúry nenachádzajú na aplikačnom serveri, sú získané z MongoDB databáze. Tá obsahuje iba samotné štruktúrne dáta.

Rozdelenie na dva databázové systémy má svoje opodstatnenie. PostgreSQL databáza umožňuje vytvorenie zásuvného modulu, v ktorom sú vstavané funkcie implementované v jazyku C a skompilované do dynamickej knižnice. Takáto funkcia je výrazne rýchlejšia, než odpovedajúca implementácia v SQL. MongoDB vstavané funkcie nepodporuje, preto by nebolo možné určiť kandidátov priamo v databáze, čo by vyžadovalo prenos všetkých odtlačkov na aplikačný server. Na druhú stranu ak je úlohou databázy iba nájsť určité kľúče a vrátiť asociované polia bajtov, ako je to v prípade databázy chemických štruktúr, javí sa použitie nerelačnej databázy ako lepšie riešenie.

V prípade MongoDB je jednoduchšie aj horizontálne škálovanie, čiže pridávanie ďalších databázových serverov za účelom zvýšenia priepustnosti.

Webový a aplikačný server spolu komunikujú pomocou RabbitMQ fronty. RabbitMQ je open-source riešenie pre sprostredkovanie správ. Po prijatí požiadavky od klienta vytvorí webový server požiadavku na aplikačný server, v ktorej uvedie parametre vyhľadávania, a vloží ju do fronty. Aplikačný server sa pri spustení prihlási k odberu týchto správ a zaviaže sa odpovedať. Po prijatí požiadavky spustí vyhľadávanie a vráti identifikátor operácie, ktorý je predaný klientovi. Ten ho použije pre následné dotazovanie sa na výsledok operácie. Toto riešenie umožňuje jednoducho pridávať do systému ďalšie aplikačné servery. Vo východzom nastavení budú požiadavky vo fronte rovnomerne rozdelené medzi dostupné aplikačné servery.

Nevýhodou rozdelenia systému je, že jednotlivé časti musia medzi sebou navzájom komunikovať, čo pridáva určitú réžiu. Taktiež správa takéhoto systému je náročnejšia. Na druhú stranu tento prístup umožňuje efektívnejšie škálovanie, kedy napríklad nie je nutné replikovať celú databázu, ak je potrebné zvýšiť priepustnosť v algoritmickej časti.

6. Výsledky meraní

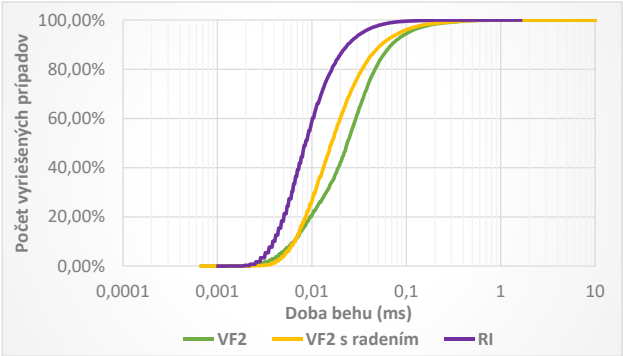
Všetky nasledujúce merania boli vykonané na počítači s touto špecifikáciou:

- CPU – AMD Ryzen 5 1600
- RAM – 8GB, DDR4, 2933Mhz
- HDD – WD5000AAKS, 500GB, 7200 otáčok/s
- SSD – Samsung 850 EVO, 500GB

Prvým meraním bolo porovnanie dvoch algoritmov pre hľadanie izomorfného podgrafu a ich variánt. Konkrétne sa jednalo o algoritmy VF2, VF2 s radením, RI a teoretický algoritmus Best, ktorý pre konkrétnu dvojicu vždy zvolí najrýchlejší algoritmus spomedzi dostupných. Radenie pri VF2 spočíva v zoradení prehľadávaných vrcholov podľa protónového čísla prvku a stupňa vrcholu. Menej bežné prvky s veľkým počtom susedov sa tak navštívia ako prvé. Meranie bolo vykonané nad reálnou databázou naplnenou miliónom chemických látok získaných z databáze PubChem. Ako vzory bolo použitých tisíc najčastejších podštruktúr z rovnakej databáze. Tie predstavujú najhorší možný prípad pri vyhľadávaní, pretože budú nájdené vo veľkom množstve záznamov. Zoznam týchto podštruktúr bol získaný z článku [9]. Aby merania odpovedali reálnemu prípadu užitia, algoritmus bol spustený iba pre záznamy, ktoré prešli testom odtlačkov. Pre každú dvojicu grafov bolo meranie zopakované päťkrát a z nameraných časov bol zvolený medián, aby prípadné vyťaženie procesora iným procesom nemalo na meranie vplyv. Priemerná hodnota by v prípade namerania výnimočne dlhého času bola skreslená. Následne bola pre každý algoritmus vypočítaná priemerná doba behu a počet prípadov, kedy bol daný algoritmus najrýchlejší spomedzi všetkých. Celkový počet inštancií bol 6472522 a vzor sa našiel v 821939 prípadoch. Výsledky sú uvedené v tabuľke 1. Obrázok 5 zobrazuje namerané časy formou kumulatívneho histogramu. Ako je možné vidieť, algoritmus RI je najrýchlejší a za teoretickým algoritmom zaostáva iba o necelých 5 %.

Tabuľka 1. Porovnanie algoritmov pre hľadanie izomorfného podgrafu. Doba behu je priemerná hodnota pre jednu inštanciu a stĺpec najrýchlejší uvádza, koľko inštancií vyriešil daný algoritmus v najkratšom čase.

Algoritmus	Doba behu (µs)	Najrýchlejší (#)
VF2	37.014	734941
VF2 s radením	29.088	96473
RI	12.817	5641108
Best	12.259	



Obrázok 5. Porovnanie algoritmov pre hľadanie izomorfného podgrafu formou kumulatívneho histogramu. Interpretovať ho je možné ako počet inštancií, pre ktoré doba behu neprekračuje určitý čas. Napríklad v prípade RI neprekračuje doba behu 10 µs v 60 % všetkých inštancií.

Ďalej bolo potrebné vykonať porovnanie jednotlivých formátov pre uloženie štruktúrnych dát predstavených v časti 4. Sledovanými parametrami boli výsledná veľkosť v bajtoch a rýchlosť opätovného získania plnej dátovej štruktúry. Formát MPM je založený na spomínanom serializačnom formáte MessagePack s využitím LZ4 kompresie. Rozdiel medzi jednotlivými verziami je iba v spôsobe uloženia vlastností atómov, preto väzby neboli súčasťou formátu pri tomto porovnaní. Meranie bolo vykonané na 100000 náhodných molekulách z databáze PubChem a de-

saťkrát zopakované. Spriemerované výsledky je možné nájsť v tabuľke 2. Ako najoptimálnejšia verzia formátu sa javí MPM 3, kedy sú súradnice a prvky uložené v poliach spoločných pre celú štruktúru a menej bežné vlastnosti atómov, ako napr. izotopy, sú uložené vo forme riedkych polí, kde každý prvok obsahuje index atómu a hodnotu danej vlastnosti.

Tabuľka 2. Porovnanie viacerých spôsobov serializácie údajov o atómoch v chemických štruktúrach.

Formát	Veľkosť (B)	Deserializácia (µs)
MPM 1	553.167	12.73
MPM 2	540.187	10.411
MPM 3	400.58	8.63
MPM 4	519.614	7.91

Po pridaní informácií o väzbách bolo možné formát MPM porovnať s formátom SDF komprimovaným pomocou metódy DEFLATE. Aby bolo porovnanie spravodlivejšie, bol dátový typ pre súradnice vo formáte MPM zmenený z `double` na `int`, keďže SDF podporuje pre súradnice iba hodnoty s pevnou desatinnou čiarkou, ktorých rozsah sa zmestí do `int`. Porovnanie je možné nájsť v tabuľke 3. Meranie bolo vykonané na 5000 náhodných molekulách a výsledky sú spriemerované z desiatich opakovaní. Výsledný formát má oproti komprimovanému SDF o 27 % menšiu veľkosť a umožňuje až trojnásobne rýchlejšiu deserializáciu do potrebnej dátovej štruktúry.

Tabuľka 3. Porovnanie formátov MPM a SDF. Formáty, u ktorých nebol meraný čas deserializácie, sú uvedené iba pre porovnanie veľkosti, ich použitie by vzhľadom na existenciu lepšej varianty nedávalo zmysel.

Formát	Veľkosť (B)	Deserializácia (µs)
SDF	4431.555	Nemerané
SDF (DEFLATE)	636.61	259.283
MPM (`double`)	614.941	Nemerané
MPM (`int`)	465	64.91

Ďalšie meranie malo za úlohu rozhodnúť, aký spôsob uloženia chemických štruktúr zvoliť. Prvá možnosť bolo uloženie priamo do súborového systému NTFS, preto bolo potrebné zmerať, či a ako sa mení prístupová doba k súborom v závislosti na počte súborov v zložke. Okrem jednoduchých zložiek bola meraná prístupová doba aj v stromovej štruktúre s dvomi úrovňami, kde každá zložka obsahovala ďalších sto zložiek alebo súborov. Merania boli vykonané pre platňový disk aj SSD. Otvorených bolo 50 náhodných súborov v zložke a meranie zopakované stokrát. Výsledná doba v tabuľke 4 je priemerom všetkých meraní. Na základe výsledkov je možné povedať, že optimálny počet súborov v zložke sa pohybuje okolo 1000. Platňový disk umožňuje aj väčšie zložky, ale nad 10000 súborov prístupová doba drasticky narastá. V prípade SSD disku je tento nárast pozvoľnejší. Vytvorenie stromovej štruktúry nemalo na prístupovú dobu žiadny vplyv, preto je možné túto stratégiu využiť na zvyšovanie kapacity úložiska.

Tabuľka 4. Prístupová doba v µs k súboru v závislosti na počte súborov v zložke pre súborový systém NTFS.

Počet	100	500	1000	5000
HDD	97.478	103.041	106.291	117.142
SSD	102.626	113.461	129.938	175.376
Počet	**10000**	**100000**	**300000**	**Strom (100)**
HDD	120.725	1059.978	3485.66	96.07
SSD	214.802	455.916	529.47	110.796

Tabuľka 5. Priemerná doba získania serializovaných dát chemickej štruktúry z úložiska. Metóda `ReadAllBytes` číta obsah súborov v stromovej štruktúre. Metóda `Find` získava dáta z databáze MongoDB, kde číslo uvádza počet súbežne získaných záznamov.

Metóda	Doba získania dát (µs)
`ReadAllBytes`	211.936
`Find` 1	234.376
`Find` 10	35.108
`Find` 100	13.408
`Find` 1000	12.137

Po určení vhodnej veľkosti zložiek bolo možné vytvoriť úložisko v súborovom systéme a porovnať ho s databázou MongoDB. V súborovom systéme na SSD disku bola vytvorená dvoj-úrovňová stromová štruktúra, kde každá zložka obsahovala najviac 1000 súborov. Každá chemická štruktúra bola uložená ako jeden súbor, ktorého obsah sa čítal v prostredí .Net pomocou funkcie `File.ReadAllBytes`. V databáze MongoDB uloženej taktiež na SSD disku bola vytvorená nová kolekcia záznamov, ku ktorej sa pristupovalo pomocou oficiálneho MongoDB ovládača pre .Net. K získaniu požadovaných záznamov bola použitá funkcia `collection.Find`, ktorej bol ako parameter predaný filter so zoznamom identifikátorov požadovaný záznamov. Keďže tento spôsob narozdiel od súborového úložiska umožňuje získať viacero záznamov jedným príkazom, boli zmerané časy pre rôzny počet súbežne získaných záznamov. Do oboch

dátových úložisiek bolo uložených milión chemických štruktúr s priemernou veľkosťou 524 B v serializovanej reprezentácii. Následne sa merala doba získania serializovaných dát pre 10000 náhodných chemických štruktúr, každé meranie bolo desaťkrát zopakované a výsledky spriemerované. V tabuľke 5 je možné nájsť výslednú priemernú dobu načítania dát jednej chemickej štruktúry pre uvedené metódy.

Ako je možné vidieť vo výsledkoch, získanie dát pre jeden záznam v databáze približne odpovedá získaniu dát zo súboru, avšak pri vyžiadaní viacerých záznamov z databáze zároveň je databázové riešenie rádovo rýchlejšie. Zrýchlenie je najvýraznejšie medzi získaním jenoho a desiatich záznamov. S ďalej sa zvyšujúcim počtom je postupne toto zrýchlenie menej výrazné, zatiaľ čo rastú pamäťové nároky na uloženie dát. Ako optimálna sa preto javí hodnota medzi 100 až 1000 súbežne získanými záznamami.

Po sčítaní doby získania štruktúrnych dát z databázy, ich deserializácie a priemernej doby behu algoritmu RI zistíme, že doba spracovania jedného kandidáta sa pohybuje okolo 91 µs. Celková doba vyhľadávania potom závisí od počtu kandidátov, ktorých je nutné skontrolovať. V prípade použitia odtlačku s vysokou presnosťou bude ich počet závisieť predovšetkým od veľkosti databáze a vstupného vzoru. V prípade zadania vzoru, ktorý je reálne možné nájsť ako podštruktúru v 30 % databázových záznamov bude vyhľadávanie z princípu trvať dlhú dobu. Takéto obecné vzory ale nebývajú pre užívateľov príliš zaujímavé. Navyše aj v prípade množiny najčastejších vzorov, spomenutej začiatkom tejto časti, bol v 80 % prípadov počet kandidátov menší ako 1 % veľkosti databáze. Ak uvážime databázu s 10 miliónmi záznamov, 1 % by predstavovalo sto tisíc kandidátov. Pre takúto kandidátnu množinu by doba spracovania bola 9.1 s a tento čas by bolo možné ďalej skrátiť paralelným spracovaním vo viacerých vláknach. Experimenty na takto veľkých kandidátnych množinách potvrdili predpovedanú dobu vyhľadávania.

7. Záver

V tomto článku bol predstavený celkový návrh systému pre podštruktúrne vyhľadávanie v databázach chemických látok. Taktiež sa článok zaoberal optimalizáciou jednotlivých častí a vyhodnotením viacerých alternatív.

Algoritmus RI sa ukázal až trikrát rýchlejší oproti zaužívanému algoritmu VF2. V súborovom systéme NTFS po prekročení 10000 súborov v zložke narastá prístupová doba k súboru lineárne s počtom súborov. Riešením je vytvorenie stromových štruktúr pomocou adresárov. V prípade dostatku pamäte RAM však umožňuje NoSQL databáza MongoDB až desaťnásobne rýchlejšie načítanie dát oproti uloženiu štruktúrnych dát do samostatných súborov v súborovom systéme.

Pri optimalizácii systému je nutné uvažovať o každej jeho časti v súvislosti s ostatnými. Použitie rýchlejšieho algoritmu bude mať na zrýchlenie systému minimálny dopad, ak sa väčšinu času čaká na načítanie dát. Taktiež je dôležité vhodne rozdeliť úlohy. Napríklad presunutie kontroly odtlačkov priamo do databáze v konečnom dôsledku zníži jej zaťaženie, pretože sa za cenu výpočtu výrazne zmenší objem prenášaných dát. V súvislosti s tým je vhodné pripomenúť, že aj pomerne zlý topologický index s nízkou presnosťou môže podstatne obmedziť spracovávanú množinu záznamov.

V prípade návrhu podobného systému predstavuje voľba rôznych metód a technológii náročnú úlohu a často sa nezaobíde bez podrobných meraní. Problém riešený v tejto práci je veľmi špecifický, no výsledky a koncepty sú relevantné pre aplikácie zaoberajúce sa hľadaním vzorov v početných databázach grafov.

Poďakovanie

Rád by som poďakoval Ing. Zbyňkovi Křivkovi Ph.D za odborné vedenie práce a pomoc pri písaní tohto článku.

Literatúra

[1] National Center for Biotechnology Information. Pubchem programmatic access. http://pubchemdocs.ncbi.nlm.nih.gov/programmatic-access$_RequestVolumeLimitations [cit. 2018-04-07].

[2] Julian R Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM (JACM)*, 23(1):31–42, 1976.

[3] Vincenzo Bonnici, Rosalba Giugno, Alfredo Pulvirenti, Dennis Shasha, and Alfredo Ferro. A subgraph isomorphism algorithm and its application to biochemical data. *BMC bioinformatics*, 14(7):S13, 2013.

[4] Luigi P Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. Performance evaluation of the VF graph matching algorithm. In *Image Analysis and Processing, 1999. Proceedings. International Conference on*, pages 1172–1177. IEEE, 1999.

[5] Luigi P Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. A (sub) graph isomorphism algorithm for matching large graphs. *IEEE transac-*

tions on pattern analysis and machine intelligence, 26(10):1367–1372, 2004.

[6] Daylight Chemical Information Systems. Fingerprints - screening and similarity, 2008. http://www.daylight.com/dayhtml/doc/theory/theory.finger.html [cit. 2018-04-01].

[7] Greg Landrum. Fingerprint-based substructure screening 1, 2013. http://rdkit.blogspot.cz/2013/11/fingerprint-based-substructure.html [cit. 2018-04-07].

[8] MDL Information Systems. Ctfile formats, 2003. http://www.daylight.com/meetings/mug05/Kappler/ctfile.pdf [cit. 2018-04-01].

[9] Peter Ertl and Bernhard Rohde. The molecule cloud-compact visualization of large collections of molecules. *Journal of cheminformatics*, 4(1):12, 2012.