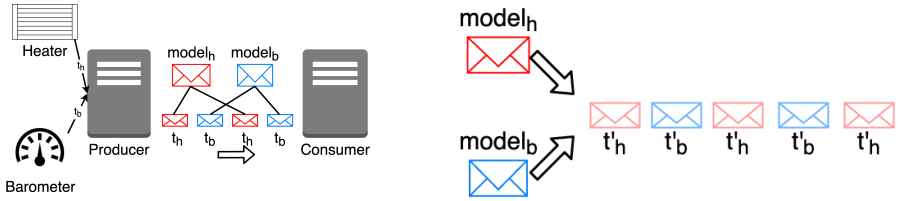
BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

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 Žeriar 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.*

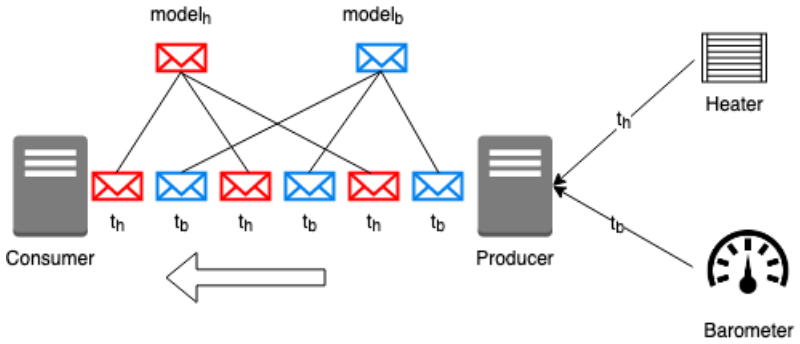


Figure 1. Example of a simple industrial network.

```
<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.

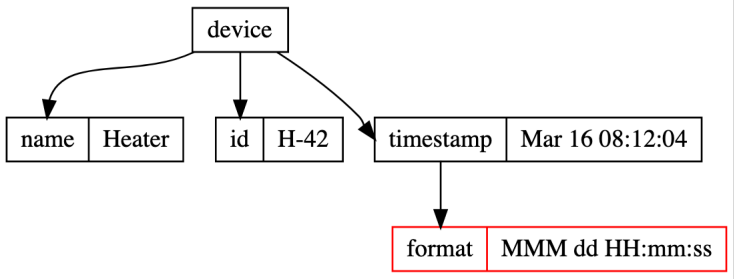


Figure 2. Tree-like representation of XML document from Listing 2.

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.

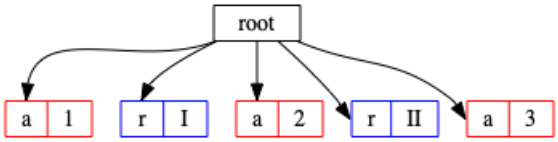


Figure 3. Before **group** operation.

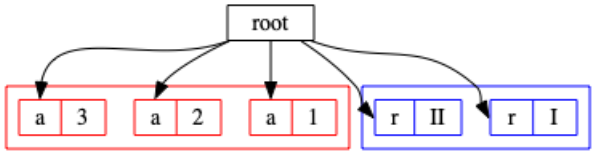


Figure 4. After **group** operation.

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

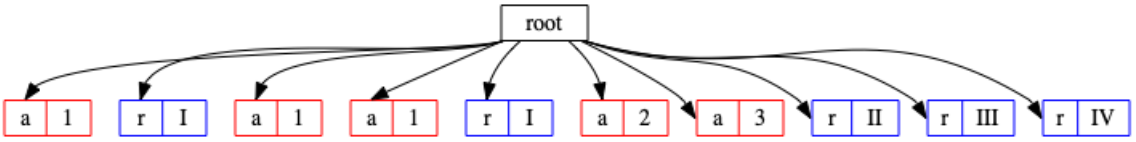


Figure 5. Before **reduce** operation.

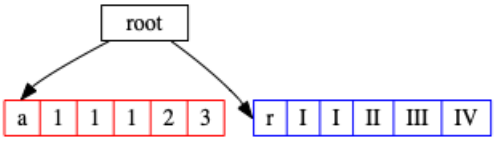


Figure 6. After **reduce** operation.

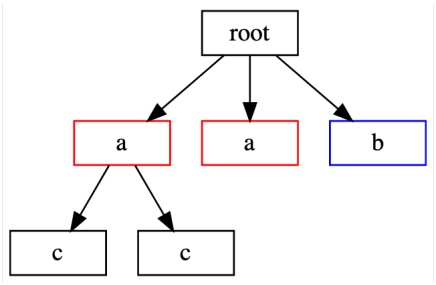


Figure 7. Left tree T_1.

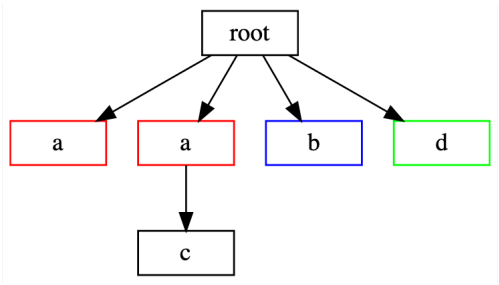


Figure 8. Right tree T_r

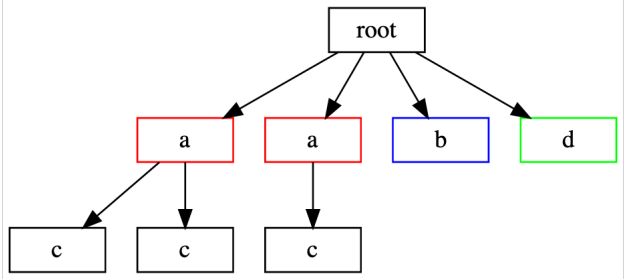


Figure 9. The merged trees T_l and T_r.

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.

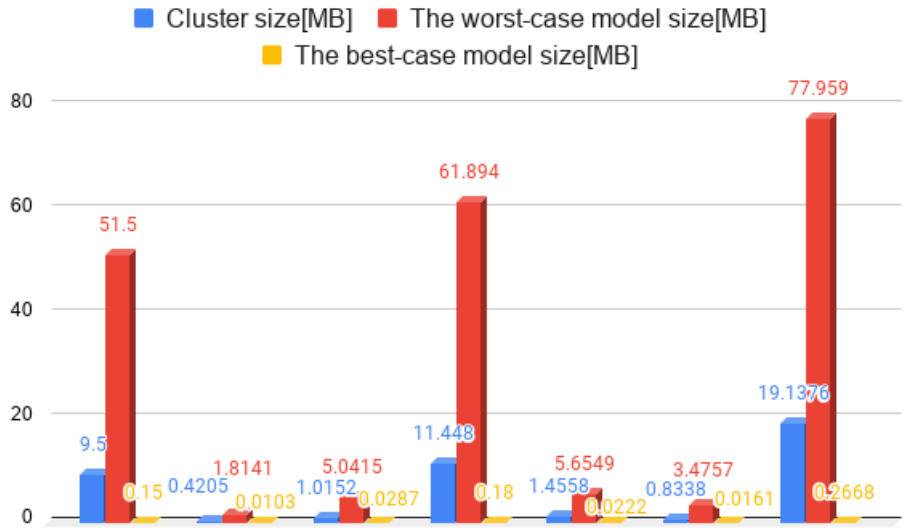


Figure 10. The comparison of the sizes of the original cluster, the worst-case model, and the best-case model.

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	$1 - \frac{13}{99} \approx 0.87$	$1 - \frac{54}{798} \approx 0.93$