Bachelor Thesis

Scaling RGCN Training through Graph Summarisation

Author: Alessandro Generale (2640447)

supervisor: Dr. Michael Cochez
2nd reader: Prof. Frank van Harmelen

A thesis submitted in fulfillment of the requirements for the VU Bachelor of Science degree in Computer Science

August 2, 2021

Abstract

Graph summarisation is a technique that enables to handle large Knowledge Graphs. However, the different methods that produce graph summaries are problem-specific and their use-case remain somewhat unspecified. This work attempts to bridge the gap between graph summarisation and novel machine learning algorithms that learn from Knowledge Graphs. Deep learning on graphs is an emerging topic where many models are proposed to perform tasks such as classification of nodes or graphs. In this research, we train an RGCN model on a graph summary to perform multi-label classification on nodes' types. The Attributes Summary and (k)-forward bisimulation summarisation techniques are tested on the *AIFB*, *MUTAG* and *AM* datasets. The model obtained when learning on the graph summary representation is then transferred to another RGCN model after which we continue training on the original graph. The current results look promising, but we can not yet conclude if the transfer learning model scales RGCN training. Additional experiments are required to determine the exact performance, especially on larger datasets.

1 Introduction

Knowledge Graphs emerged as an abstraction to represent and exploit massive amounts of data obtained over the internet to ease accessibility [14]. As a result, extensive collections of data stored in Knowledge Graphs are now publicly available, spurring the interest in investigating novel technologies that aim to structure and analyse such data. Despite all of the efforts invested in giving birth to these Knowledge Graphs, even the biggest ones, such as DBPedia and WikiData [3], remain incomplete. There is an evident trade-off between the quantity of data possessed and its adequate coverage (completeness) [6]. The more data is collected to create a Knowledge Graph, the more complex it becomes to organise it according to a schema and free of errors. Knowledge Graphs are negatively affected by missing information as the data needs to be explicitly stored. Predicting missing information in Knowledge Graphs is the main focus of statistical relational learning [28]. Previous literature has proposed graph embedding techniques to group similar nodes as points in the embedding space. Although graph embeddings offer a solution to missing information, such methods remain susceptible to a large quantity of data and the unknown structure of Knowledge Graphs, harming the scalability of applications [24]. New research has been published offering different techniques to produce summaries of Knowledge Graphs, a more compact and scalable KG that retains the original structure of the graph. The paper [26] proposed to produce a highly dense subset of nodes from the original graph. The work found in [13] creates a fused graph that embeds the topology of the original graph and in turn recursively coarsens into smaller graphs for a number of iterations. The methods showed to improve the classification accuracy and accelerate the graph embedding process. The following research will focus on the work of graph summarisation by training an RGCN on a version of the summarised graph. In particular, two summarisation methods are considered: Attributes Summary [9] and (k)-forward bisimulation [8]. Unfortunately, there exist a limited number of datasets equipped with a reasonable number of labelled nodes needed to test performance. Ultimately, progress in this area remains difficult to estimate [6]. In this research, the experiments are carried out performing entity classification on the type of nodes present in the Knowledge Graph. Therefore, we strip existing information from the original graph and use it to train and test our model using the remaining structure of the Knowledge Graph. Most of the nodes within a Knowledge Graph have zero-to-many 'rdf-type' relations. Such information is removed from the original graph, which allows to produce a large number of training and testing nodes. Large Knowledge Graphs tend to represent a higher level of detail. An entity node may have more than single type relation describing the different properties of a node. As a result, an arbitrary node's types form a hierarchical structure. The main intuition is to summarise the original graph by partitioning nodes considered equivalent under a summarisation relation. The reason why an RGCN was chosen as the machine learning model lies in its computation intensity,

as it produces a weight matrix for each relation in the Knowledge Graph [28]. A graph summary would require less computations as its size is ideally smaller. The size of a graph summary depends on the compression rate of the summarisation technique used. The setup of the following experiments consists in learning on the graph summary representation by performing multi-label classification on nodes' types. Multi-label classification is chosen because a summarisation method may place nodes with different types into the same partition. The parameters of the model obtained after training on the graph summary are transferred to another RGCN model that has the structure of the original graph. Additional training on the second model is performed. Finally, the accuracy is compared to the benchmark model proposed in the paper [28] using the same training and testing nodes splits. The intention behind this setup is that the model obtained by training on the graph summary improves the original model's performance early in the training.

1.1 Research Focus

Our research question focuses on the literature of graph summaries and to what extent provide valuable insights on the original graph structure. Therefore, the research question can be formulated as follows: *If we train an RGCN model on a graph summary, can we transfer the parameters of that model to an RGCN model for the original graph, and does that help getting good classification performance?*
Our primary focus lies in evaluating summarisation techniques to provide a scalable alternative to the original graph in the training phase. The framework proposed in this work offers an intuitive approach to measure the performance of graph summarisation techniques. The model chosen to investigate the performance of graph summaries is a Relational Graph Convolutional Network. The two main summarisation techniques that are investigated are the Attribute Summary [9] and the (k)-forward bisimulation [8].

1.2 Background - RGCN

Graph Convolutional Networks (GCNs) have spurred great interest in Machine Learning with graph entities. A GCN takes as input a graph and outputs embeddings generally representing nodes. GCNs use a message-passing algorithm which means that neighbouring information can influence the embedding representation of a node. However, a drawback of a GCN is that it treats all relations within the heterogeneous graph as the same. Therefore, the primary focus shifted from analysing simple undirected graphs to models that analyse multi-relational graphs, where each edge has a label and direction. Different research has proposed new kinds of GCNs able to deal with Knowledge Graphs. The COMPGCN [31] framework generalises different existing multi-relational GCN methods by learning a d-dimensional representation for each relation in the graph along with the node embeddings. Models often deal overparametrisation as the number of relations and node entities present in the graph increase. COMPGCN models relations as vectors, alleviating the issue of overparametrisation. An RGCN is defined as a convolution that performs message passing in the context of multi-relational graphs [29]. The RGCN model produces a unique projection matrix for each relation in the graph (see Figure 1) [28]. The component $W_r^{(l)}$ represents the specific weight matrices per relation. Therefore, each RGCN layer contains r weight matrices where r is the number of unique relations. The first summation term refers to all types of relations present in the graph. The second term of the summation takes into consideration the neighbours of a node under a specific relation. The $h_j^{(l)}$ term of the forward pass refers to the hidden representation of a node at that layer. The RGCN gives self-connection a special treatment when updating the representation of a node, storing its weight matrix for self-connections indicated by the term $W_0^{(l)}$.

$$h_i^{(l+1)} = \sigma\left(\sum_{r\in\mathcal{R}}\sum_{j\in\mathcal{N}_i^r}\frac{1}{c_{i,r}}W_r^{(l)}h_j^{(l)} + W_0^{(l)}h_i^{(l)}\right)$$

Figure 1: Forward pass for RGCN model [28].

The RGCN is another model that encounters the issue of overparametrisation. In the paper [28] basis decomposition was proposed where the number of weight matrices is to be reduced, and each relation learns a coefficient to scale the remaining matrices. With maximal decomposition, each of the relations has to learn r number of relations coefficients to scale the single projection matrix. Block diagonal decomposition is another technique proposed in [28] in which the projection matrices are stacked in a diagonal manner into a new larger matrix. The newly created matrix will have its majority of entries equal to 0. Both of these techniques are proposed to avoid overparametrising the model and therefore, overfit the data. Intuitively, the larger the entity graph becomes the larger the projection matrices per relation become. The first layer of an RGCN is generally the number of nodes by the number of outputs. Therefore the number of weight matrices of the size ($num\_nodes * num\_outputs$) is equal to the number of different relations in the original graph. As the size of the graph increases, the more computationally intensive the RGCN model becomes for the application. Relational Graph Convolutional Networks were shown in previous research to perform link prediction and entity classification tasks [2].

2 Graph Summarisation

A graph can hold a function similar to a schema in databases [25], understanding its underlying structure can bring valuable insights into the data. Relational databases are structured to keep track of the relationships of the individual entities but are less flexible when new types of data and relationships are added to the dataset. The structure of a graph allows for the simple addition of new entities and corresponding edges between them [14]. In this research, we define a Graph following Definition 1 proposed in the literature [9]. The set denoted with L consists of the edge and nodes labels. Each edge connects a pair of nodes, indicating a direct relationship between them, capturing symmetric relations.

Definition 1 (Graph) *(definition obtained from [9] page 15).*
Let V be a set of nodes and L a set of labels. The set of directed labelled edges is defined as $A \subseteq \{(x, \alpha, y) \mid (x, y) \in V^2, \alpha \in L\}$. A Graph G is a tuple $G = (V, A, l_V)$ where $l_V : V \Rightarrow L$ is a node-labelling function.

The data inside a Knowledge Graphs is serialised in the form of triples. A triple is composed of a source node s that has a direct relationship p to a destination node o, also referred to as object node. Knowledge Graphs have a similar structure to heterogeneous graphs, which contain different edge and node types. Figure 2 shows a small Knowledge Graph fragment showing different nodes connecting with different edge relations. The set of nodes $\{v_1, v_2, v_3, v_4, v_5\}$ are entity nodes which posses a unique IRI (Internationalised Resource Identifier). In this basic example, the nodes are given pseudo IRIs. The set of nodes $\{"Book1", "Book2"\}$ are literal nodes, normally used for values such a strings, numbers or dates. An example of triple within this graph is the following: $(v_1, title, "Book1")$. We define an attribute to be the label specific to an edge. In this case the attribute relating node v_1 and literal node "$Book1$" is title. Graph summarisation's intended first use is to inspect the underlying structure of a graph and produce a smaller representation of the original graph [16]. Intuitively, the size of the original graph influences the size of the produced summary. A graph containing a vast

amount of information would produce a large summary. However, this highly depends on the internal structure of the graph and the summarisation relation used. A Graph that has many nodes that share similar characteristics is likely to be summarised into a compact graph representation.

There are different challenges with regards to graph summarisation which can be divided into three different dimensions [9]:

- size of the graph;
- size of the summary;
- the impact of the graph's heterogeneity on the summary.

Previously, we briefly touched upon the relationship between the size of the original graph and its summary. When we refer to the size of a graph we generally define it as the number of edges $|A|$ that the graph is composed of [9]. A reasonable graph summary should be smaller in size than the original graph, since it is generally computed for performance reasons. The more the edges the graph has, the more computational power and memory is required within the application's settings. Due to the increasing popularity of linked data, techniques to concisely represent Knowledge Graphs are essential for efficient storage and analysis [3].

A graph is considered heterogeneous when not consistent with regards to its labelling and structure [9]. The more heterogeneous a graph, the more complex it becomes to create a concise and precise summary. Concise refers to the size of the graph, whereas precise concerns the preservation of the entity structure. In turn, there exist two families of graph summaries: approximate graph summaries and precise graph summaries [9].

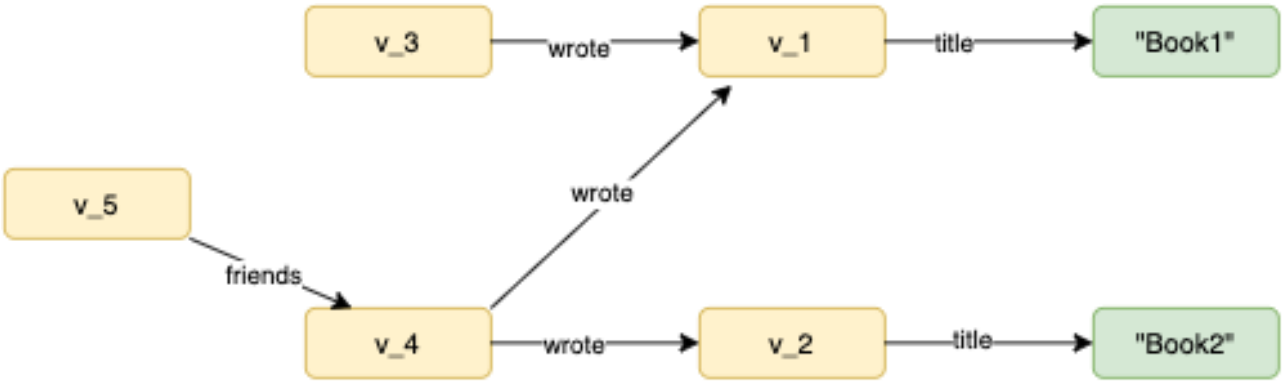


Figure 2: Fragment of a Knowledge Graph. The entities are the nodes v_1 to v_5 and the edges have attributes e.g. 'wrote', 'title' and 'friends'. The green nodes are Literals.

A fourth challenge that can be mentioned with regards to graph summarisation is the role of the literal nodes. In this research, the summaries are computed taking into consideration literal nodes but their content is abstracted away. However, additional summaries were produced without considering the existing literal nodes. As a result, the compression rate increases because many nodes within the graph are ignored when computing the summary. For instance, looking back at Figure 1, the triples connecting to literal nodes "Book1" and "Book2" are stripped away and then the summary is computed. In the following sections, we will discuss several methods concerning Approximate Graph and Precise Graph summaries.

2.1 Approximate Graph Summaries

Approximate graph summaries exploit the different features of the data which form the local information of a node. For instance, a node has incoming and outgoing edges with different labels also

called attributes [9]. The direction of an edge has semantics within the structure of the graph. A node has a one-to-many 'rdf:type' relations to an object node defining a specific property of a node. The local information of a node described above is combined to produce different partitions of the original graph. The combination of information defines the summarisation relation. The partitions, which correspond to summary nodes, are produced based on the summarisation relation. Nodes belonging to the same partition are aggregated and mapped accordingly. This section of the paper discusses several approximate graph summary methods such as Attributes Summary, IO Summary and Incoming Attributes summary.

2.1.1 Attributes Summary

The Attributes Summary has been defined in previous literature [9][8] as an approximate graph summary. In the context of a Knowledge Graph, each entity node has zero-to-many relations with different attributes. The set of the attribute per nodes is computed and the entities that have equivalent attribute sets are partitioned together. Each partition is then considered a unique summary node. Each of the original entity nodes is mapped to a partition, hence a summary node, if it has at least one outgoing edge and is not a literal node. It is important to notice the summary method being discussed aggregates a node with few edges into the same partition as another node with many more edges if they have equivalent attribute sets. Therefore, the number of relations a node has is not considered. The nodes that do not have any outgoing edges, often the literal nodes in Knowledge Graphs, are aggregated to the same partition or summary node. The literal nodes information when performing the following summarisation technique is abstracted away. Figure 3 shows the Attributes Summary result on the Knowledge Graph fragment presented in Figure 2.

Definition 2 (Summarisation Relation R_a) *(definition obtained from [9] page 48).*
Let $G = (V, A, l_V)$ and $S_a = (W_a, B_a, l_{W_a})$ be two graphs. G refers to the original graph. S_a refers to the summary graph. The set of nodes W_a contains as many elements as the powerset of attributes in the graph. Therefore, S_a corresponds to the Attribute Summary of G according to the summarisation relation $R_a \subseteq V \times W_a$ defined as follows:

$$R_a = \{(u, x) \in V * V_a \mid attributes(u) = attributes(x)\}$$

Definition 2 describes the summarisation relation R_a. Two nodes are placed in the same partition if their attributes set are equivalent. As shown in 3, the nodes $\{v_1, v_2\}$ belong to the same partition as they share the same attribute set $\{title\}$. A remark to make is that the summarisation relation described above maps nodes without any outgoing edges to undefined sumnodes. As mentioned above, most of the time a sumnode represents literal nodes as they do not possess any outgoing edges. In this case, as shown in Figure 3, the string literals e.g. 'Book1' and 'Book2' map to the same sumnode defined with the $\emptyset$ symbol. The *attribute* function computes the node's mapping to a summary node according to the summary relation R_a (see Table 1). The input to the function is a node's original IRI which and the output is the summary node's IRI based on the set of attributes.

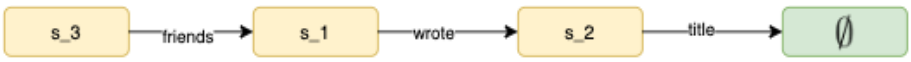


Figure 3: Attributes Summary graph with generated mapping from original node to summary node. The green node labelled by the empty set symbol abstracts the content of Literal nodes.

V	W_a
v_1, v_2	s_2
v_3, v_4	s_1
v_5	s_3

Table 1: The table indicates the mapping produced by the Attributes Summary from original nodes $\in V$ to summary nodes $\in W_a$ under $R_a(V, W_a)$.

Appendix A and Appendix B contain the SPARQL query from [9] needed to perform the Attributes Summary. More specifically, the query found in Appendix A computes the mapping from summary nodes to original nodes. In Appendix B, the query generates the whole graph summary. It is important to notice that once the mapping is created, iterating over all original triples and applying the mapping function produces the summary graph in linear time. No duplicates triples are added as we store the triples into a set, and by definition a set does not contain duplicate items.

2.1.2 IO Summary

The Input-Output summary is similar to the Attributes Summary. In fact, the structure of Definition 3 is almost identical. However, in this case, for each node in the graph it creates two sets defined as incoming attributes set and outgoing attributes set. All the nodes that have equivalent incoming and outgoing attributes sets are inserted in the same partition.

Definition 3 (Summarisation Relation R_{io}) *(definition obtained from [9] page 48-49).*
Let $G = (V, A, l_V)$ and $S_{io} = (W_{io}, B_{io}, l_{W_{io}})$ be two graphs. G refers to the original graph. S_{io} refers to the summary graph. The set of nodes W_{io} contains as many elements as the powerset of attributes in the graph. Therefore, S_a corresponds to the Incoming Outgoing Attribute Summary of G according to the summarisation relation $R_{io} \subseteq V \times W_{io}$ defined as follows:

$$R_{io} = \{(u, x) \in V * V_{io} \mid attributes(u) = attributes(x) \wedge attributes^{-1}(u) = attributes^{-1}(x)\}$$

The function *attribute* and $attribute^{-1}$ compute the outgoing edges partition and the incoming edges partition respectively. The negative exponent indicates an inverse operation. Therefore, instead of computing the attributes sets of outgoing edges, it computes the attributes sets of incoming edges. A node u that has a equivalent incoming and outgoing attributes sets to a node x shows that node u and x belong to the partition under the summarisation relation R_{io}. When applying the following summarisation relation to the KG fragment shown in Figure 2, the result is the graph summary and mapping found in Appendix B.

2.1.3 Incoming Attributes Summary

The Incoming Attributes Summary is the inverse operation of the Attributes Summary. The following partitions each node based on the incoming attributes set. Two nodes that have equivalent incoming attributes sets are inserted into the same partition. Definition 4 has a similar structure to the other previously mentioned summarisation techniques.

Definition 4 (Summarisation Relation R_{ia}) *(definition obtained from [9] page 50).*
Let $G = (V, A, l_V)$ and $S_{ia} = (W_{ia}, B_{ia}, l_{W_{ia}})$ be two graphs. G refers to the original graph. S_{ia} refers to the summary graph. The set of nodes W_{ia} contains as many elements as the powerset of attributes in the graph. Therefore, S_{ia} corresponds to the Incoming Attributes Summary of G according to the summarisation relation $R_{ia} \subseteq V \times W_a$ defined as follows:

$$R_{ia} = \{(u, x) \in V * V_{ia} \mid attributes^{-1}(u) = attributes^{-1}(x)\}$$

The function $attribute^{-1}$ takes as input the original node and outputs the corresponding summary node based on the partitions formed on the incoming edges of a node. Therefore, a node u with an equivalent incoming edges partition to node x shows that nodes u and x belong to the same summarisation relation R_{ia}. When computing the summary with the following summarisation relation on the KG in Figure 1, the result is the graph summary listed in Appendix C along with the mapping.

2.1.4 Implementation

The work carried out in [9] proposes a SPARQL query to produce a graph summary based on 'rdf:type' relations sets. This query is adapted to perform an Attribute Summary as can be seen in Appendix A. The query shown in Listing 2 produces the Attributes summary of a graph. Lines 8-12 and 18-22 compute the attributes sets of the source node and the object node respectively as per the summarisation relation R_a. Another summarisation relation can be used by changing the lines of the query mentioned above. For instance, the summarisation relation R_{ia} can be inserted considering also 'rdf:type' relations. This is an idea for future research to explore more complex summarisation techniques. We formulated a similar query (see Listing 1), which was not proposed in the paper [9] to retrieve the mapping from summary nodes to original nodes. The two SPARQL queries described output two '.n3' files which are then parsed using the 'rdflib 6.0.0' [1] Python library.

2.2 Precise Graph Summaries

Partitioning nodes based on their local schema is somewhat limited. New graph summarisation techniques compute the schema of vertices considering neighbouring schemas over multiple hops [17][10]. Similarly, the notion of aggregating nodes based on local and neighbouring information is shared in modern machine learning models that deal with graph entities. The general purpose of a graph summary is to mirror the structure of the original graph whilst being consistently smaller in size. Precise graph summaries are considered a stricter method because the summarisation relation is more complex as it partitions vertices based on equivalent substructures of the original graph. In order for two nodes to belong to the same partition, neighbouring nodes must also be related under the same relation. A graph summary is said to be precise when indiscernible from the original graph [9]. A summary graph may not have paths that are present in the entity graph but the overall structure of the entity graph is preserved due to graph homorphism [9]. Additionally, the structure of the original graph is kept in the summary but the inverse may not hold true. From the mapping in Table 1, it is possible to reconstruct a version of the original graph, however it would produce a different graph.

Definition 5 (Precise Graph Summary) *(definition obtained from [9] page 41).*
Let $G = (V, E, R)$ and $S = (W, B, L_W)$ be two graphs. The graph S is the summary of the graph G according to a summarisation relation $R \subseteq V \times W$. Let $p = (x_1, \alpha_1, x_2) \in B \wedge \ ... \wedge \ (x_n, \alpha_n, x_{n+1}) \in B$ be a path in the summary graph S where $(x_1, ..., x_{n+1}) \in W^{n+1}$. Let the set $\{u_1, ..., u_{n+1}\}$ be a summary path instance $u_i \in V$. A summary is therefore called precise when each instance of a summary path in p forms a path in the original entity graph with regards to the edges in p.

$$\forall \in [1, n] \ \exists \ (u_i, \alpha_i, u_{n+1}) \in E$$

Definition 5 states that a graph summary is fully precise if all paths that can be formed in the summary graph can be found in the original graph. The original graph may hold paths that are not present in the graph summary. Looking at the example of graph summary produced in Figure 3, there exist paths which are not present in the original graph, such as the paths (v_5, v_3, v_2). Such path can be

[1] **rdflib 6.0.0:** `https://rdflib.readthedocs.io/en/stable/`

formed in the summary graph as the nodes $\{v_5, v_3, v_2\}$ are partitioned by the nodes $\{s_3, s_1, s_2\}$ which form the previously mentioned path. With the same subset of summary nodes, it is possible to create paths that are contained in the original graph, for instance, the path (v_5, v_4, v_2).

2.2.1 Bisimulation

The intuition behind bisimulation, in the context of graphs, is the interpretation of the graph data as transition systems to discover structurally equivalent parts [8][27]. Two nodes' states are bisimilar if their states change following equivalent edge relations [8]. Bisimulation is a binary relation that relates two arbitrary nodes in a Knowledge Graph when itself and its inverse are simulations [9]. For instance, consider two nodes u and v, a simulation relation states that an edge with type α departing from u and pointing to an arbitrary node x implies that there exists an edge with type α from v pointing to an arbitrary node y such that x and y are simulations. It is important to notice that two nodes are bisimilar if they share the same outgoing paths. The notion of bisimulation is stricter as it is a symmetric equivalence relation, ensuring that each node can substitute one another.

2.2.2 (k)-forward Bisimulation

(k)-forward bisimulation is an example of the many graph summarisation techniques which aggregates nodes based on neighbors' schema over multiple hops [18][21][23][30]. It has also been defined as a stratified bisimulation in [8], a summarisation relation that is restricted to a maximum path length of k-edges. Other summarisation techniques that propose bisimulation over k hops also consider the direction of the edges within the knlowedge graph [27]. These are referred to as *forward*, *backward* and *forward/backward* [19] bisimulation. In this research, we will be experimenting with the *FLUID* framework [7] that allows us to perform a *forward* (k)-bisimulation summary. The above mentioned framework already outputs a mapping from original nodes to summary nodes. Figure 6a represents a small fragment of a Knowledge Graph. A forward (k)-bisimulation with chaining parameter $k = 3$ is performed on this small entity graph to illustrate the inner workings of the framework used. The resulting summary graph with alongside the one-to-one mapping from original node to summary node is shown in Figure 6b.

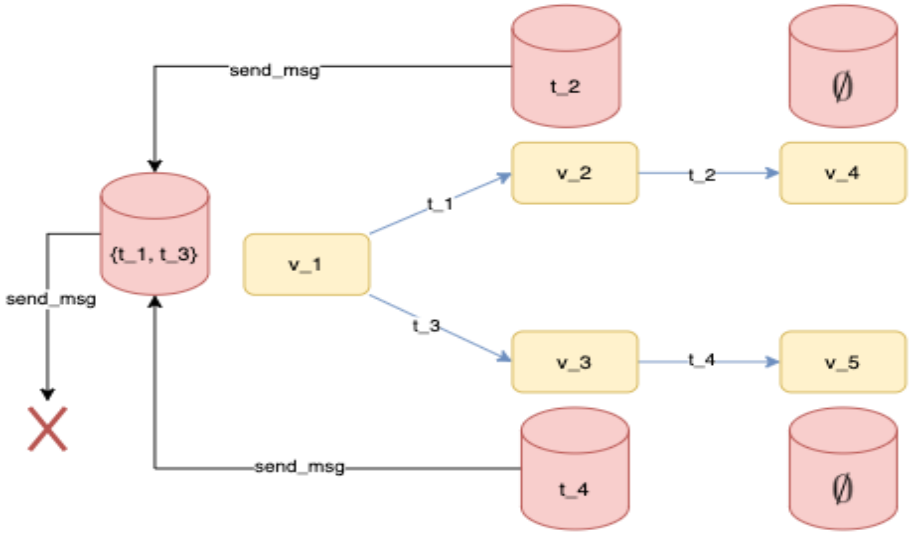


Figure 4: **Step 1:** Send local schema through incoming edges.

The intuition behind the implementation of the forward (k)-bisimulation (see Figure 4) consists in performing a message passing algorithm over multiple hops. At the first iteration, the message created is the local schema of each node. This newly created message is sent back through all incoming edges. In this way, each node gathers information about the schema of neighbouring entities. As described in the documentation of the framework [7], the message passing algorithm has to follow the inverse direction of the edges. In this case, the node v_2 has an outgoing edge labelled t_2 to node v_4. Therefore, node v_2 will send a message to node v_1 with the information t_2. Intuitively, Figures 4 and 5 are only showing the the top sub-graph of the Knowledge Graph, but similar message passing will apply for the nodes v_{96} to v_{100}. The newly arrived messages are aggregated and used to update the state of a node. As shown in Figure 5, only the nodes that have received a message and updated their states will send a message again in the next iteration through all incoming edges. Therefore, this step of the message passing algorithm aggregates the incoming messages per node, if any, and sends out the newly updated state up to k hops. Generally, the value of k is suggested to be around 2 or 3. In the framework [7] used in this research paper the chaining parameter k is set to 3.

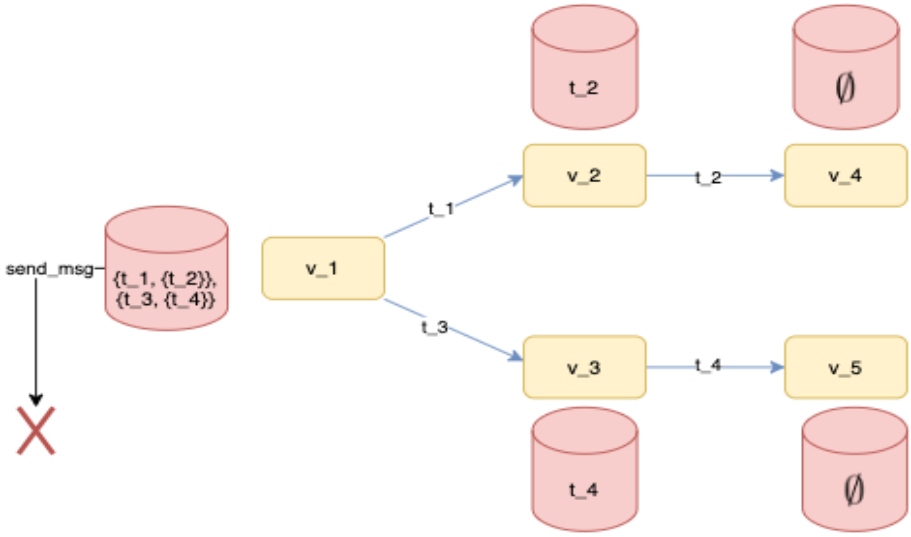
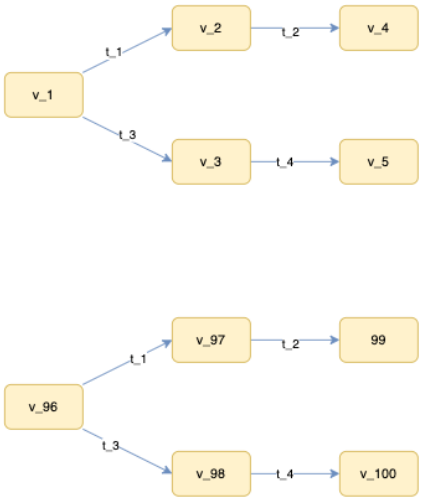
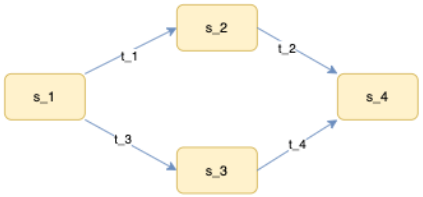


Figure 5: **Step 2:** Aggregate incoming messages and update state. If state has been updated, the message is sent again through incoming edges up to k iterations.

Looking at the diagram in Figure 4, at the first iteration of the algorithm, the message $\{t_2\}$ is sent from the node $\{v_2\}$. Intuitively, the message $\{t_1, t_3\}$ is sent from the node v_1. However, node v_1 does not possess any incoming edges and is unable to propagate its message to neighbours. At node v_1 the messages $\{t_2\}$ and $\{t_4\}$ are aggregated according the structure of the path. The next iteration of the algorithm sends the updated states as messages. The node v_1 has updated its state to $\{\{t_1, \{t_2\}\}, \{t_3, \{t_4\}\}\}$, but cannot send this message to any neighbours (see Figure 5). Therefore, the nodes in the graph at the second iteration are not able to update their states further as no messages are propagated further. The algorithm terminates. The nodes are partitioned according to their internal state representations. The bottom sub-graph is structurally equivalent to the upper sub-graph described above. Therefore, after k iterations, its nodes updated their states in the same way. Nodes (v_1, v_{96}) are mapped to summary node s_1, (v_2, v_{97}) to s_2, (v_3, v_{98}) to s_3. All nodes without outgoing edges are mapped to node s_4. Following this mapping, the newly formed graph summary is shown in Figure 6b.



(a) Fraction of a Knowledge Graph



(b) (k)-bisimulation Graph Summary

Figure 6: Figure 6a shows the original KG fragment. Figure 6b shows (k)-bisimulation graph summary produced with a chaining parameter $k = 3$.

3 Methods

Similarly to previous research [6][28], the experiments exploit the data found in commonly used datasets: AIFB [5], MUTAG [12] and AM [11]. More details on the datasets can be found in Appendix D. The task that will be performed is multi-label classfication on the types of nodes. Therefore, the RGCN is making use of a summarised version of the original entity graph and inferring the 'rdf:type' relations for each of the original nodes. We believe that the summarised version of the graph retains the main structure of the entity graph, helping to generalise the complex hierarchy of types in different nodes. From the one-to-one mapping generated by the summarisation method, it is possible to transfer the parameters relating the summarised nodes to the original nodes. We believe the that the training phase will require fewer iterations to converge to an optimal solution. To provide support to such an argument, a third model is deployed. The third model acts as a benchmark to compare performance. All RGCN models follow the proposed architecture found in [28] and developed by [15]. Differently, the models proposed in this framework use *binary cross-entropy* loss and apply a sigmoid activation function on the output to predict multiple labels. An issue that was discussed in previous literature [6] was the absence of labelled nodes. In the Knowledge Graphs generated from the previously mentioned datasets, only a small subset of nodes is split for training and testing. This is an evident problem when it comes to testing the actual performance. A design decision aimed to counter the issue of few labelled nodes is proposed by capturing a specific relation that is present in most nodes and to use its value as a label. An entity node in a Knowledge Graph has a one-to-many 'rdf:type' relations. The edge indicating such a relation is indicated with 'rdf:type' and its object defines the actual type of a node. Intuitively, a node can belong to multiple types depending on the level of details within the Knowledge Graph. Therefore, the nodes' target values are constructed based on the 'rdf:type' relations. This is explained in more detail in Section 3.2. The code base for the framework is written in Python 3.9.0. The implementation of the RGCN is taken from Pytorch Geometric [15]. In order to run longer tasks, with relatively large datasets, the DAS5 system [4] is used.

3.1 Data Pre-Processing & Summary Generation

The first step of the experiment is to create a map of original nodes to their types. In the previous literature [32], the labels of the nodes are obtained from properties that are decided beforehand. The

nodes may belong to a single or multiple classes. The class label is not necessarily reflected or present in the graph. It is a property that has been assigned prior or after permutation of the data. However, as mentioned in previous research, there are very few datasets equipped with a reasonable number of labelled nodes [6]. In this experiment, a greater number of labelled nodes that can be split into training and testing data that is obtained by considering the respective 'rdf:type' relations, and removing these triples from the original graph. Other properties of nodes within the Knowledge Graph can be used as instances to train the model. An example is considering existing edges for link prediction tasks [29].

The graph that was stripped of the triples is fed to the two summarisation frameworks to compute the Attributes Summary (Section 2.1.1) and the forward (k)-bisimulation (Section 2.2.2) with the chaining parameter $k = 3$. The summarisation techniques provide two *'.n3'* files, one with the set of triples of the summary graph and the second one with the triple mapping original node to summary node, which contain all the relevant information needed to proceed further with the experiment. In the framework of the research, there is a directory with all the needed files which contain both the mapping and the summary file. In order to produce the summary graph, we iterate through all original triples and apply the mapping function to each source and object node. The mapping function yields the corresponding summary nodes which form a triple with the predicate relation connecting the original source and object node. The new triple is added to the set of triples composing the graph summary. As the data structure containing the triples is a set, no duplicate triples will be added. Therefore, from the mapping and the original triples it is possible to generate the graph summary with or without literal nodes. We reconstruct the graph summary without literals. Prior to computing the summaries, the Knowledge Graph is also stripped of triples with edges containing the Web Ontology Language, a Semantic Web language designed to represent rich and complex knowledge about entities, as suggested in [9]. In order to process and clean the Knowledge Graphs, we used *GraphDB*[2] [1], a Knowledge Graph platform where you can perform SPARQL queries and visualise graphs. *GraphDB* is also used to perform the SPARQL query for the Attributes Summary.

3.2 Summary Nodes Labels

A summarisation technique may map multiple nodes with different 'rdf:type' values to the same partition or summary node. As mentioned in the above section, the labels of each node is obtained from a specific 'rdf:type' relation. The labels of the summary nodes is calculated as following: count the occurrence of each type in a partition and divide it by the number of entities in the same partition.

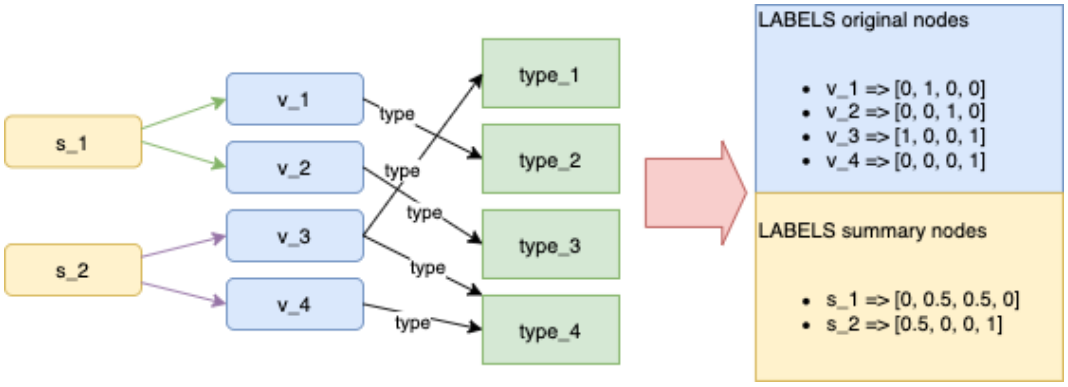


Figure 7: Diagram showing the original and summary nodes labelling process.

Figure 7 shows the general labelling of the summary and original nodes. The nodes s_1 and s_2 are two summary nodes that represents original nodes $\{v_1, v_2\}$ and $\{v_3, v_4\}$ respectively. The node v_1 has a 'rdf:type' relation and is labelled $type_2$. Similarly, v_3 has two 'rdf:type' relations and is therefore

[2]**GraphDb:** `https://graphdb.ontotext.com/`

labelled $type_1$ and $type_4$. From the partitioning of the summary nodes we can obtain the labels for the summary nodes. The node s_2 represents nodes $\{v_3, v_4\}$ and its labelling is 0.5 for $type_1$ because only one node has that relation. The $type_4$ is labelled 1.0 because both nodes point to $type_4$. The machine learning model that learns on the summary representation uses the labelling produced by the summary partitions. The transfer learning model and benchmark model trains on binary values to predict the types of nodes.

3.3 Machine Learning Model: RGCN

To perform the experiments, we set up three different RGCN models, a model that learns on the graph summary, a second model with parameters initialised from the previous model and a final model that follows a normal initialisation to act as a benchmark. The initialisation of the second model is performed by transferring the embedded parameters from the model that learns on the graph summary using the mapping from original nodes to summary nodes. The implementation by [15] provide us with a working RGCN model to perform the experiments. As mentioned earlier, the model has been slightly modified in order to fit the purpose of the task: multi-label classification. A *binary cross-entropy loss* function [22] is used to compute the loss during training. The assumption is that an element that belongs to one class does not influence the probability of the same element belonging to another class. The following loss function requires target values to be between 0 and 1. The target values of the summary and original nodes are created as described previously in Section 3.2. The labelling process gives high values to the labels that belong to most of the original nodes inside the partition. The same loss function is used on the second model and the benchmark model. On the output of all models, a sigmoid activation function is applied to yield a value between 0 and 1 corresponding to the probabilities of the labels of type relations. The output output of the model is then rounded to the nearest integer to match the original nodes' labels.

A parameter that needs to be taken into consideration is the number of hidden units between the two RGCN convolutions. In previous literature [20][28], the following is suggested to be between 16 and 32. In this research, this value is set to 16 and is equal across all three models. In [20] a regulariser is omitted and we do the same. An Adam optimiser [22] with a learning rate of $1.0 * 10^{-2}$ and a weight decay of $5.0 * 10^{-4}$ is applied as suggested in [28]. The number of basis is another parameter that needs to be taken into consideration. In our settings, we do not use basis decomposition. If used, number of basis has to be equivalent across all models in order to compare them. The number of basis used depends on the dataset as suggested in [28].

The model which learns on the summary graph is trained first for a total of 51 epochs. The parameters of this model are transferred to the second RGCN through the mapping produced by the summarisation technique. An additional training parameter can be set which consists of freezing the layers of one or more layers so that the weights do not update during training. This is suggested to be done on the model to which the weights have been transferred to run a faster training. If the first convolutional layer of the transfer learning model is frozen, the embedded parameters that were transferred from the graph summary are not updated in the backward pass. All of the general parameters of the RGCNs trained in this research are specified in detail in Appendix E. The entire labelled instances are split 80% for training and 20% for testing for singular runs.

4 Analysis of the Results

The AIFB, MUTAG and AM datasets were summarised with the Attributes Summary and (k)-forward bisimulation techniques. The resulting graph summary is fed as input to an RGCN model. The embedded parameters of this RGCN are transferred to another model which has the structure of the original graph. Additional training on the original graph is performed and the accuracy is tested at each epoch. The idea behind the transfer learning consists in using the knowledge obtained from the

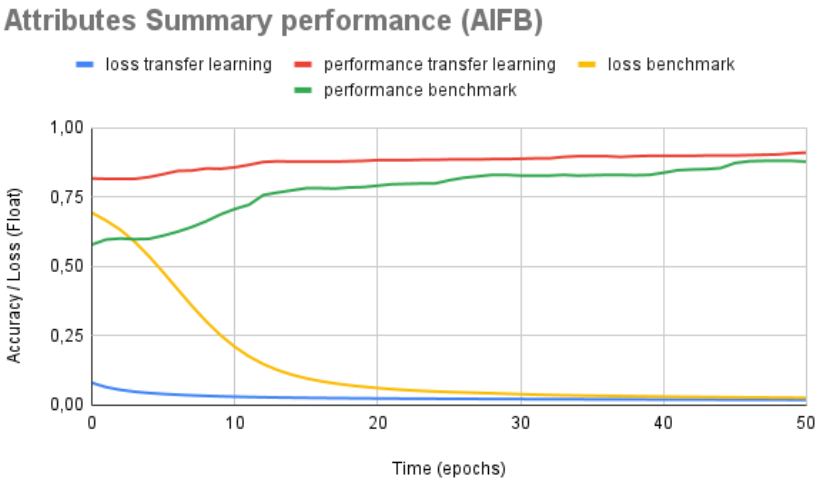


Figure 8: Graph showing the performance of the Attributes Summary performed on the dataset AIFB. The transfer learning model (red line) starts predicting at a higher accuracy than the benchmark (green line) from early iterations and converges to a better optimal solution.

graph summary to solve a task directly on the original graph. In Figure 8, a plot of the performance of the Attributes Summary graph used to train the RGCN is shown. The red line represents the model to which the embedded parameters of the summary model are transferred to. The green line represents the original model, with weights that are randomly initialised. The transfer learning model (red line) performs reasonably well from the first few iterations, with an accuracy of 81.7%. Instead, the original model (green line), shows the opposite behaviour. The accuracy starts at a lower accuracy of 57.9%. The performance of both models increases until it converges with the transfer learning model at 91.06%. Whereas the benchmark converges at a lower accuracy of 87.82%. The dataset *AIFB* contains 8285 entities, of which type information has been stripped. The Attributes Summary reduces the size in total number of edges by 63.7% on the *AIFB* Knowledge Graph. In this case, it appears that the number of iterations needed for the transfer learning model predict with a good accuracy has decreased. Additionally, we may observe that the end of training iterations, the transfer learning model seems to predict at a higher accuracy. This difference shows promising results, however we would like to observe a similar behaviour in the training on larger Knowledge Graphs.

Figure 8 shows the performance of the transfer learning model against the benchmark for a single run of the dataset. Table 2 shows the average starting and converging accuracy. The average is calculated by performing (k)-fold cross validation with k set to 5. It is possible to see that on average, on the *AIFB* dataset, the model that learned on the Attributes Summary starts predicting with 82.52% accuracy from iteration zero. The resulting accuracy on the test set is 92.54%. The model that learned on the (k)-forward bisimulation summary starts predicting with a lower accuracy of 77.53% and converges to a final accuracy of 91.30%. In this case, both models seem to perform better than the benchmark.

The *MUTAG* dataset provided much worse results (see Appendix G for single runs). There could be several reasons explaining the poor performance. First of all, both summarisation techniques provided very concise summaries of the dataset. This means that the model learned on tiny small versions of the original graph. The compression rates are 93.1% and 85.4% for the Attributes Summary and (k)-forward bisimulation respectively. In addition to this, 113 different labels were found when producing the training and testing nodes in the data pre-processing step. The performance altogether stayed low throughout the experiment runs. An explanation to this poor performance can be explained by the large number of labels increasing the task's complexity. In previous research, node classification

		Attributes Summary		(3)-forward bisimulation		Benchmark	
		Start	End	Start	End	Start	End
Datasets	AIFB	82.52 ± 3.93	92.54 ± 2.42	77.53 ± 4.72	91.30 ± 2.03	58.03 ± 14.15	87.98 ± 2.13
	MUTAG	0.21 ± 0.41	35.07 ± 9.70	15.94 ± 13.47	36.56 ± 8.56	24.77 ± 16.33	28.77 ± 2.00
	AM	62.36 ± 3.61	80.14 ± 2.17	61.75 ± 3.83	72.63 ± 5.31	13.90 ± 11.02	78.63 ± 3.21

Table 2: Results of the experiments with (k)-fold cross validation with k set to 5. Note that (k)-fold cross validation was not performed for the AM dataset due to computational resources restrictions, only two runs are reported for each model.

on the *MUTAG* dataset was done considering as few as 2 classes [29].

Finally, the *AM* dataset, the largest Knowledge Graph used in this experiment, showed somewhat promising results (see Appendix H for single runs). The knowledge obtained from the Attributes Summary resulted in an initial prediction of 62.36% and a convergence point of 80.14%. Both values seem to remain consistently higher than the benchmark. The (k)-forward bisimulation did not perform as expected on the larger dataset. Its initial prediction starts at 61.75% converging to 72.63%, a lower final accuracy on the test set compared to both the Attributes Summary and the Benchmark. This is the only case observed in all experiments, where the summarisation technique is consistently stuck at a lower local minimum than the benchmark. It is important to notice that due to the size of the *AM* dataset, only two training runs of each summarisation technique were measured and reported due to computational resource restrictions.

As mentioned at the beginning of Section 2, the graph summaries were produced considering the relations with the literal nodes. However, similar experiments were conducted without taking into consideration the literal nodes when computing the summary. The results for single runs on the *AIFB* and *MUTAG* datasets can be found in Appendix I. However, the results did not provide useful insights or improvements in the performance.

5 Conclusion and Future Work

We have introduced a possible approach to use graph summaries in the context of machine learning with Knowledge Graphs. The results of the experiments were to some extent promising for the *AIFB* and *AM* datasets. We were not able to achieve good results for the *MUTAG* dataset throughout our experiments. In both cases, it appears that the transfer learning model performs slightly better than the benchmark. This behaviour can be motivated by the attempt to transfer the parameters learned from the graph summary summary representation, resulting in a jump-start on the learning process. This helps support the importance and relevancy of graph summarisation methods, which seem to provide smaller graph representations to scale down and reduce the computational overhead involved with novel machine learning models dealing with large Knowledge Graphs. However, due to the complexity of the classification task, the variation in the accuracy remains large. The summarisation relation (k)-forward bisimulation and Attributes Summary showed similarities in the training behaviour for the *AIFB* and *AM* datasets. One clear limitation of this experiment set up consists in the abstract task performed. Multi-label classification is a more trivial task compared to single-label classification performed in previous research, and is highly dependent on the dataset's characteristics. For instance, in the *MUTAG* dataset, the model had to predict out of 113 total labels. Due to the graph's low heterogeneity, both summary methods produced a very dense graph summary. Both of these factors did not aid the performance of our model.

For future research, when evaluating the RGCN model, we should have first considered experimenting with a task in which the performance has been recorded in previous research. This would allow the possibility to replicate the pre-tuned RGCN and obtain performance metrics similar to ones al-

ready documented. In this case, the difference in performance would be easier to gauge between the transfer learning model and original model. Another aspect to keep in mind for future research is the hyperparameter k for the (k)-forward bisimulation. In this experiment, the paramter k was kept to a fixed value of 3. However, it would be interesting to compare it to the number of layers in an RGCN to examine the relationship between the machine learning model and the summary. (k)-forward bisimulation is a technique that partitions nodes based on neighboring information. To some extent, the RGCN layers embed nodes as points in embedding space also considering neighboring information. In addition to this, the model that the learns on the graph summary is solely learning on the set nodes that were mapped. A large number of nodes is discarded because such entities were not mapped when producing the graph summary. It would be interesting to perform an embedding of the literal nodes when computing the different partitions of the graph summary. More experiments can be performed with this framework using more complex summarisation techniques and different datasets. Altogether, this work provided a framework that can make use of graph summaries to predict nodes' labels and showed relatively promising results when training an RGCN on the graph summary. Overall, we cannot yet fully conclude that the summarisation techniques explored provide a good method to scale RGCN training. Early reasonable performance has been observed and reported for the *AIFB* and *AM* datasets for both summarisation methods. However, due to the large variation in accuracy it still remains difficult to determine a relevant difference in the training.

References

[1] Graphdb. The Best RDF Database for Knowledge Graphs. `https://graphdb.ontotext.com/`. Accessed: 2021-07-19.

[2] Ralph Abboud and İsmail İlkan Ceylan. Node Classification Meets Link Prediction on Knowledge Graphs, 2021.

[3] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. DBpedia: A Nucleus for a Web of Open Data. In Karl Aberer, Key-Sun Choi, Natasha Noy, Dean Allemang, Kyung-Il Lee, Lyndon Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, editors, *The Semantic Web*, pages 722–735, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[4] H. Bal, D. Epema, C. de Laat, R. van Nieuwpoort, J. Romein, F. Seinstra, C. Snoek, and H. Wijshoff. A Medium-Scale Distributed System for Computer Science Research: Infrastructure for the Long Term. *Computer*, 49(05):54–63, may 2016.

[5] Stephan Bloehdorn and York Sure. Kernel Methods for Mining Instance Data in Ontologies. In Karl Aberer, Key-Sun Choi, Natasha Noy, Dean Allemang, Kyung-Il Lee, Lyndon Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, editors, *The Semantic Web*, pages 58–71, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[6] Peter Bloem, Xander Wilcke, Lucas van Berkel, and Victor de Boer. kgbench: A Collection of Knowledge Graph Datasets for Evaluating Relational and Multimodal Machine Learning. In *Eighteenth Extended Semantic Web Conference - Resources Track*, 2021.

[7] Till Blume, David Richerby, and Ansgar Scherp. Incremental and parallel computation of structural graph summaries for evolving graphs. In *CIKM*, pages 75–84. ACM, 2020.

[8] Till Blume, David Richerby, and Ansgar Scherp. FLUID: A Common Model for Semantic Structural Graph Summaries Based on Equivalence Relations. *Theoretical Computer Science*, 854:136–158, Jan 2021.

[9] Stéphane Campinas. Graph Summarisation of Web Data: Data-driven Generation of Structured Representations, 2016.

[10] Q. Chen, A. Lim, and K.W. Ong. D(K)-Index: An Adaptive Structural Summary for Graph-Structured Data, 2003.

[11] Victor De Boer, Jan Wielemaker, Judith Van Gent, Michiel Hildebrand, Antoine Isaac, Jacco Van Ossenbruggen, and Guus Schreiber. Supporting Linked Data Production for Cultural Heritage Institutes: The Amsterdam Museum Case Study. In *Extended Semantic Web Conference*, pages 733–747. Springer, 2012.

[12] Asim Kumar Debnath, Rosa L Lopez de Compadre, Gargi Debnath, Alan J Shusterman, and Corwin Hansch. Structure-Activity Relationship of Mutagenic Aromatic and Heteroaromatic Nitro Compounds. Correlation with Molecular Orbital Energies and Hydrophobicity. *Journal of medicinal chemistry*, 34(2):786–797, 1991.

[13] Chenhui Deng, Zhiqiang Zhao, Yongyu Wang, Zhiru Zhang, and Zhuo Feng. GraphZoom: A Multi-Level Spectral Approach for Accurate and Scalable Graph Embedding, 2020.

[14] Dieter Fensel, Umutcan Şimşek, Kevin Angele, Elwin Huaman, Elias Kärle, Oleksandra Panasiuk, Ioan Toma, Jürgen Umbrich, and Alexander Wahler. *Introduction: What Is a Knowledge Graph?*, pages 1–10. Springer International Publishing, Cham, 2020.

[15] Matthias Fey and Jan E. Lenssen. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[16] F. Goasdoué, Pawel Guzewicz, and I. Manolescu. RDF graph summarization for first-sight structure discovery. *The VLDB Journal*, pages 1–28, 2020.

[17] R. Kaushik, P. Shenoy, P. Bohannon, and E. Gudes. Exploiting Local Similarity for Indexing Paths in Graph-Structured Data. In *Proceedings 18th International Conference on Data Engineering*, pages 129–140, 2002.

[18] R. Kaushik, P. Shenoy, P. Bohannon, and E. Gudes. Exploiting Local Similarity for Indexing Paths in Graph-Structured Data. *Proceedings 18th International Conference on Data Engineering*, pages 129–140, 2002.

[19] Raghav Kaushik, Philip Bohannon, Jeffrey F Naughton, and Henry F Korth. Covering Indexes for Branching Path Queries. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 133–144, 2002.

[20] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks, 2017.

[21] Mathias Konrath, Thomas Gottron, Steffen Staab, and Ansgar Scherp. SchemEX — Efficient Construction of a Data Catalogue by Stream-Based Indexing of Linked Data. *Journal of Web Semantics*, 16:52–58, 2012. The Semantic Web Challenge 2011.

[22] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[23] Chen Qun, Andrew Lim, and Kian Win Ong. D(k)-Index: An Adaptive Structural Summary for Graph-Structured Data. pages 134–144, 01 2003.

[24] Petar Ristoski, Gerben Klaas Dirk de Vries, and Heiko Paulheim. A Collection of Benchmark Datasets for Systematic Evaluations of Machine Learning on the Semantic Web. In Paul Groth, Elena Simperl, Alasdair Gray, Marta Sabou, Markus Krötzsch, Freddy Lecue, Fabian Flöck, and Yolanda Gil, editors, *The Semantic Web – ISWC 2016*, pages 186–194, Cham, 2016. Springer International Publishing.

[25] Noa Roy-Hubara, Lior Rokach, Bracha Shapira, and Peretz Shoval. Modeling Graph Database Schema. *IT Professional*, 19(6):34–43, 2017.

[26] Guillaume Salha, Romain Hennequin, Viet Anh Tran, and Michalis Vazirgiannis. A Degeneracy Framework for Scalable Graph Autoencoders, 2019.

[27] Davide Sangiorgi. On the Origins of Bisimulation and Coinduction. *ACM Trans. Program. Lang. Syst.*, 31(4), May 2009.

[28] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling Relational Data with Graph Convolutional Networks. In Aldo Gangemi, Roberto Navigli, Maria-Esther Vidal, Pascal Hitzler, Raphaël Troncy, Laura Hollink, Anna Tordai, and Mehwish Alam, editors, *The Semantic Web*, pages 593–607, Cham, 2018. Springer International Publishing.

[29] Thiviyan Thanapalasingam, Lucas van Berkel, Peter Bloem, and Paul Groth. Relational Graph Convolutional Networks: A Closer Look, 2021.

[30] Thanh Tran, Günter Ladwig, and Sebastian Rudolph. Managing Structured and Semistructured RDF Data Using Structure Indexes. *IEEE Transactions on Knowledge and Data Engineering*, 25(9):2076–2089, 2013.

[31] Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha Talukdar. Composition-based Multi-Relational Graph Convolutional Networks, 2020.

[32] Shichao Zhu, Chuan Zhou, Shirui Pan, Xingquan Zhu, and Bin Wang. Relation Structure-Aware Heterogeneous Graph Neural Network. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 1534–1539, 2019.

A Appendix

Listing 1: SPARQL query to map summary nodes to original nodes types adapted from [9].

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
CONSTRUCT {
    ?hs <http://issummaryof> ?s .
    ?ho <http://issummaryof> ?o .
}
WHERE
{
    {
        SELECT ?s (SHA1(group_concat(?p; separator = ",")) as ?sID) WHERE {
            SELECT DISTINCT ?s ?p {
                ?s ?p ?_
            } order by ?p
        } group by ?s
    }
    BIND(URI(CONCAT("http://example.org/", ?sID)) AS ?hs)
    ?s ?p ?o
    OPTIONAL {
        {
            SELECT ?o (SHA1(group_concat(?p; separator = ",")) as ?oID) WHERE {
                SELECT DISTINCT ?o ?p {
                    ?o ?p ?_
                } order by ?p
            } group by ?o
        }
        BIND(URI(CONCAT("http://example.org/", ?oID)) AS ?ho2)
    }

    BIND(IF(BOUND(?oID), ?ho2, ?o) AS ?ho)
}
```

Listing 2: SPARQL query to produce Attributes Summary adapted from [9].

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
CONSTRUCT {
    ?hs ?p ?ho .
}
WHERE
{
    {
        SELECT ?s (SHA1(group_concat(?p; separator = ",")) as ?sID) WHERE {
            SELECT DISTINCT ?s ?p {
                ?s ?p ?_
            } order by ?p
        } group by ?s
    }
    BIND(URI(CONCAT("http://example.org/", ?sID)) AS ?hs)
    ?s ?p ?o
    OPTIONAL {
        {
            SELECT ?o (SHA1(group_concat(?p; separator = ",")) as ?oID) WHERE {
                SELECT DISTINCT ?o ?p {
                    ?o ?p ?_
                } order by ?p
            } group by ?o
        }
        BIND(URI(CONCAT("http://example.org/", ?oID)) AS ?ho2)
    }

    BIND(IF(BOUND(?oID), ?ho2, ?o) AS ?ho)
}
```

B Appendix

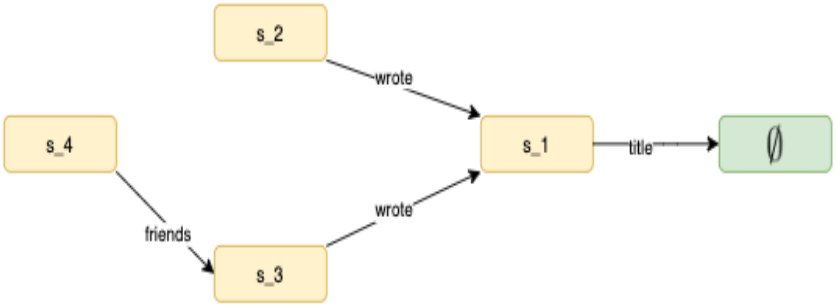


Figure 9: IO Summary Summary performed on Knowledge Graph from Figure 2.

V	W_{io}
v_1, v_2	s_1
v_3	s_2
v_4	s_3
v_5	s_4

Table 3: The table indicates the mapping produced by the IO Summary from original nodes $\in V$ to summary nodes $\in W_{io}$ under $R_{io}(V, W_{io})$.

C Appendix

Figure 10: Incoming Attributes Summary performed on Knowledge Graph from Figure 2.

V	W_{ia}
v_1, v_2	s_2
v_3, v_5	s_3
v_4	s_1

Table 4: The table indicates the mapping produced by the Incoming Attributes Summary from original nodes $\in V$ to summary nodes $\in W_{ia}$ under $R_{ia}(V, W_{ia})$.

D Appendix

	AIFB	AIFB	BGS	AM
Entities	8285	23,644	333,845	1,666,764
Relations	45	23	103	133
Edges	29,043	74,227	916,199	5,988,321
Classes	26	113	1	21

Table 5: Statistics of commonly used RDF format datasets in research

E Appendix

	Summary	TransferL	Benchmark
number hidden units	16	16	16
number RGCN layers	2	2	2
learning rate	0.01	0.01	0.01
weight decay	0.0005	0.0005	0.0005
number of bases	None	None	None
layer 1 frozen	False	False	False
layer 2 frozen	False	False	False
number training iterations	51	51	51
no literals in summary	False	-	-

Table 6: Hyperaparameters of the three different RGCN models

F Appendix

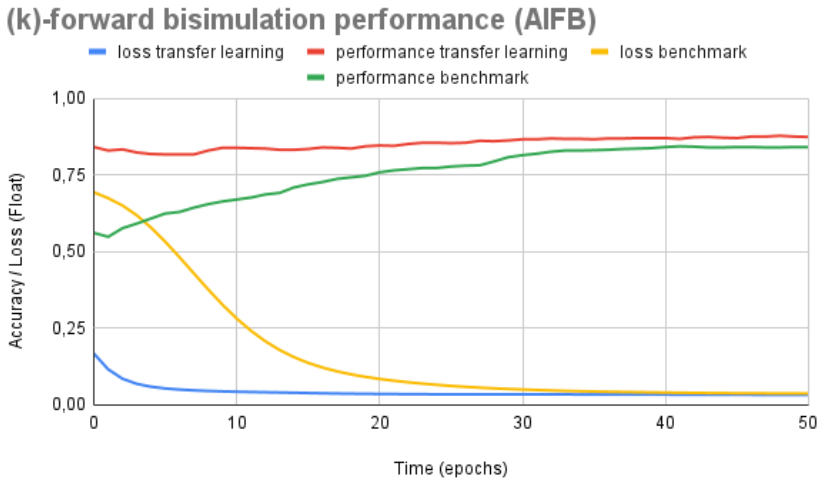


Figure 11: Graph showing the performance of the summarisation relation (k)-forward bisimulation on the dataset AIFB

G Appendix

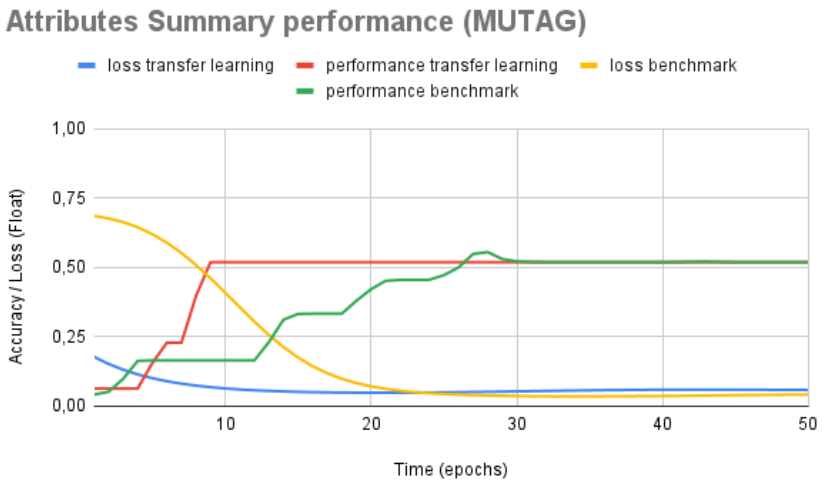


Figure 12: Graph showing the performance of the summarisation relation PC on the dataset MUTAG

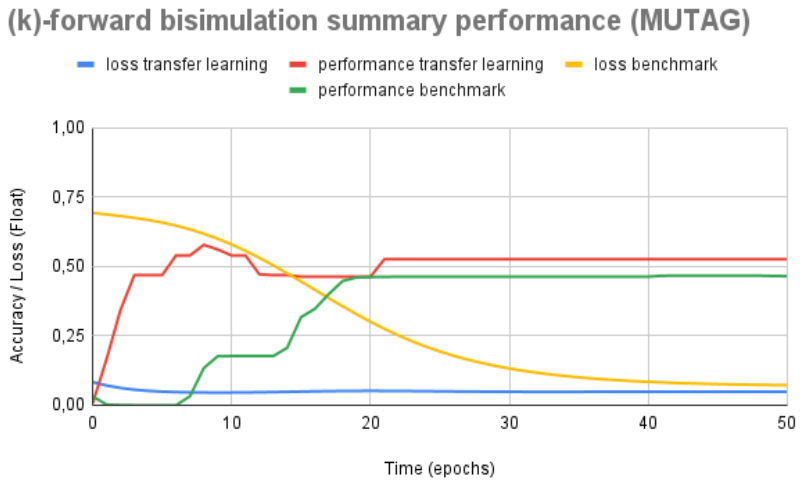


Figure 13: Graph showing the performance of the summarisation relation (k)-forward bisimulation on the dataset MUTAG

H Appendix

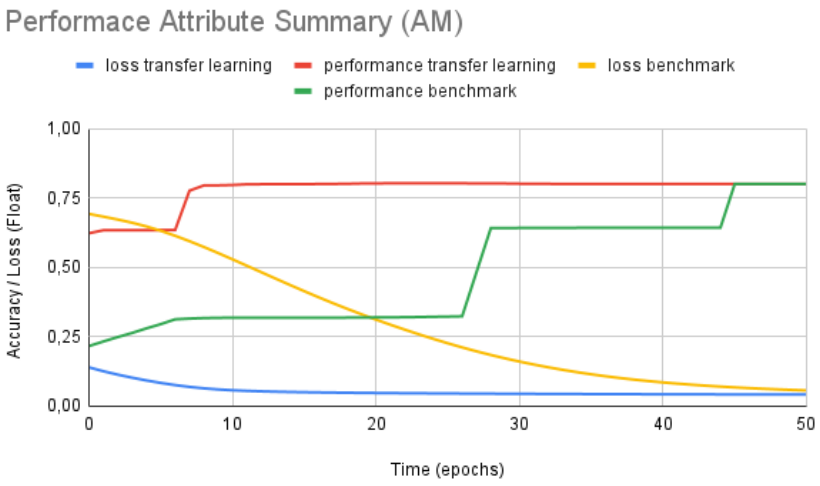


Figure 14: Graph showing the performance of the summarisation relation PC on the dataset AM

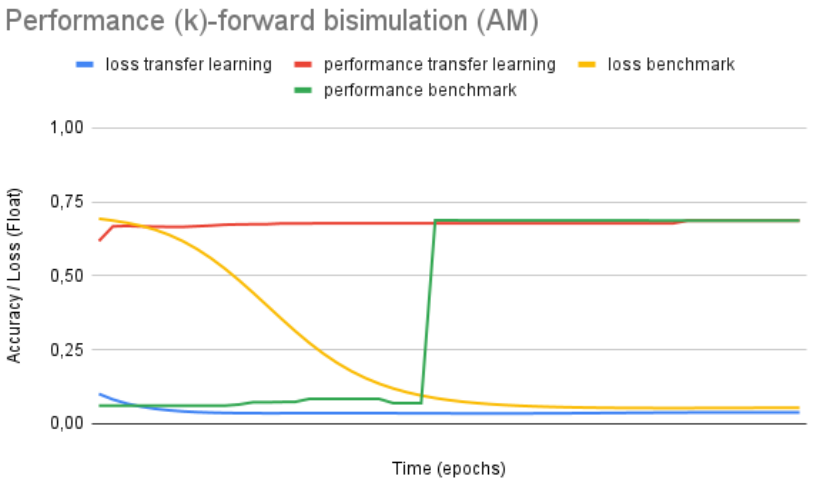


Figure 15: Graph showing the performance of the summarisation relation (k)-forward bisimulation on the dataset AM

I Appendix

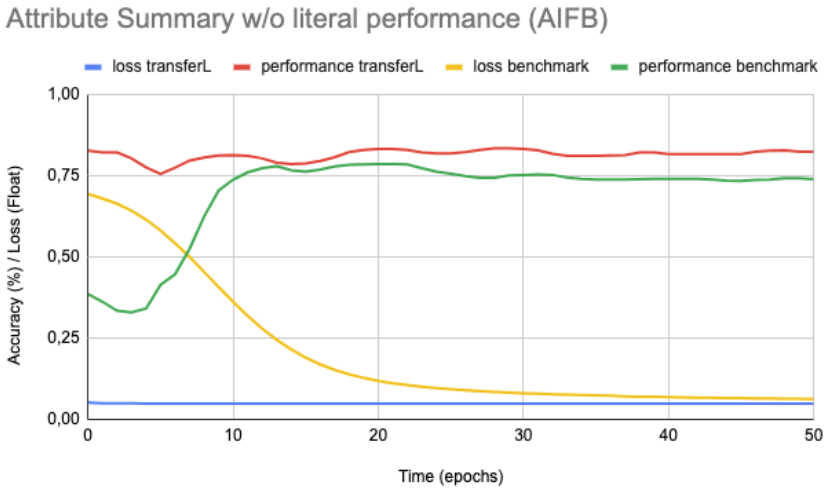


Figure 16: Graph showing the performance of the Attributes Summary without literals on the dataset AIFB.

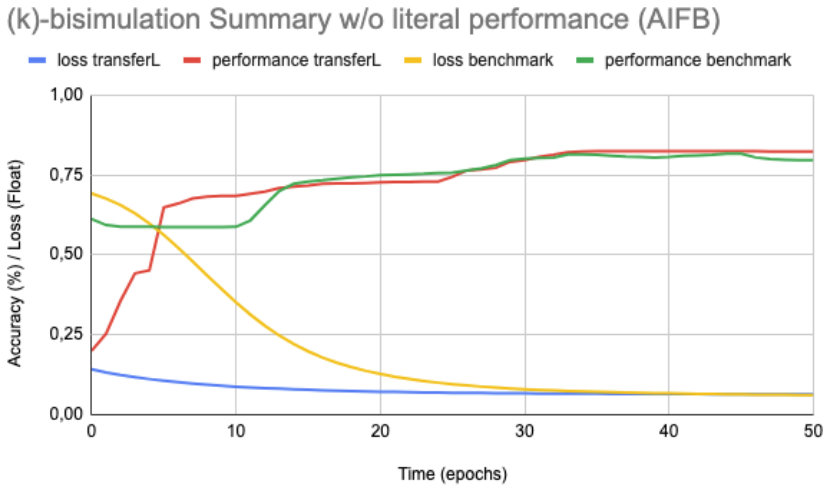


Figure 17: Graph showing the performance of the (k)-forward bisimulation Summary without literals on the dataset AIFB.

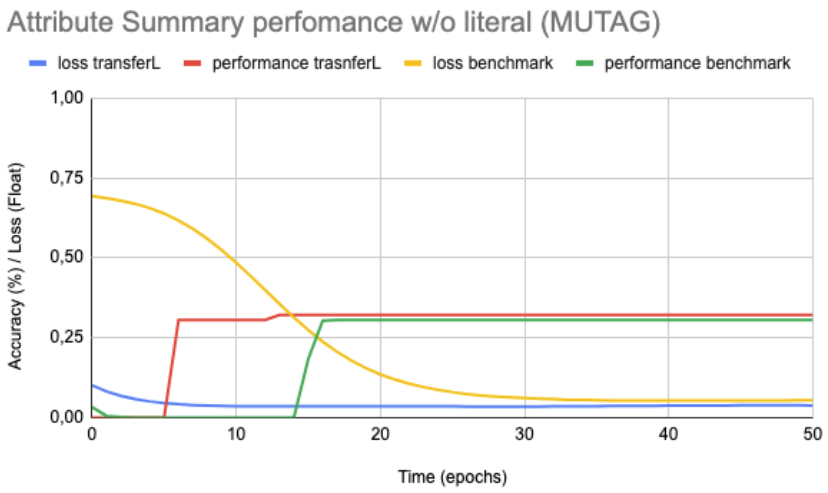


Figure 18: Graph showing the performance of the Attributes Summary without literals on the dataset MUTAG.

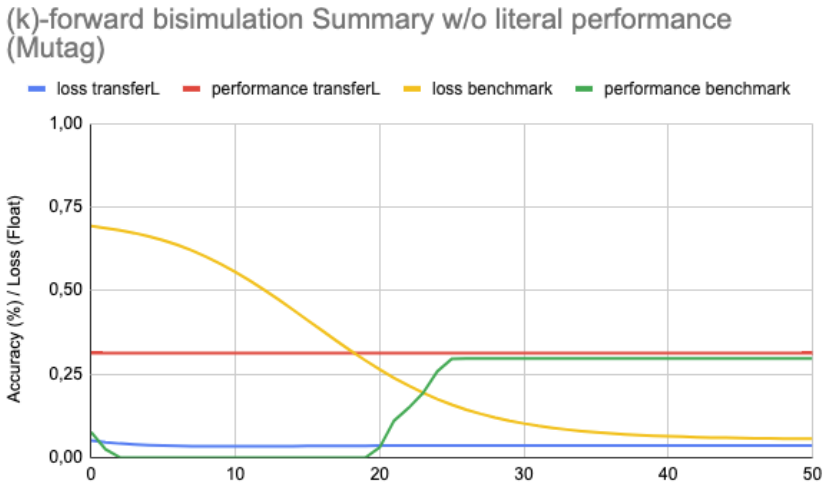


Figure 19: Graph showing the performance of the (k)-forward bisimulation Summary without literals on the dataset MUTAG.