Scaling R-GCN Training with Graph Summarization

Alessandro Generale*
Vrije Universiteit Amsterdam
The Netherlands

Till Blume
Ernst & Young GmbH WPG – R&D
Berlin, Germany
till.blume@de.ey.com

Michael Cochez
Vrije Universiteit Amsterdam
Elsevier Discovery Lab
Amsterdam, The Netherlands
m.cochez@vu.nl

ABSTRACT

Training of Relational Graph Convolutional Networks (R-GCN) is a memory intense task. The amount of gradient information that needs to be stored during training for real-world graphs is often too large for the amount of memory available on most GPUs. In this work, we experiment with the use of graph summarization techniques to compress the graph and hence reduce the amount of memory needed. After training the R-GCN on the graph summary, we transfer the weights back to the original graph and attempt to perform inference on it. We obtain reasonable results on the *AIFB*, *MUTAG* and *AM* datasets. Our experiments show that training on the graph summary can yield a comparable or higher accuracy to training on the original graphs. Furthermore, if we take the time to compute the summary out of the equation, we observe that the smaller graph representations obtained with graph summarization methods reduces the computational overhead. However, further experiments are needed to evaluate additional graph summary models and whether our findings also holds true for very large graphs.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning algorithms**; **Semantic networks**.

KEYWORDS

graph neural network, scalability, graph summarization

ACM Reference Format:
Alessandro Generale, Till Blume, and Michael Cochez. 2022. Scaling R-GCN Training with Graph Summarization. In *Companion Proceedings of the Web Conference 2022 (WWW '22 Companion), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3487553.3524719

1 INTRODUCTION

Knowledge Graphs (KG) emerged as an abstraction to represent and exploit complex data, and to ease accessibility [11]. As a result, extensive collections of data stored in KGs are now publicly available, spurring the interest in investigating novel technologies that aim to structure and analyze such data. However, KGs, including well-known ones such as DBpedia and WikiData [1], remain incomplete. There is an evident trade-off between the quantity of data available and its adequate coverage (completeness) [3]. Predicting missing information in KGs, e.g., predicting missing links, is the main focus of statistical relational learning [29]. However, such methods are challenged by large quantities of data and the unknown structure of KGs, harming the scalability of applications [24].

Several techniques have been developed to tackle these larger graphs while still retaining their structure. First, training of GCNs can be scaled by developing a memory-optimized implementation or distributing the computations over multiple GPUs [22]. Second, some techniques try to work with condensed representation of the KG that retains the original structure of the graph. For example, Salha et al. [26] proposed to use a highly dense subset of nodes from the original graph. Deng and his co-authors [10] suggest the creation of 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 have been shown to improve classification accuracy and accelerate the graph embedding process [10].

In this work, we propose to use graph summarization techniques to create a condensed representation of the KG, i. e., the graph summary. In our approach, we train an R-GCN on the graph summary and, subsequently, transfer the obtained weights to an R-GCN based on the original KG. Finally, we evaluate the later R-GCN and then investigate how its performance behaves when trained further, compared to an R-GCN based on the full KG that was not pre-trained on a summary. From our experiments we observe that training on the graph summary can yield a comparable or higher accuracy to training on the original graph. Furthermore, we show that the smaller graph representations obtained with graph summarization methods reduces the computational overhead if we do not incorporate the time needed for the summarization.

2 RELATIONAL GRAPH CONVOLUTIONAL NETWORKS

Graph Convolutional Networks (GCNs) have spurred great interest in Machine Learning with graph entities. A GCN takes as input a graph, potentially with input features for the nodes, and outputs embeddings for each of the nodes. GCNs use a message-passing algorithm which means that neighboring information influences the embedding representation of a node. However, a drawback of a GCN is that it treats all relations within a heterogeneous graph the same. Hence, the R-GCN was proposed, which enables the inclusion of relational information into the Graph Neural Network (originally proposed by Schlichtkrull et al. [29] and extensively analyzed by Thanapalasingam et al.[30]).

*This paper is the outcome of the thesis work of the first author

WWW '22 Companion, April 25–29, 2022, Virtual Event, Lyon, France

ACM ISBN 978-1-4503-9130-6/22/04.
https://doi.org/10.1145/3487553.3524719

An R-GCN is defined as a convolution that performs message passing in the context of multi-relational graphs. The updated state of a node i after message passing step $l+1$ of this graph neural network is as follows:

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

This state is a combination of the state of the neighboring nodes after the previous message passing step l and the previous state of the node itself. In the formula, we sum over all relation types r in R and then over all neighbors j which have a relation r to node i (N_i^r). The state of these nodes is projected using relation specific matrices $W_r^{(l)}$ (the weight matrix for the relation r at layer l) before being summed. The R-GCN adds a self-connection when updating the representation of a node, and learns a dedicated weight matrix $W_0^{(l)}$ to project nodes onto themselves. Finally, the state is set to the value of that sum after applying a non-linear activation function. The state after several message passing steps form a representation for each node in the graph. These representations can be used for node classification or other downstream machine learning tasks.

The R-GCN model suffers from over-parametrisation. Schlichtkrull et al. [29] use basis decomposition to reduce the number of weight matrices. With matrix decomposition, each of the relations has to learn relation coefficients to scale the single projection matrix. Alternatively, a block diagonal decomposition can be used to reduce the parameters [29].

3 GRAPH SUMMARIZATION

Representing data as a graph is increasingly popular since graphs allow a more efficient and a more flexible implementation of certain applications compared to relational databases [12, 16]. Relational databases operate on tabular data models and employ strict data schema [15]. Graph databases use labeled nodes to represent entities such as people or places and labeled edges to represent relationships between entities [16]. The combination of node labels and edge labels can be considered the schema of the graph database [25].

In this paper, we define our (Knowledge) Graph as follows, and note that this definition includes RDF graphs[1] as a special case.

Definition 1 (Graph). *(definition obtained from [7] page 15). Let V be a set of nodes and L a set of labels. The set of directed labeled 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-labeling function.*

In the context of RDF web graphs, it is common practice to think of the graph as a set of triples (x,α,y). Such triple has a head or source node x that has a direct relationship α to a destination node y, also referred to as target or object node.

Figure 1 displays a small example KG. The set of nodes $\{v_1, v_2, v_3, v_4, v_5\}$ are entity nodes. The set of nodes {"*Book*1", "*Book*2"} 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"). Following [7], we define an attribute to be the label of an edge. In this case the attribute relating node v_1 and literal node "Book1" is *title*.

[1] https://www.w3.org/TR/rdf11-concepts/

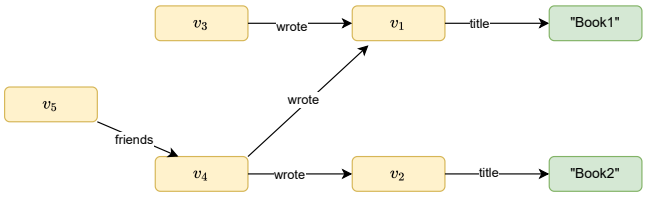


Figure 1: Fragment of a KG. The entities are the nodes v_1 to v_5 and the edges have attributes e.g. 'wrote', 'title' and 'friends'. The green nodes are Literals.

Such Knowledge Graphs allow to flexibly add new node labels and edge labels, i. e., adapt to new entities and relationships [11]. This flexibility regarding the data schema means that the inherent structure of the data may be unclear at first or evolve during the lifetime of a graph database. Graph summarization facilitates the identification of meaning and structure in data graphs [14, 21]. Thus, graph summaries can be used, among other applications, to discover the data schema of an existing graph database [8]. When we describe a graph summarization techniques, we typically characterize it with three dimensions [7]: (I) size of the graph, (II) size of the graph summary, (III) the impact of the graph's heterogeneity on the graph 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 [7]. A fourth dimension of graph summarization is the role of the literal nodes. Often, summaries are computed taking into consideration literal nodes but their content is abstracted away. 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.

Intuitively, graph summaries are condensed representations of graphs such that a set of chosen features of the graph summary are equivalent to the features in the original graph [5]. To achieve this, structural graph summaries partition nodes based on equivalent subgraphs. To determine equivalent subgraphs, features such as specific combinations of labels in that subgraph are used. We call this the schema of nodes.

A graph is considered heterogeneous when not consistent regarding its labeling and structure, e. g., there is a huge variety in schema used to describe entities and their relationship [7]. The more heterogeneous a graph, the more complex it becomes to create a concise and precise summary. Concise refers to the size of the summary, whereas precise concerns the preservation of the entity structure. In practice, a reasonable summary is typically smaller than the original graph [28]. Below, we discuss two families of graph summaries: approximate graph summaries and precise graph summaries.

3.1 Approximate Graph Summaries

Approximate graph summaries exploit the different features of the data that form the local information of a node. For instance, a node has incoming and outgoing edges with different labels also called attributes [7]. 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 summarization relation. The partitions, which correspond to summary nodes, are produced based on the summarization 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.

3.1.1 Attributes Summary. The Attributes Summary has been defined in previous literature [6, 7] as an approximate graph summary. In the context of a KG, 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 KGs, are aggregated to the same partition or summary node. The literal nodes information when performing the following summarization technique is abstracted away. Figure 2 shows the Attributes Summary result on the KG fragment presented in Figure 1.

Definition 2 (Summarization Relation R_a). *(definition obtained from [7] 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 power set of attributes in the graph. Therefore, S_a corresponds to the Attribute Summary of G according to the summarization relation $R_a \subseteq V \times W_a$ defined as follows:

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

This means the nodes are placed in the same partition if the sets containing their attributes are the same. Note that no duplicates triples are added as we store the triples into a set, and by definition a set does not contain duplicate items.

Computing this relation and assigning a new label to each partition, we obtain the mapping in Table 1. We can no use this mapping to summarize our running example from Figure 1 into the graph in Figure 2. For example, the nodes v_1 and v_2 belong to the same partition as they share the same attribute set $\{title\}$ and hence they are summarized into the same node, which received the label s_2.

The summarization relation described above maps nodes without any outgoing edges to an *empty summary node*, which we give the label $\emptyset$. Usually that node represents literal nodes as they do not have any outgoing edges. In our example, the string literals 'Book1' and 'Book2' map to the same empty summary node.

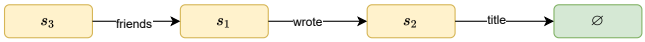


Figure 2: Attributes Summary graph with generated mapping from original node to summary node. The green node labeled by the empty set symbol abstracts the content of Literal nodes.

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 $R_a(V, W_a)$.

3.1.2 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 (Summarization Relation R_{io}). *(definition obtained from [7] 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 power set of attributes in the graph. Therefore, S_a corresponds to the Incoming Outgoing Attribute Summary of G according to the summarization 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 summarization relation R_{io}.

3.1.3 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 summarization techniques.

Definition 4 (Summarization Relation R_{ia}). *(definition obtained from [7] 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 power set of attributes in the graph. Therefore, S_{ia} corresponds to the Incoming Attributes Summary of G according to the summarization 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 summarization relation R_{ia}.

3.2 Precise Graph Summaries

Partitioning nodes based solely based on their local subgraph schema structures has certain drawbacks. In contrast to such approximate graph summaries, precise graph summaries compute the schema of nodes considering neighboring nodes over multiple hops [9, 18]. Similarly, the notion of aggregating nodes based on local and neighboring 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 summarization relation is more complex and takes more information from the original graph into account. In order for two nodes to belong to the same partition, neighboring nodes must also share the same schema. A graph summary is said to be precise when indiscernible from the original graph [7]. 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 homomorphism [7]. Additionally, the structure of the original graph is kept in the summary but the inverse may not hold true. Comparing the mapping in Table 1, there is no unique way to always reconstruct the original KG.

Definition 5 (Precise Graph Summary). *(definition obtained from [7] 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 summarization 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 2, 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 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).

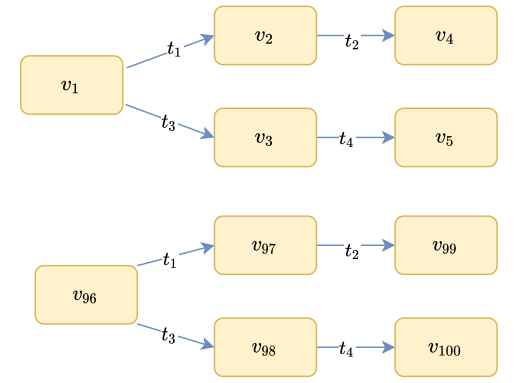
3.2.1 Bisimulation. The intuition behind bisimulation, in the context of graphs, is the interpretation of the graph data as transition systems to discover structurally equivalent parts [6, 27]. Two nodes' states are bisimilar if their states change following equivalent edge relations [6]. Bisimulation is a binary relation that relates two arbitrary nodes in a KG when itself and its inverse are simulations [7]. 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 useful to realize 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.

3.2.2 (k)-forward Bisimulation. (k)-forward bisimulation is an example of the many graph summarization techniques which aggregates nodes based on neighbors' schema over multiple hops [18, 20, 31]. It has also been defined as a stratified bisimulation in [6], a summarization relation that is restricted to a maximum path length of k-edges. Other summarization techniques that propose bisimulation over k hops also consider the edge direction [27]. These are referred to as *forward*, *backward* and *forward/backward* [17] bisimulation. In this paper, we use the *FLUID* framework [4], which allows us to create a *forward* (k)-bisimulation summary. Figure 3a represents a small fragment of a KG. A forward (k)-bisimulation with chaining parameter $k = 3$ is performed on this small entity graph. The resulting summary graph with alongside the one-to-one mapping from original node to summary node is shown in Figure 3b.

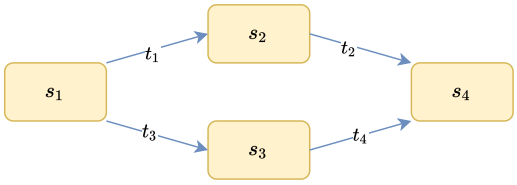
4 METHODS

In this work, we focus on multi-label classification on the types of the nodes. Therefore, the R-GCN uses of a summarized version of the original entity graph and infers the `rdf:type` relations for each of the original nodes. We expect that the summarized version of the graph retains enough structure of the entity graph, helping to generalize to the complex hierarchy of types in different nodes. It is possible to transfer the parameters of the summarized nodes back to the original nodes by storing links between nodes in the original graph and in the graph summary. This allows us to evaluate the performance of a model only trained on the graph summary and, possibly, to continue training on the original graph. As baseline, we train an R-GCN only on the original graph. All used R-GCN models follow the architecture proposed in [29]. However, the models in our work use *binary cross-entropy* loss and apply a sigmoid activation function on the output to predict multiple labels.

An issue raised by Bloem et al. [3] is the low number of labeled nodes in the datasets. This is an evident problem when it comes to testing the actual performance. To counter this issue we capture a specific relation that is present in most nodes and use its value as a label. We use the `rdf:type` relation which indicates the type of the



(a) Fraction of a KG



(b) (k)-bisimulation Graph Summary

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

node (a node can have multiple types) and use these as the classes for the nodes. This is explained in more detail in Section 4.2. The R-GCN models used in this work are implemented using Python 3.9.0 with the PyTorch geometric [13] framework.

4.1 Data Pre-processing & Summary Generation

The first step of the experiment is to create a map of original nodes to their labels. There are several ways to obtain these labels. It might be that they are not present in the graph, but rather retrieved from an external source. Alternatively, they are obtained from a set of chosen properties available in the graph [33]. In both cases nodes may belong to a single or multiple classes.

However, there are only few datasets equipped with a reasonable number of labeled nodes [3]. Hence, for our experiments we obtain labels by considering the `rdf:type` relation, and removing these triples from the original graph. Other properties of nodes within the KG are used to train the model.

The graph that was stripped of the `rdf:type` triples is used as an input for the two summarization frameworks to compute the Attributes Summary (see Section 3.1.1) and the forward (k)-bisimulation (see Section 3.2.2) with a chaining parameter $k = 3$. To compute the Attribute Summary, we use a SPARQL query developed by Campinas [7] and a slightly modified version which also gives us the mappings. To compute the forward (k)-bisimulation, we use the FLUID framework [4, 5]. We then also generate the mapping from the summary nodes to the original nodes.

Then, we create the summary graph by iterating over the edges in the original graph. For each edge, we apply the mapping function on the source and destination node, and use the outcomes and the relation type to form a new edge for our summarized graph. At this step, we also filter out the literal nodes and remove triples containing the Web Ontology Language attributes as suggested by Campinas [7].

4.2 Summary Nodes Labels

A summarization technique may map multiple nodes with different `rdf:type` values to the same partition or summary node. And hence, we have multiple labels for the same summary node. We create a *weighted multi-label* for each summary node by taking the relative frequency of each type in the partition (i.e., all nodes mapping to the summary node).

Figure 4 shows an example of the labeling 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 labeled $type_2$. Similarly, v_3 has two `rdf:type` relations and is therefore labeled $type_1$ 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 labeling is 0.5 for $type_1$ because only one node has that relation. The $type_4$ is labeled 1.0 because both nodes point to $type_4$.

The machine learning model that is trained with the summary representation uses these weighted labels. The transfer learning model and benchmark model (our baseline) train on binary values to predict the types of nodes.

4.3 Machine Learning Model: R-GCN

To perform the experiments, we set up three different R-GCN models, a model that learns on the graph summary, a second model with parameters transferred from the graph summary model, and finally, a model that follows a normal initialization to act as a baseline. The initialization 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.

As mentioned earlier, the model has been slightly modified in order to fit the purpose of the multi-label classification task. A *binary cross-entropy loss* function [23] 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 in Section 4.2 The labeling process gives values to the labels that depending on the frequency of that label applying to the original nodes inside the partition. A sigmoid activation function restricts the output of all models between 0 and 1, corresponding to the probabilities of the labels. The output of the model is then rounded to the nearest integer to match the original node labels.

A parameter that needs to be taken into consideration is the number of hidden units between the two R-GCN convolutions. Following the literature [19, 29], we set the number of hidden values to 16 for all used R-GCN models. As proposed in [19], we also do not employ a regularizer. The Adam optimizer [23] with a learning

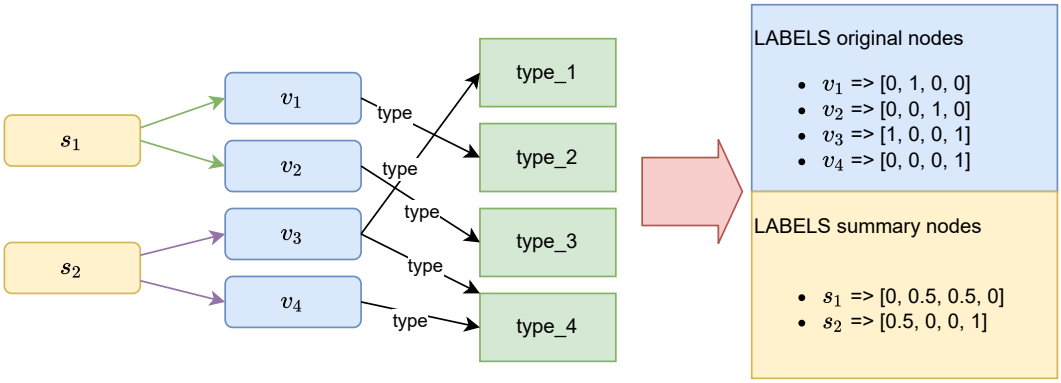


Figure 4: Diagram showing the original and summary nodes labeling process.

rate of $1.0 * 10^{-2}$ and a weight decay of $5.0 * 10^{-4}$ is applied as suggested in [29]. We do not use basis decomposition.

The model that learns on the summary graph is trained first for a total of 51 epochs. The parameters of this model are transferred to the second R-GCN through the mapping produced by the summarization 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. The parameters of the R-GCNs are detailed in Appendix A. The labeled instances are split 80% for training and 20% for testing.

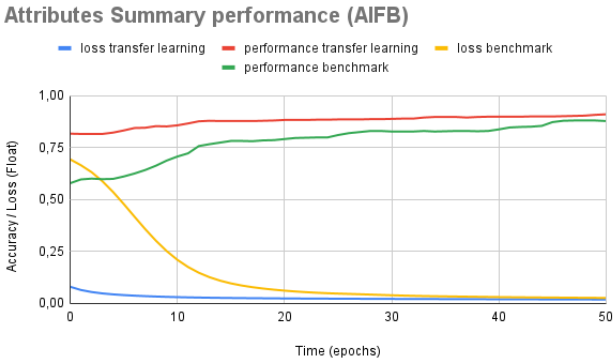


Figure 5: Graph showing the performance of the Attributes Summary performed on the dataset AIFB. The transfer learning model (red line) starts predicting at a higher accuracy than the baseline (green line) from early iterations and converges to a better optimal solution.

5 DATA SETS

Following existing research works [3, 29], we conduct our experiments on the AIFB, MUTAG and AM datasets. Statistics of these datasets can be found in Table 2.

Table 2: Statistics of commonly used RDF format datasets in research

	AIFB	AIFB	BGS	AM
Entities	8,285	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

6 EXPERIMENTAL RESULTS

We are interested in several results. First, we want to know how the transfer model, which gets initialized with the parameters from the summarized model performs. Secondly, we want to know how its performance changes with respect to the epochs if we perform more training on this transfer model. Finally, we have to compare these performances to the R-GCN which was no initialized had nothing to do with the summarized model (i.e., the baseline). In our graphs, we show 2 performance curves. The first one is the transfer learning model. At the zeroth epoch, this shows the performance without any additional training. After that, it shows the performance with additional training. The second curve shows the performance of the standard R-GCN with random initialization in function of the training epochs. In our graphs, we also include the value of the respective loss functions.

Figure 5 shows the performance of the transfer learning model trained on the Attributes Summary against the baseline for a single run of the on the *AIFB* dataset. Table 3 shows the average starting and converging accuracy. The average is calculated by performing (k)-fold cross validation with k set to 5. 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 initialized. The transfer learning model (red line) starts with a reasonably high accuracy of 81.7% for the first few epochs. The accuracy of the transfer learning

		Attributes Summary		(3)-forward bisimulation		Baseline	
		No training	After 50 epochs	No training	After 50 epochs	No training	After 50 epochs
Dataset	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 3: Results of the experiments with (k)-fold cross validation with k set to 5. Note that this cross validation was not performed for the AM dataset due to computational cost; only two runs are reported for each model.

model steadily increases up to 91.06% after 50 epochs. In contrast, the accuracy of the original model (green line) starts at a lower accuracy of 57.9% and converges at a lower accuracy of 87.82%.

The results for the transfer learning model trained on the (k)-forward bisimulation can be found in Figure 6. Summarily to the transfer learning model trained on the Attribute Summary, the transfer learning model trained on the (k)-forward bisimulation summary consistently outperforms the baseline model over all training epochs. This transfer learning model starts with an accuracy of 77.53% and converges to a final accuracy of 91.30% after 50 training epochs.

The dataset *AIFB* contains 8,285 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* KG. In this experiment, we need fewer training epochs for the transfer learning model to predict with a reasonably good accuracy compared to the baseline model. Additionally, we observe that the transfer learning model consistently outperforms the baseline model when trained for the same number of epochs.

The *MUTAG* dataset provided similar insights (see Appendix C for single runs). However, the positive effect of the summarization is not that large. After 20 to 25 epochs, the baseline model catches up with the summarized model. We think this difference has to do with the fact that both summarization 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 were 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 preprocessing step, making this an overall much harder task. In previous work, node classification on the *MUTAG* dataset was done considering as few as 2 classes [30].

Finally, we look at the largest KG, the *AM* dataset (see Appendix D for single runs). We notice that the transfer models show better results to the baseline models at the start, but that after a while the performance of the baseline catches up to become roughly the same.

The graph summaries which we used in our experiments consider the relations with literal nodes as well (see Section 3). As an ablation, we conducted experiments where we do not take the literal nodes into account when computing the summary. The results for single runs on the *AIFB* and *MUTAG* datasets can be found in Appendix E. What we note is that the performance of the transfer learning models becomes less consistent and often fall below the baseline. Hence, considering literal information seems critical for the performance of transfer learning models trained on graph summaries.

7 CONCLUSION AND FUTURE WORK

We introduced an approach to use graph summaries to efficiently and effectively train R-GCN models on KGs. In our experiments, the R-GCN models trained on the graph summary often outperformed the baseline models and typically started at a much higher initial accuracy before the first training epoch. The only exception is the model trained on the forward (k) bisimulation summary on the *MUTAG* dataset, which yields at 4% lower accuracy after 50 training epochs than the baseline. In all other cases, the transfer learning model outperforms the baseline. What we identify is that transferring the parameters learned from the graph summary model leads to a jump-start of the learning process. This supports the importance and relevance of graph summarization methods, whose smaller graph representations scale down and reduce the computational overhead involved with novel machine learning models dealing with large KGs. However, due to the complexity of the classification task, the variation in the accuracy remains large. The summarization relation (k)-forward bisimulation and Attributes Summary behaved similarly in the training behavior for the *AIFB* and *AM* datasets.

For future work, we plan to extend the experiment with additional tasks. It might be possible to include a baseline study comparing the results with previously reported performances on single-class prediction. Besides, while the graphs we used are common in heterogeneous graph neural networks literature, experiments with larger graphs are necessary.

Another aspect to further experiment with is the hyperparameter k for the (k)-forward bisimulation. In our experiments, the parameter k was fixed to 3. It remains open to investigate a possible relationship between the optimal hyperparameter k in the graph summary and the optimal number of layers in an R-GCN.

In addition to these, the model trained with the graph summary can only learn for the nodes that are mapped. A number of nodes is discarded because such entities are not mapped when producing the graph summary, so their embedding will initially be random. The number of discarded nodes can be reduced by increasing k or by using more complex summarization methods.

Finally, one could also investigate whether these summarization techniques can also be used for other graph neural networks besides R-GCN models, for example for the CompGCN [32].

In this work, we established a framework that uses graph summaries to efficiently and efficiently train R-GCN to predict nodes' labels. The model's performance observed and reported for the *AIFB* and *AM* datasets for both summarization methods suggests that summarization can be used to scale training without harming the performance and even increase the performance.

ACKNOWLEDGMENTS

We used the DAS5 system [2] for training the machine learning models. Part of this research was funded by Elsevier's discovery Lab.

REFERENCES

[1] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. DBpedia: A Nucleus for a Web of Open Data. In *The Semantic Web*, 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 (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 722–735.

[2] H. Bal, D. Epema, C. de Laat, R. van Nieuwpoort, J. Romein, F. Seinstra, C. Snoek, and H. Wijshoff. 2016. A Medium-Scale Distributed System for Computer Science Research: Infrastructure for the Long Term. *Computer* 49, 05 (may 2016), 54–63. https://doi.org/10.1109/MC.2016.127

[3] Peter Bloem, Xander Wilcke, Lucas van Berkel, and Victor de Boer. 2021. kgbench: A Collection of Knowledge Graph Datasets for Evaluating Relational and Multimodal Machine Learning. In *The Semantic Web*, Ruben Verborgh, Katja Hose, Heiko Paulheim, Pierre-Antoine Champin, Maria Maleshkova, Oscar Corcho, Petar Ristoski, and Mehwish Alam (Eds.). Springer International Publishing, Cham, 614–630. https://openreview.net/forum?id=yeK_9wxRDbA

[4] Till Blume, David Richerby, and Ansgar Scherp. 2020. Incremental and Parallel Computation of Structural Graph Summaries for Evolving Graphs. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management* (Virtual Event, Ireland) *(CIKM '20)*. Association for Computing Machinery, New York, NY, USA, 75–84. https://doi.org/10.1145/3340531.3411878

[5] Till Blume, David Richerby, and Ansgar Scherp. 2021. FLUID: A common model for semantic structural graph summaries based on equivalence relations. *Theor. Comput. Sci.* 854 (2021), 136–158. https://doi.org/10.1016/j.tcs.2020.12.019

[6] Till Blume, David Richerby, and Ansgar Scherp. 2021. FLUID: A Common Model for Semantic Structural Graph Summaries Based on Equivalence Relations. *Theoretical Computer Science* 854 (Jan 2021), 136–158. https://doi.org/10.1016/j.tcs.2020.12.019

[7] Stéphane Campinas. 2016. *Graph Summarisation of Web Data: Data-driven Generation of Structured Representations.* Ph. D. Dissertation. National University of Ireland–Galway. http://hdl.handle.net/10379/6495

[8] Šejla Čebirić, François Goasdoué, Haridimos Kondylakis, Dimitris Kotzinos, Ioana Manolescu, Georgia Troullinou, and Mussab Zneika. 2019. Summarizing semantic graphs: a survey. *International Journal on Very Large Data Bases (VLDB Journal)* 28, 3 (2019), 295–327. https://doi.org/10.1007/s00778-018-0528-3

[9] Qun Chen, Andrew Lim, and Kian Win Ong. 2003. D(k)-Index: An Adaptive Structural Summary for Graph-Structured Data. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data* (San Diego, California) *(SIGMOD '03)*. Association for Computing Machinery, New York, NY, USA, 134–144. https://doi.org/10.1145/872757.872776

[10] Chenhui Deng, Zhiqiang Zhao, Yongyu Wang, Zhiru Zhang, and Zhuo Feng. 2020. GraphZoom: A Multi-Level Spectral Approach for Accurate and Scalable Graph Embedding. arXiv:1910.02370 [cs.LG]

[11] Dieter Fensel, Umutcan Şimşek, Kevin Angele, Elwin Huaman, Elias Kärle, Oleksandra Panasiuk, Ioan Toma, Jürgen Umbrich, and Alexander Wahler. 2020. *Introduction: What Is a Knowledge Graph?* Springer International Publishing, Cham, 1–10. https://doi.org/10.1007/978-3-030-37439-6_1

[12] Diogo Fernandes and Jorge Bernardino. 2018. Graph Databases Comparison: AllegroGraph, ArangoDB, InfiniteGraph, Neo4J, and OrientDB. In *Proceedings of the International Conference on Data Science, Technology and Applications (DATA)*. SciTePress, 373–380. https://doi.org/10.5220/0006910203730380

[13] Matthias Fey and Jan E. Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric, In ICLR Workshop on Representation Learning on Graphs and Manifolds. *arXiv preprint arXiv:1903.02428.*

[14] François Goasdoué, Pawel Guzewicz, and Ioana Manolescu. 2020. RDF graph summarization for first-sight structure discovery. *The VLDB Journal* 29, 5 (April 2020), 1191–1218. https://doi.org/10.1007/s00778-020-00611-y

[15] Jan L Harrington. 2016. *Relational database design and implementation.* Morgan Kaufmann.

[16] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutiérrez, José Emilio Labra Gayo, Sabrina Kirrane, Sebastian Neumaier, Axel Polleres, Roberto Navigli, Axel-Cyrille Ngonga Ngomo, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan F. Sequeda, Steffen Staab, and Antoine Zimmermann. 2020. Knowledge Graphs. *CoRR* abs/2003.02320 (2020). arXiv:2003.02320 https://arxiv.org/abs/2003.02320

[17] Raghav Kaushik, Philip Bohannon, Jeffrey F Naughton, and Henry F Korth. 2002. Covering Indexes for Branching Path Queries. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data* (Madison, Wisconsin) *(SIGMOD '02)*. Association for Computing Machinery, New York, NY, USA, 133–144. https://doi.org/10.1145/564691.564707

[18] R. Kaushik, P. Shenoy, P. Bohannon, and E. Gudes. 2002. Exploiting Local Similarity for Indexing Paths in Graph-Structured Data. In *Proceedings 18th International Conference on Data Engineering*. IEEE Computer Society, Los Alamitos, CA, USA, 129–140. https://doi.org/10.1109/ICDE.2002.994703

[19] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. arXiv:1609.02907 [cs.LG]

[20] Mathias Konrath, Thomas Gottron, Steffen Staab, and Ansgar Scherp. 2012. SchemEX — Efficient Construction of a Data Catalogue by Stream-Based Indexing of Linked Data. *Journal of Web Semantics* 16 (2012), 52–58. https://doi.org/10.1016/j.websem.2012.06.002 The Semantic Web Challenge 2011.

[21] Yike Liu, Tara Safavi, Abhilash Dighe, and Danai Koutra. 2018. Graph Summarization Methods and Applications: A Survey. *Comput. Surveys* 51, 3 (2018), 62:1–62:34. https://doi.org/10.1145/3186727

[22] Vasimuddin Md, Sanchit Misra, Guixiang Ma, Ramanarayan Mohanty, Evangelos Georganas, Alexander Heinecke, Dhiraj Kalamkar, Nesreen K. Ahmed, and Sasikanth Avancha. 2021. DistGNN: Scalable Distributed Training for Large-Scale Graph Neural Networks. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (St. Louis, Missouri) *(SC '21)*. Association for Computing Machinery, New York, NY, USA, Article 76, 14 pages. https://doi.org/10.1145/3458817.3480856

[23] 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. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

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

[25] Noa Roy-Hubara, Lior Rokach, Bracha Shapira, and Peretz Shoval. 2017. Modeling Graph Database Schema. *IT Professional* 19, 6 (2017), 34–43. https://doi.org/10.1109/MITP.2017.4241458

[26] Guillaume Salha, Romain Hennequin, Viet Anh Tran, and Michalis Vazirgiannis. 2019. A Degeneracy Framework for Scalable Graph Autoencoders. arXiv:1902.08813 [cs.LG]

[27] Davide Sangiorgi. 2009. On the Origins of Bisimulation and Coinduction. *ACM Trans. Program. Lang. Syst.* 31, 4, Article 15 (May 2009), 41 pages. https://doi.org/10.1145/1516507.1516510

[28] Ansgar Scherp and Till Blume. 2021. Schema-level Index Models for Web Data Search. *J. Data Intell.* 2, 1 (2021), 47–63. https://doi.org/10.26421/JDI2.1-3

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

[30] Thiviyan Thanapalasingam, Lucas van Berkel, Peter Bloem, and Paul Groth. 2021. Relational Graph Convolutional Networks: A Closer Look. arXiv:2107.10015 [cs.LG]

[31] Thanh Tran, Günter Ladwig, and Sebastian Rudolph. 2013. Managing Structured and Semistructured RDF Data Using Structure Indexes. *IEEE Transactions on Knowledge and Data Engineering* 25, 9 (2013), 2076–2089. https://doi.org/10.1109/TKDE.2012.134

[32] Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha Talukdar. 2020. Composition-based Multi-Relational Graph Convolutional Networks. arXiv:1911.03082 [cs.LG]

[33] S. Zhu, C. Zhou, S. Pan, X. Zhu, and B. Wang. 2019. Relation Structure-Aware Heterogeneous Graph Neural Network. In *2019 IEEE International Conference on Data Mining (ICDM)*. IEEE Computer Society, Los Alamitos, CA, USA, 1534–1539. https://doi.org/10.1109/ICDM.2019.00203

A MODEL PARAMETERS

	Summary	TransferL	baseline
number hidden units	16	16	16
number R-GCN 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
training iterations	51	51	51
no literals in summary	False	-	-

Table 4: Hyper-parameters of the three different R-GCN models

B (K)-FORWARD BISIMULATION (AIFB)

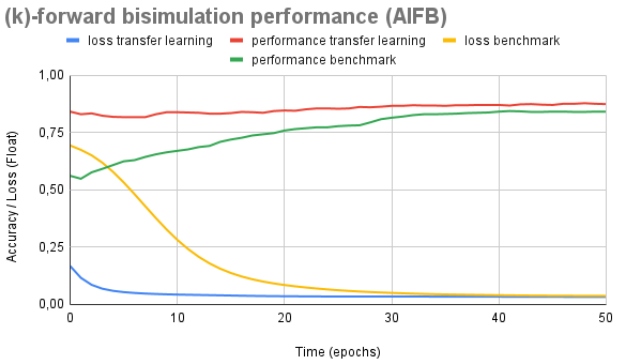


Figure 6: Graph showing the performance of the summarization relation (k)-forward bisimulation on the dataset AIFB

C ATTRIBUTES SUMMARY AND (K)-FORWARD BISIMULATION (MUTAG)

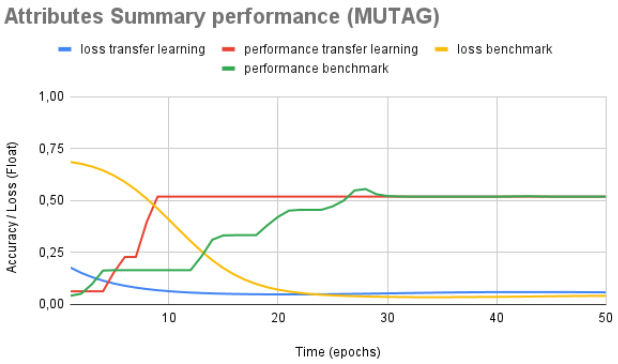


Figure 7: Graph showing the performance of the summarization relation PC on the dataset MUTAG

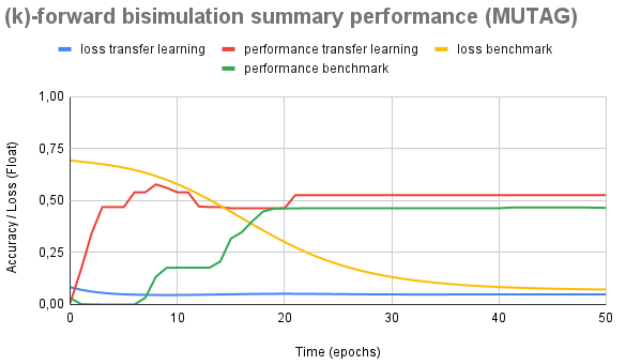


Figure 8: Graph showing the performance of the summarization relation (k)-forward bisimulation on the dataset MUTAG

D ATTRIBUTES SUMMARY AND (K)-FORWARD BISIMULATION (AM)

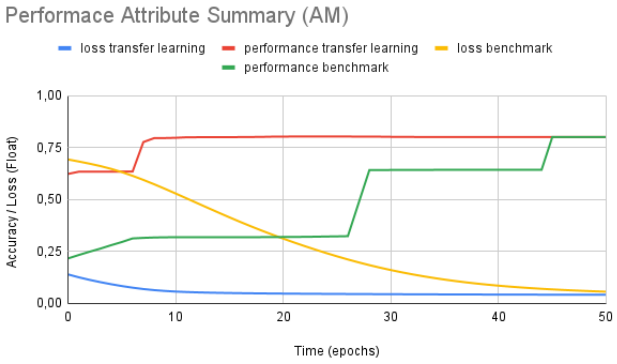


Figure 9: Graph showing the performance of the summarization relation PC on the dataset AM

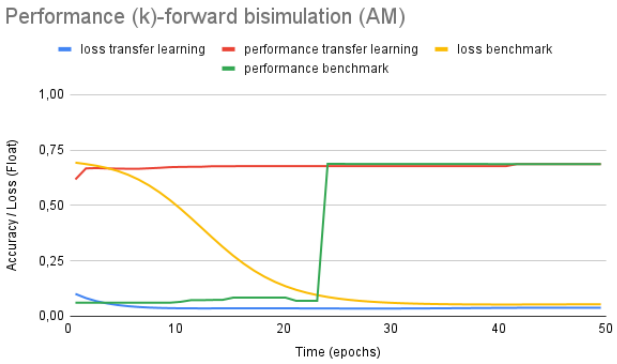


Figure 10: Graph showing the performance of the summarization relation (k)-forward bisimulation on the dataset AM

E THE EFFECT OF LEAVING OUT LITERALS

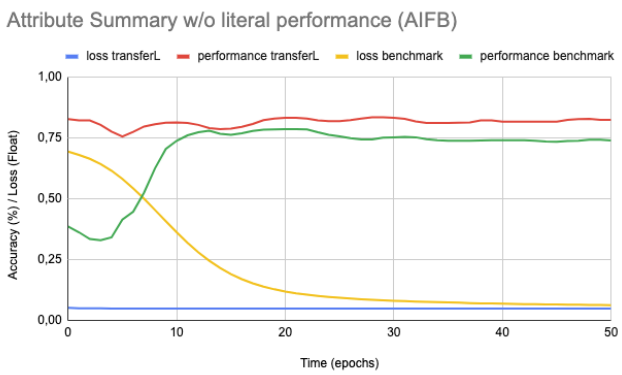


Figure 11: Graph showing the performance of the Attributes Summary without literals on the dataset AIFB.

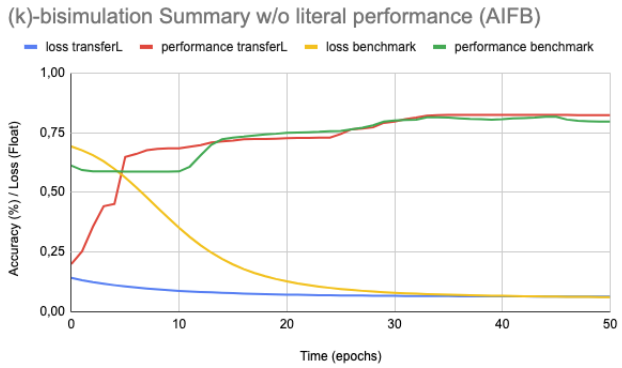


Figure 12: Graph showing the performance of the (k)-forward bisimulation Summary without literals on the dataset AIFB.

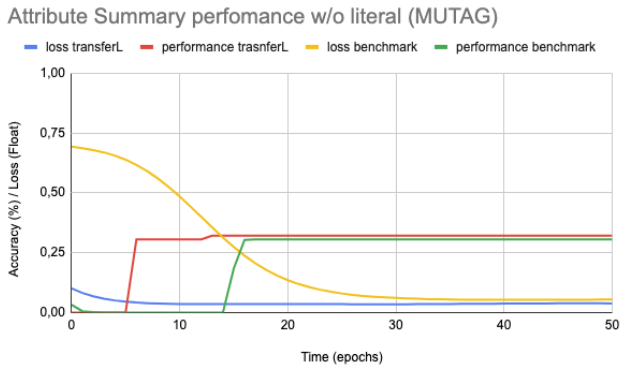


Figure 13: Graph showing the performance of the Attributes Summary without literals on the dataset MUTAG.

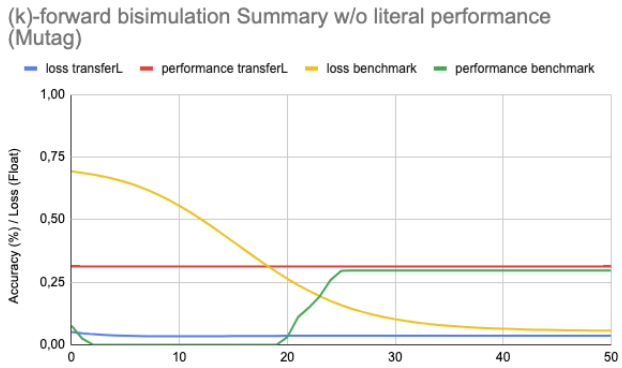


Figure 14: Graph showing the performance of the (k)-forward bisimulation Summary without literals on the dataset MUTAG.