University of Bergen
Department of informatics

Rule mining on extended knowledge graphs

Author: Johanna Jøsang
Supervisors: Ana Ozaki and Ricardo Guimarães

UNIVERSITETET I BERGEN
Det matematisk-naturvitenskapelige fakultet

June, 2022

Abstract

Knowledge graphs (KGs) have risen in both size and use over the past few years, and there are a range of approaches for evaluating information in them. Symbolic approaches, such as rule-based machine learning, offer an explainable way to determine the appropriateness of a new fact based on rules mined from the KG in question. Some of the most successful approaches for fact prediction in KGs today are knowledge graph embeddings (KGEs) that use deep neural networks, however, these lack the explainability of symbolic approaches. We would like to see how the extension of a KG using KGEs affects the rules mined from the KG. A set of rules is mined from a KG and compared with another set mined from an *extended* version of said KG. The experiments examine three classical KGEs: TransE, DistMult, and ComplEx, and use the rule mining algorithm AMIE3. AMIE3 is treated as a black box during the experiment, as the study only evaluates factors playing a role in the KG-extension process, one of which is the choice of KGE. The experiments show that there can be considerable discrepancies in the rules mined based on the choice of KGE and that TransE leads to a substantial amount of nonsensical rules being mined.

Acknowledgements

First and foremost, I would like to thank my two supervisors, Ana and Ricardo. Ana was, from the start, a fantastic guide for me in the field of academic research and taught me a lot about writing and publishing work. Ricardo was always eager to answer and discuss questions, and his attention to detail taught me a lot. A great thank you to both of you. You have given me more support through this master's than I could have hoped for.

I would like to thank my fellow master's students that battled with me over the two years. The workload is not so heavy when you are in it together. Thank you John Isak, Hans Martin, Knut, Halvor, Mathias, Magnus, Emir and Oda. You have been a fantastic group to work with, and I wish you all the best in your future work/studies.

My final thanks goes out to family and friends, who have helped me by listening to my thoughts and reviewing the thesis. Complicated ideas become tangible and manageable when you put the correct words to them. Thank you for helping me find them.

Johanna Jøsang
01 June, 2022

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 Context and motivation

Knowledge graphs (KGs) are an increasingly popular way to represent data [38]. A KG is often seen as a directed graph with labelled edges where the nodes represent the elements in a domain of interest (e.g. people), and the edges represent a relation between two elements. For instance, a KG such as Wikidata might include the node "Oslo" with an outgoing edge labelled "capital of" to the node "Norway". Knowledge graphs that are large and interesting are generally not complete. They may, for example, be extracted from natural language resources and may contain facts that are wrong or exhibit gaps in their knowledge. However, most of the present data is typically correct and implicitly contains meaningful rules. For example, Wikidata will implicitly contain the rule that siblings tend to have the same mother:

$$has\_sibling(x, z) \wedge has\_mother(z, y) \Rightarrow has\_mother(x, y)$$

There will of course be exceptions to this rule, but generally, it will hold. Such a rule can be used to infer new information in a KG, and using a set of such rules to make predictions is the core idea behind rule-based machine learning. Other statistical methods are accurate and scalable when it comes to inferring new facts from KGs, but one of the main issues of these approaches is that results usually are not explainable [9]. Rule-based machine learning approaches, on the other hand, provide an explanation in the form of the rules used to make

a prediction. Meilicke et al. showed that rule bases mined with the AMIE+ algorithm are good competitors and often outperform vector embedding models [52]. One explanation for this is that the standard benchmark datasets, such as WN18 and FB15k, have many relational regularities, such as symmetry and equivalence. Meilicke et al. also compared the two approaches for triple prediction and found that they complement each other. An ensemble of the two families of methods gave better results than either of the two alone.

This work by Meilicke et al. inspired the idea of using knowledge graph embedding (KGE) models to improve rule bases or vice versa. The idea is to use one technique to "extend" the original KG and thereafter train the second one on the extended KG. The KG-extension-and-mining pipeline can be done both ways, as shown in figure 1.1. A rule-base as the end result was chosen, ultimately because it results in an explainable model.

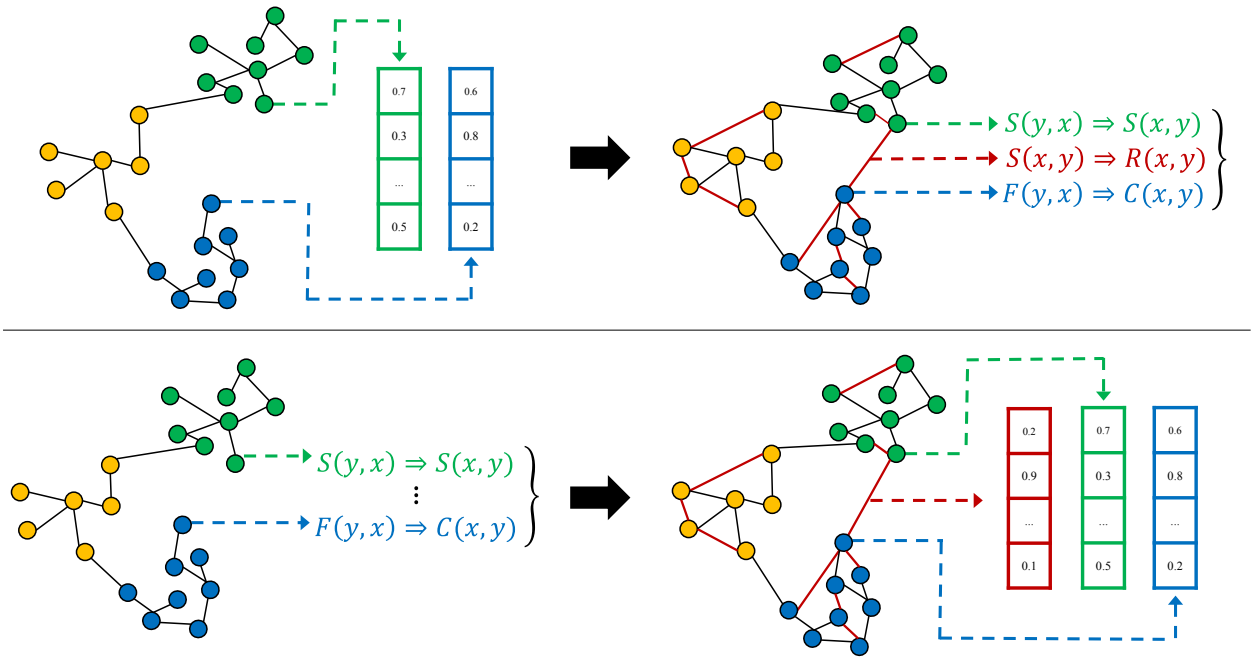


Figure 1.1: *The two versions of KG improvement for better model creation. In the top version, an embedding of the original KG is used to improve the graph, from which a rule base is mined. Red edges represent new links made by the models, which originally weren't present in the KG. The bottom half of the figure describes the same process but with the rule base used to improve the KG, resulting in an embedding of the improved KG.*

1.2 Research questions

With this idea in mind, the general question to be explored is what role different factors in the KG extension process have on the eventual rules that are mined from the extended dataset. The extension process can be summarized as follows: first, candidate facts are generated, then an embedding model ranks the candidates, and finally facts above a certain threshold are added to the KG. So the factors to be evaluated are:

- How candidate facts are generated.
- How candidates are ranked (choice of KGE model).
- Minimum rank for candidates to be added to the KG.

In addition to evaluation of these parameters, the current work explores three central research questions:

1. Does adding new plausible facts lead to new rules being mined?

2. How does the quality (approximated with PCA confidence) of new rules compare to the rules mined from the original KG?

3. Can the rules mined from the original KG also be mined after the KG is extended?

To answer these questions, the mentioned KG-extension-and-mining pipeline is implemented and the results evaluated. Experiments are conducted on two different datasets, both of which are commonly used for benchmarking KGEs. The most important results of this work are presented in a paper submitted to this year's *International Workshop on Knowledge Representation for Hybrid intelligence*. The submission can be accessed through this link: OneDrive.

1.3 Thesis outline

The outline for the rest of the thesis is as follows:

Chapter 2 - Background provides the reader with the background knowledge required for a proper understanding of the work.

Chapter 3 - Rule mining on extended knowledge graphs explains the methodology and material used for experiments.

Chapter 4 - Results presents and discusses the findings of the experiment.

Chapter 5 - Related work provides the reader with an overview of works related to the thesis.

Chapter 6 - Discussion evaluates and discusses the current work and explains an earlier approach explored during research.

As a final note, this thesis assumes that the reader is familiar with basic machine learning concepts, such as training, testing and overfitting. For an introduction or refresher to machine learning basics, chapter 5 in the book *Deep Learning*, by Goodfellow et al. [33] covers the topic quite well. The entire book is publicly available at https://www.deeplearningbook.org/.

Chapter 2

Background

This chapter presents KGs and different paradigms for how to interpret information missing from a KG. Then we continue with KGEs, examining the training and evaluation process and explaining the ideas behind the three KGE architectures used in the experiments. Finally, the chapter introduces rule mining by first giving a general overview of rule-based machine learning before focusing on the *association rule mining* approach.

2.1 Knowledge Graphs

There is no single agreed-upon definition of KGs [7, 9, 28]. Meanings and usages vary from specific technical proposals to more general descriptions. In this thesis, we use a more inclusive definition similar to the one proposed by Hogan et al. [38], where we view a KG as *a graph of data intended to capture the semantic connections within real-world knowledge, where nodes represent relevant entities and edges represent relations between these entities.* The type of graph may vary, i.e. it may be simple, directed, etc. A graph may contain knowledge over a broad range of domains, such as Wikidata [46], or be limited to a specific domain, such as DBpedia [30]. The concept of "knowledge" has been widely debated in epistemology [15, 43, 81, 34], but here we use it to mean descriptive knowledge, meaning facts that can be stated. Knowledge can be simple statements, such as "*Leo is a cat*", or quantified statements such as "*at least one cat is black*". Additional knowledge can

be inferred from KGs through inductive or deductive methods. For example, from a KG containing the information that "*Leo is a cat*" and "*cats are mammals*", one can deductively infer that "*Leo is a mammal*". If all cats mentioned in the knowledge graph like to eat fish, one can inductively infer "*cats like to eat fish*".

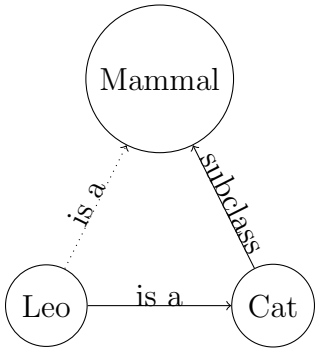


Figure 2.1: *Example of a KG, where the dotted line represents a relationship that can be deductively inferred.*

In this thesis, we loosely follow the Resource Description Framework (RDF) standard and view KGs as sets of semantic triples. RDF is a standard for the representation and exchange of graph data introduced by World Wide Web Consortium (W3C). Semantic triples are the data types used in the RDF data model. As the name suggests, a triple is a tuple of three elements. It has the form (subject, predicate, object) and can therefore represent statements about semantic data, for example, "*cats are mammals*" or "*Ann knows Bob*".

Listing 2.1: *Example of RDF triple set written in informal pseudocode.*

```
1 (Leo, is_a, cat)
2 (cat, is_a, mammal)
3 (Ann, is_a, person)
4 (Bob, is_a, person)
5 (Ann, knows, Bob)
6 (Ann, has_pet, Leo)
```

RDF statements express relationships between two web resources: the subject and the object, while the predicate encapsulates the nature of the relationship. The relationship is phrased in a directional way, so a set of RDF statements can also be viewed as a directed graph. Triple statements are represented by the graph, where the predicate in the triple denotes the edge going from the subject to the object, both of which are vertices.

With this type of data organisation, one can, for example, query for a list of all people who own cats in the dataset. Treating KGs as a set of triples is the data model used in this

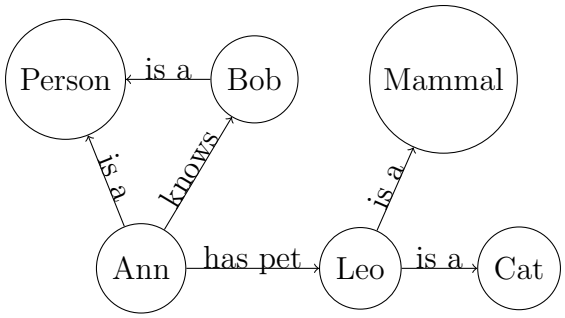


Figure 2.2: *Informal visualization of the KG consisting of the example triples from Listing 2.1*

thesis. As a note on use of terminology, when referring to a "fact" in a KG this is the same as a triple.

2.1.1 Knowledge base vs. knowledge graph

A knowledge-based system consists of two parts: a *knowledge base* containing the knowledge, and an *inference engine* that can be used to derive new facts from or answer questions about the knowledge base (KB) [3]. KBs often have both a terminological component and assertional component, respectively called the *TBox* and *ABox* [12]. The TBox represents knowledge about the structure of the domain, while the ABox has knowledge about specific instances. For example, the fact that *cats are mammals* would be a TBox statement, while *Leo is a cat* would be an ABox statement, as here we are making an assertion about the individual Leo.

KGs can be thought of as KBs with a graph-structured data model, but often lack the strict terminological schema. While the combination of a TBox and ABox in a KB allows the inference engine to derive a potentially infinite number of facts, the lack of rigid structure in a KG results in limited opportunities for reasoning in a KG. This problem has led to various approaches for KG completion, some of which are mentioned in chapter 5. The use of a rigid schema has been key to the success of relational databases [18]. However, it leads to a severe bottleneck when dealing with the integration of data from heterogeneous, semi-structured, and dynamic sources, such as Wikipedia. Therefore it is not a bug but a feature for large KGs to *not* incorporate the strict schema often used in KBs.

2.1.2 Incompleteness

Generally, KGs contain only true facts where consensus about truth is assumed. In reality, however, different and conflicting beliefs about truth are quite common, which can also be represented in KGs [58]. This thesis will not deal with conflicting beliefs about truth. As with all databases, there is seldom use in documenting all things that are not true. KGs are also generally incomplete, as it is impossible to store information about *all* entities and relationships in the world. By the closed world assumption (CWA), all facts not present in the KG are considered false. For example, by the KG in figure 2.2 the statement `(Ann, has_sibling, Bob)` is false under the CWA, as it is not present in the KG. So under the CWA Bob is not a sibling of Ann. The open world assumption (OWA) makes no such claims, and the validity of a triple not present in the KG is considered unknown. In the above example, under the OWA, Bob is therefore neither considered a sibling of Ann nor *not* a sibling of Ann.

In the context of KGs, the OWA is often more justified, as most large interesting KGs are far from complete. For example, the KG Wikidata5M does not contain information about the national bird of countries. This fact, of course, does not make the statement "*The kiwi is the national bird of New Zealand*" any less true. The information is merely not included in the KG. Another example is Freebase, the precursor to Wikidata, in which 70% of people listed had the place-of-birth attribute missing [80].

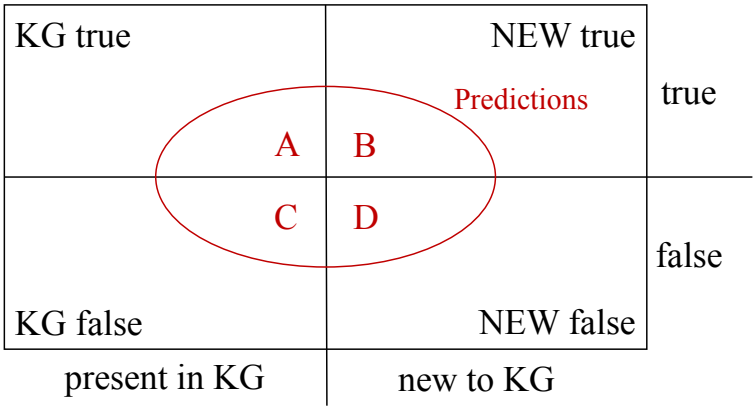


Figure 2.3: *KG prediction under incompleteness. Based on figure by Galárraga et al. [32].*

The OWA has a central problem regarding machine learning; it fails to provide counterexamples. Because missing facts are assumed to be neither true nor false, there are no

false facts to give as examples when mining rules or training an embedding model. We used figure 2.3, similar to the one used in the paper introducing AMIE [32], to aid the explanation of solutions to this problem. The figure represents all possible triples given the relation r and entities in a KG. These triples can be split into four types:

1. KG true: True facts in the KG
2. NEW true: True facts not in the KG
3. KG false: False facts *known to* the KG
4. NEW false: False *unknown to* the KG

When considering false facts, we use the words "*known*" and "*unknown*" as a KG generally does not contain false triples. We consider a given rule $B \Rightarrow r(x, y)$, where B denotes the body of the rule and $r(x, y)$ the consequent. In figure 2.3 the red circle denotes predictions made by this rule. These predictions can be true and in the KG (A), true and outside the KG (B), false and known to be false (C), or false and unknown to be false (D).

In the context of the predictions of the rule, we want to maximize B over D. Under the OWA, however, there are no false facts, so we do not know what B and D are. The authors of AMIE, therefore, introduced a new paradigm called the partial completeness assumption (PCA) [32]. It assumes that if $r(x, y) \in KGtrue$ for some entities x, y, then

$$\forall y' : r(x, y') \in KGtrue \cup NEWtrue \Rightarrow r(x, y') \in KGtrue,$$

where x is the *head* of the triple (the first variable in the binary predicate), and y is the *tail* of the rule (the second variable in the binary predicate). This assumption says that if the KG already has r-related information about the entity x, it contains *all* r-related information about x. For example, if a KG only contains `(Ann, has_sibling, Bob)` and no other sibling entries with Ann as the head, then Bob is Ann's only sibling by the PCA. Therefore, all other facts claiming that Ann has other siblings will be negative examples. However, if we ask what *parents* Ann has in a KG without information about Ann's parents, we cannot conclude that Ann has no parents. PCA is a more particular version of the broad *partial-closed world assumption* (PCWA), where the KG generally is treated under the OWA, but parts of it that are considered complete are treated under the CWA [57]. The PCA is more well-defined because it specifically states what parts of the KG should be treated under closed-world semantics.

2.2 Knowledge Graph Embeddings

Let $\mathcal{K}$ be a KG with triples of the form (h, r, t) with $r \in \mathbb{P}$ and $h, t \in \mathbb{E}$, where $\mathbb{P}$ and $\mathbb{E}$ are respectively the set of all relations and the set of all entities in $\mathcal{K}$. Entities and relations are embedded in a vector space. To simplify the explanation, we let the dimension of both entities and relations in the embedding space be d. Given $\mathcal{K}$ and d, a KGE seeks to represent all entities and relations in the continuous vector space of d dimensions. These representations are meant to capture the semantic information in the graph. An embedding that manages this can then be used to evaluate the probability of new triples being true in the context of the KG and identify false information in $\mathcal{K}$.

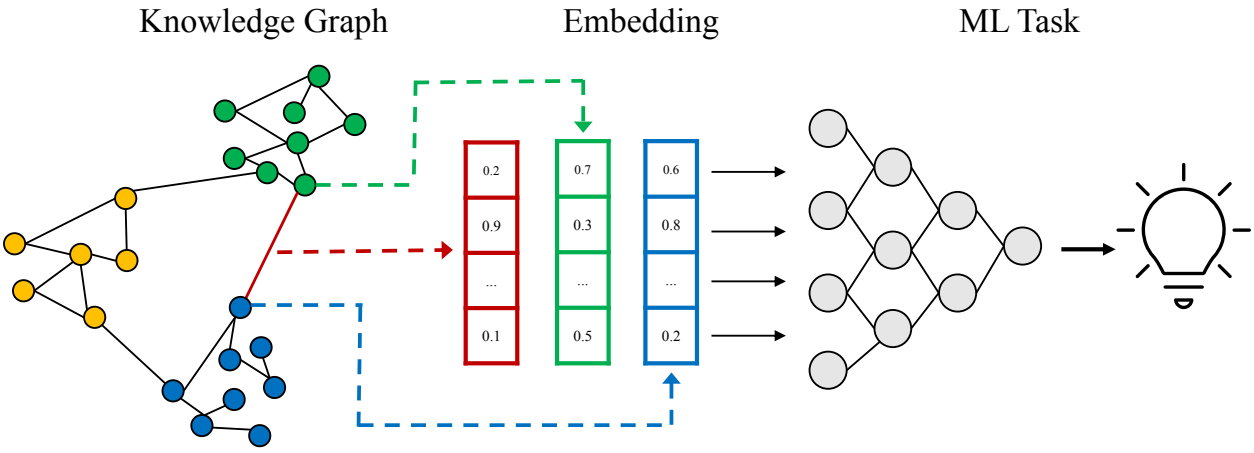


Figure 2.4: *KGE process. The embedding of a KG can be used for machine learning tasks. The figure is based on work by Edoardo Ramalli [27].*

The procedure for training KGE models is similar to any other statistical-based machine learning. The values of the embeddings are usually initialized as random values. These embeddings are continuously optimized through a training loop, which stops once the stop condition is met. A stop condition could be that a certain number of iterations have been completed, or when the model starts overfitting on the training set. For each triple in the training set, η negative counterexamples are generated by triple corruption. This is done by swapping out either the head h or tail t (not both) with some other $h', t' \in \mathbb{E}$ [10]. Both the original triple and the corrupted triples are added to the training batch. A *scoring function* is used to measure a triple's "goodness" or plausibility. The model should give a good score to triples from the KG and a bad score to the corrupted triples. By updating the embedding to optimize the scoring function, the model should by the end of training have meaningful

embeddings that can accurately evaluate unseen triples. In the training process this is done by minimizing the model's *loss function*.

2.2.1 Loss functions

A loss function is a function that assigns a "cost" to an event in the form of a real number. The loss function is minimized with some optimization algorithm, such as stochastic gradient descent. If the reader is curious, this algorithm is well-described in the Deep Learning book by Goodfellow et al. [33, p. 149]. In the context of KGE models, the loss function is used to update the embeddings [22]. There are many different types of loss functions, such as ranking losses [10], binary logistic regression [72], and multiclass log loss [41]. The next two subsections will briefly present the ranking loss and multiclass log loss versions used in the experiments.

Pairwise, margin-based ranking loss

Learning to rank is a machine learning task where a model is trained to rank a set of datapoints. In the *pairwise* approach, the problem is approximated to a binary classification problem, so the task becomes to determine which datapoint out of a pair is the better datapoint. The classifier takes as input two datapoints, one of higher rank x_+ and one of lower rank x_-, and has the goal to minimize the loss function that penalizes cases where x_- is given a higher rank. So the goal is to create a "gap" between positive and negative datapoints in the model's internal representation of the training data. In margin-based ranking loss, a minimum value is set for this distance. Once that distance is achieved, the model no longer needs to make updates regarding the pair of datapoints with adequate distance, thereby allowing training to be spent on learning other more challenging differences. The authors of TransE use this pairwise, margin-based ranking approach as their loss function [10]. Given a set of training triples $\mathcal{S}$, a scoring function f and a margin $\gamma > 0$, the pairwise, margin-based ranking loss is defined as:

$$\mathcal{L} = \sum_{(h,r,t)\in\mathcal{S}} \sum_{(h',r,t')\in\mathcal{S}'} [\gamma + f(h,r,t) - f(h',r,t')]$$

where $\mathcal{S}'$ is the set of corrupted triples. This loss function favours low scores for corrupted triples and high scores for non-corrupted triples. The aim is not to give negative triples a score below a certain value nor positive triples a score above a certain value, rather, the goal is to create distance between them. This loss function can be used across many KGE models, as it is the *scoring function* that differs across models. The scoring functions for TransE, DistMult, and ComplEx will be presented in Section 2.2.3.

Negative log-likelihood loss

This loss function was used in the paper introducing ComplEx [72]. Simply put, the function uses likelihood minimization to push the model toward the embedding that minimises the likelihood of loss. It is the *opposite* of maximum likelihood estimation, where one wants to maximise the likelihood of some datapoints given a set of parameters, hence the name *negative* log-likelihood. Since the likelihood of facts in a KG are independent, the likelihood of a set of triples is the same as the product of the likelihoods of each individual triple. With many datapoints, multiplying many small probabilities with each other quickly leads to unmanageable small numbers. This causes *underflow*, where a number is too small for a computer to be capable of storing it. To solve this problem, we take the log of the likelihoods so that products become sums. As log functions monotonically increase, the relative likelihood is maintained. For example, if $f(h,r,t) \geq f(h',r,t')$ then $log(f(h,r,t)) \geq log(f(h',r,t'))$, so the $\geq$ is maintained. Since the goal is to create a distance between positive and negative triples, only the relative difference between them needs to be maintained. Thus the negative log-likelihood loss is:

$$\mathcal{L} = \sum_{(h,r,t)\in\mathcal{S}\cup\mathcal{S}'} log(1 + exp(-y\, f(h,r,t)))$$

where $y \in [1,-1]$ is the truth value of the triple (true or false).

2.2.2 Performance indicators

Three different performance indicators are commonly used to evaluate the embedding quality of a model. While KGE models assign scores to triples, these scores are relative and cannot

be used to give an absolute estimate of how well a model evaluates a triple. Instead, the scores are compared with those of other triples so that the *rank* of a triple can be used as an approximation of absolute measurement. A small rank number indicates a good rank, while a high rank value indicates a bad rank. This can be somewhat confusing, so the reader is encouraged to take care when distinguishing between rank "number" and high/low rank, where a *low* rank number indicates a *high* rank and vice versa.

As the performance indicators are simple to calculate, they are suitable for measuring performance on a large scale. With $\mathcal{T}$ as the set of ranked triples and $\mathcal{S}$ as the set of true triples, this section defines three performance indexes.

Hits@K

Hits@K is the probability of finding the correct triple in the top K ranked triples. Usually, $k = 10$ or lower. This metric measures the model's ability to rank positive triples higher than the corrupted triples.

$$\text{Hits@K} = \frac{|\{t \in \mathcal{T} : t_{rank} < k, t \in \mathcal{S}\}|}{|\mathcal{T}|}$$

The larger the value, the better the performance of the model.

Mean rank

Mean rank (MR) is the average rank of all the ranks assigned to the triples in the test set. A small value indicates a good model because more triples from the knowledge graph have been given higher ranks, and accordingly, their rank numbers are smaller.

$$MR = \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} t_{rank}$$

MR has an advantage over Hits@K in being more sensitive to slight changes in the model because if the changes do not affect the top k ranked triples, then the Hits@K score remains unchanged.

Mean reciprocal rank

Mean reciprocal rank (MRR) is a measurement for the number of triples correctly ranked. If the triple with the best rank is a positive triple, 1 is added. If the triple with the second-best rank is positive, then $\frac{1}{2}$ is added, etc. If a ranked triple is negative, nothing is deducted, the fraction simply is not added to the MRR.

$$MRR = \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} t_{rank} : t \in \mathcal{S}$$

The larger the value, the better the model.

2.2.3 KG embedding model architectures

KGE models differ mainly in three aspects: (1) how entities and relations are represented, (2) the scoring function, and (3) how the embedding is optimized. The difficulty with embedding KGs is that there are many different entities and relationships, so the embedding model needs to be generic enough to capture all the different entities and relationships simultaneously. We now consider the three models used in this thesis.

TransE

In 2013, Mikolov et al. proposed a technique for natural language processing called *word2vec* [56, 55], which used a word-embedding algorithm that managed to capture some of the semantics in words. Words were represented as vectors, and the authors found that the embeddings had interesting qualities, such as

$$\overrightarrow{Queen} - \overrightarrow{King} \approx \overrightarrow{Woman} - \overrightarrow{Man}$$

where the words with an overhead arrow denote *word2vec*'s vector embedding of the word. Inspired by this, Bordes et al. applied the idea to embedding entities and relations in KGs and proposed the embedding model TransE [10]. The main motivation behind this approach was that "*hierarchical relationships are extremely common in KBs and translations are the natural transformations for representing them.*" [10]. This approach learns entity embeddings

and treats relations as translations in the entity embedding space. For a true triple, the tail entity should be very close to the head entity plus the relation translation in the embedding space.

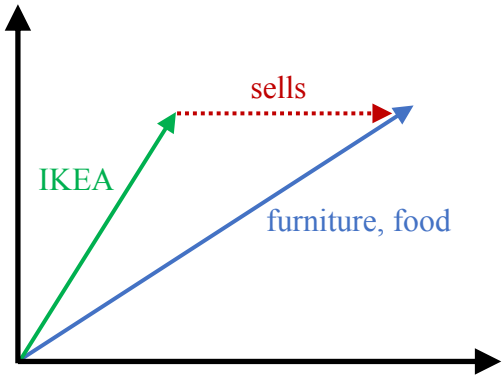


Figure 2.5: *Visualization of entity and relation embedding for TransE.*

If **(h, r, t)** denote an embedding of a head, relation and tail, then $\mathbf{h} + \mathbf{r} \approx \mathbf{t}$. The score function f_{TransE} is thus defined as the distance between $\mathbf{h}+\mathbf{r}$ and $\mathbf{t}$, using l_1 or l_2 to calculate distance:

$$f_{TransE}(h, r, t) = ||\mathbf{h} + \mathbf{r} - \mathbf{t}||_{l_1/l_2}$$

Let d denote the dimension of the embedding space. The model must learn an entity vector $\mathbf{e} \in \mathbb{R}^d$ for each entity and a translation vector $\mathbf{d}_r \in \mathbb{R}^d$ for each relationship. Let n_e and n_r be the number of unique entities and relations, respectively. The number of parameters needed to be learned for TransE is thus $\mathcal{O}(n_e d + n_r d)$.

TransE does have some limitations. For example, it cannot properly embed complex relations, meaning 1-N, N-1, or N-N relations [78, 50]. Imagine a complex relation such that $\forall i \in \{1, 2, ..., n\}$, $(h, r, t_i) \in \mathcal{S}$, where $\mathcal{S}$ is the set of correct triples. Following the TransE approach, $\mathbf{h} + \mathbf{r} \approx \mathbf{t}_i$, therefore $\mathbf{t}_1 \approx \mathbf{t}_2 \approx ... \approx \mathbf{t}_i$, even though the tail entities are not necessarily semantically similar. Taking an example depicted in figure 2.5, the store IKEA sells both furniture and food, causing TransE to give the semantically dissimilar concepts a similar entity embedding.

For a symmetric relation r_{sym}, TransE will assign high scores to both (h, r_{sym}, t) and (t, r_{sym}, h). This results in the embedding for r_{sym} to be pushed toward zero and the embedding of the entities h and t toward each other [77]. When $r_{sym} \approx 0$, then the relation

is treated as if it were reflexive. This is problematic when embedding symmetric relations, especially those that additionally are *not* reflexive. Other translation-based models, such as TransH [78], TranR [50] and TransD [40], have later been proposed to alleviate some of the limitations of TransE.

DistMult

DistMult is based on the tensor factorization-based model RESCAL [60]. A tensor is a multidimensional array representing the relationships between multiple sets of objects. The dimension of the array is the number of types of objects in the combinations. In the context of triples, there is the head entity, the relation, and the tail entity, so three sets of object types. In such models, triples in a KG are therefore transformed into a 3D binary tensor $\mathcal{X}$. We want the embeddings of the entities and relations to be such that one can mathematically combine them to obtain the tensor representing the KG. As seen in fig. 2.6, in the KG tensor each relation is represented by an $n \times n$ matrix, where n is the number of unique entities. The number of relations is denoted by m, so $\mathcal{X} \in \mathbb{R}^{n \times n \times m}$. An element $X_{ijk} = 1$ if there is a triple (e_i, r_j, e_k) in the graph, and $X_{ijk} = 0$ otherwise.

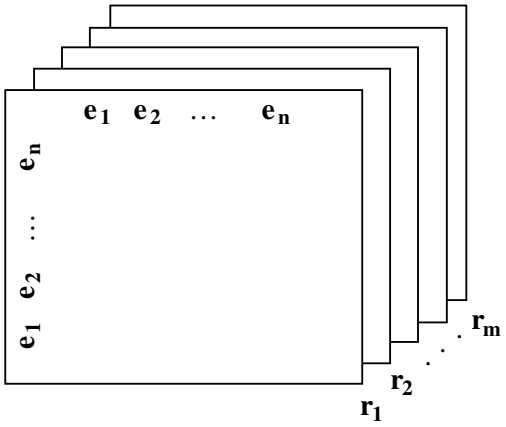


Figure 2.6: *A tensor model of a knowledge graph, inspired by the figure by Dai et al. [22].*

RESCAL uses rank-d factorization to obtain the latent semantics [60]. The *rank* of a matrix corresponds to its number of linearly independent columns. Given a matrix $\mathcal{Y} \in \mathbb{R}^{n \times m}$ of rank o, the rank factorization of $\mathcal{Y}$ is of the form $\mathcal{Y} = FG$ where $F \in \mathbb{R}^{m \times r}$ and $G \in \mathbb{R}^{r \times o}$. For $\mathcal{X}$ the rank factorization is applied so that tensor is "split" into slices, where each slice corresponds to the semantic embedding of a relation. Rank d is used because the relations

are embedded in $d \times d$-dimensional space. Formally we can define the pth slice in the KG tensor as:

$$\mathcal{X}_p \approx AR_pA^T, \text{ for } p = 1, 2, ..., m$$

where $A \in \mathbb{R}^{n \times d}$ is a matrix representing all entities and $R_k \in \mathbb{R}^{d \times d}$ is a matrix representing the pth relation. So the scoring function used in RESCAL is

$$f_{\text{RESCAL}}(h, r, t) = \mathbf{h}^\top \mathbf{M}_r \mathbf{t}$$

where $\mathbf{h}$, $\mathbf{t} \in \mathbb{R}^d$ are the embedding vectors of entities and $\mathbf{M}_r \in \mathbb{R}^{d \times d}$ is the semantic embedding of the relation. This method requires thus $\mathcal{O}(n_e d + n_r d^r)$ parameters. In order to lower this complexity, DistMult restricts $\mathbf{M}_r$ to be a diagonal matrix, meaning that all entries apart from those in the diagonal of the matrix are zero. After $\mathbf{M}_r = \text{diag}(\mathbf{r}), \mathbf{r} \in \mathbb{R}^d$. The scoring function is transformed to

$$f_{\text{DistMult}}(h, r, t) = \mathbf{h}^\top \text{diag}(\mathbf{r}) \mathbf{t}$$

Now only d parameters need to be learned per relationship, and the number of parameters to be learned for DistMult is $\mathcal{O}(n_e d + n_r d)$. The space required is also $\mathcal{O}(n_e d + n_r d)$.

A main problem with this approach is that $f_{\text{DistMult}}(h, r, t) = f_{\text{DistMult}}(t, r, h)$, so DistMult cannot embed the asymmetry of a relation. This issue is addressed by later models, such as ComplEx [72] and SimplE [42].

ComplEx

ComplEx extends DistMult with complex-valued embeddings, thereby allowing it to distinguish between symmetric and asymmetric facts [72]. This was achieved without increasing parameter or memory complexity. The imaginary and real part of the embeddings represent the "direction" of the relation, and thus asymmetric qualities can be expressed in the embedding. The scoring function for ComplEx is quite similar to that of DistMult:

$$f_{\text{ComplEx}}(h, r, t) = \mathbf{Re}(\mathbf{h}^\top \text{diag}(\mathbf{r}) \bar{\mathbf{t}})$$

where $\bar{\mathbf{t}}$ represents the complex conjugate of $\mathbf{t}$ and $\mathbf{Re}$ means we are taking the real values of the score. With such a scoring function, triples with asymmetric relations are able to obtain different scores depending on the ordering of entities.

2.3 Rule-based machine learning

Rule-based machine learning aims to create rules that make new true predictions going beyond the data on which the rule was applied [79]. Other areas of statistical machine learning often focus on training a single model that can be applied to make a broad range of predictions. Conceptually, the end result of rule-based machine learning is similar to a rule-based system. Rule-based systems are often hand-crafted and require a knowledge expert to be curated, while rule-based machine learning requires no knowledge expert and rules are automatically created by the learning algorithm.

Classically, a rule comprises of a condition and consequent, or a so-called "if-then" statement.

IF *'the condition is met'* **THEN** *'the consequent holds'*

The rule's condition specifies attributes in the data on which the rule will be applied. If these attributes are present in this data, the condition is met. Once this happens, the attributes in the consequent should necessarily also be met. We define an *atom* as a triple in which the head and/or tail are variables. A *Horn rule* is a rule where the consequent of a rule is a single atom, while the body is a set of atoms. We denote a Horn rule by $B \Rightarrow r(x, y)$, where B is the antecedent and $r(x, y)$ the consequent. Two examples of such rules are 2.1 and 2.2, where the first is satisfied by the KG in Example 1 below, while the latter is not.

$$has\_sibling(x, y) \Rightarrow has\_sibling(y, x) \tag{2.1}$$

$$has\_sibling(x, y), has\_parent(y, z) \Rightarrow has\_parent(x, z) \tag{2.2}$$

Example 1. A simple KG.

```
1 (Ann, has_parent, Carol)
2 (Carol, has_child, Ann)
3 (Carol, has_child, Bob)
4 (Ann, has_sibling, Bob)
5 (Bob, has_sibling, Ann)
```

Consider the two rules and the example KG with numbered triples given above. Under this KG the antecedent of the first rule is satisfied by both triple 4 and 5 listed in the example. The resulting consequent of rule 2.1 is also present in the KG. So the predicate *has_sibling* is reflexive in this KG. The antecedent of the second rule is satisfied by triples 1 and 5 in the KG, but the resulting consequent would then be $has\_parent(Bob, Carol)$. Since `(BoB, has_parent, Carol)` is not present in the KG, rule 2.2 does not hold. If we know that this rule is reliable, we could for example extend the incomplete KG with this triple.

Within rule-based machine learning there are many different approaches, including learning classifier systems [68] and association rule mining [2]. The latter is the approach used in this thesis in the form of the AMIE3 rule mining algorithm. Both methods aim to create a set of rules to act as a model for a set of data. The association rule mining approach will now be explained further.

2.3.1 Association rule mining

Association rules [2] are types of "if-then" statements that describe frequent associations between items in a dataset containing *transactions*. A transaction can be thought of as a set of related items. Association rule mining was initially proposed as a new method for finding relationships between store sales items. The idea of mining association rules over transactions has successfully been applied to many other scenarios, such as within health informatics and recommender systems [5, 48]. In the context of convenience store sales, each transaction can be thought of as a set of items that a customer has purchased. The rules are of the form $\{Sugar, Flour, Eggs\} \Rightarrow Butter$, meaning that a person who bought sugar, flour and eggs is likely to also purchase butter. The antecedent is some set of items in the dataset, while the consequent is an item often found in combination with the antecedent in the dataset. So sugar, flour, and eggs can be thought of as "associated with" butter. These original association rules are also not Horn rules over binary predicates as in AMIE3. However, it is cited as a main inspiration for the AMIE3 algorithm used for experiments in

this thesis. AMIE3 also limits rules to be in Horn form, meaning that the consequent can contain at most one predicate.

2.3.2 Significance and quality measurement

The goal is to find formal rules that make true predictions beyond the explicit information in the KG. If we look back at fig. 2.3 this means we want to maximise B and minimise D. For this, we need to know what those areas are, and this is where the problem of missing negative examples from section 2.1.2 occurs as the area **KG false** is empty. While it is hard to define *quality* when evaluating rules, some indicators can be used. We now consider different measures to address this problem.

Support

The *support* of a rule is the number of correct predictions. If we recall figure fig. 2.3 with KG prediction under incompleteness, the support of a rule $B \Rightarrow r(x, y)$ is area A. What counts as an instance of a correct prediction can vary. The authors of AMIE point out that if one chooses the number of instantiations of the rule, then the measure becomes non-monotonic [44]. They give an example with the rule:

$$married\_to(x, y) \Rightarrow married\_to(y, x)$$

where if $has\_gender(x, male)$ is added to the body, then the number of instantiations in the KG can only decrease because there are stricter requirements for what can be in the body of the rule. If on the other hand $has\_friend(x, z)$ is added to the body, then the number of instantiations may increase, because for every x there can be many z possibly resulting in many more instantiations of the rule. In order to preserve monotonicity and only a single measure of support for each rule, the authors of AMIE define support of a rule R to be the number of true predictions p in the KG $\mathcal{K}$ that the rule makes:

$$support(R) := |\{p : (\mathcal{K} \wedge R \models p) \wedge p \in \mathcal{K}\}|$$

Now the support of a rule decreases monotonically if more atoms are added to the body, and two equivalent rules cannot have different values for support.

Head coverage

Support is not an absolute measurement, but rather a relative measurement. It requires the complete size of the KG for the values to have meaning. This is countered with a proportional version of support called *head coverage*. Given a rule $B \Rightarrow r(x, y)$, the head coverage is the ratio of instantiations of the head of the rule $r(x, y)$ that are predicted by the rule:

$$hc(B \Rightarrow r(x, y)) = \frac{support(B \Rightarrow r(x, y))}{|\{(x, y) : r(x, y) \in \mathcal{K}}|$$

Head coverage is fully correlated with support and hence is also monotonic. We now have an absolute measurement of significance, but this is not a measure of the quality of a rule, only relevance. For this, we need confidence measures.

Standard Confidence

Confidence is the proportion of a rule's true predictions out of all its predictions. In order to determine if a new fact is true or false, one must make assumptions about the facts missing from the KG. The *standard confidence* adopts the CWA and labels all facts not already present in the KG as false. Thus, the standard confidence of a rule R in a KG $\mathcal{K}$ is the ratio of its predictions that are in the KG:

$$conf(R) := \frac{support(R)}{support(R) + |\{p : (\mathcal{K} \wedge R \models p) \wedge p \notin \mathcal{K}\}|}$$

If we take the KG from Example 1 and the rule $has\_child(y, x) \Rightarrow has\_parent(x, y)$, then the rule implies two datapoints: `(Ann, has_parent, Carol)` and `(Bob, has_parent, Carol)`. The first triple is in the KG, while the second is not, therefore the rule has predicted one correct and one incorrect triple, thus the standard confidence is 0.5.

If one wants to mine rules that only describe the data at hand, and has a relatively complete KG, this is a good measure of confidence. If the aim, on the other hand, is to mine rules that predict new facts, then standard confidence is not a good measurement as it penalizes rules that make predictions outside of the current knowledge.

PCA Confidence

This measurement for confidence differs from standard confidence in its definition of what is a false prediction. Under standard confidence, any triple outside the KG is a false prediction, while this confidence metric adopts PCA, introduced in section 2.1.2. The definition of a false prediction is thus more conservative, so not all triples outside the KG are considered false. The PCA confidence of the rule R is the fraction of its predictions that are not labeled as false by the PCA [32]:

$$pca - conf(B \Rightarrow r(x,y)) := \frac{support(B \Rightarrow r(x,y))}{|\{(x,y) : \exists y' : B \wedge r(x,y')\}|}$$

Again, given the KG from Example 1 and the rule $has\_child(x,y) \Rightarrow has\_parent(y,x)$, the rule implies two datapoints: `(Ann, has_parent, Carol)` and `(Bob, has_parent, Carol)`. The first is in the KG, therefore correct, while the latter is outside the KG. As there are no `has_parent` triples with `Bob` as the subject in the KG, we cannot under the PCA assume that `(Bob, has_parent, Carol)` is a false triple. Hence, the PCA confidence of the rule is 1.0, in contrast to the standard confidence of 0.5.

This is the standard PCA confidence measure, but it could, of course, also be calculated on the head entity of the consequent instead of the tail, i.e.:

$$pca - conf_h(B \Rightarrow r(x,y)) := \frac{support(B \Rightarrow r(x,y))}{|\{(x,y) : \exists x' : B \wedge r(x',y)\}|}$$

Note that the PCA confidence calculated on the *tail* of the consequent is the metric used throughout this thesis to indicate the quality of rules.

GPRO and GRANK confidence

Ebisu et al. [26] showed that PCA confidence might underestimate the likelihood of a predicted triple due to non-unique mappings of variables in rules. Consider, for example, the rule $has\_parent(x,z) \wedge has\_child(z,y) \Rightarrow has\_sibling(x,y)$, and the KG from Example 1. By mapping both x and y to the entity `Ann`, the body of this rule can be instantiated

with the triples `(Ann, has_parent, Carol)` and `(Carol, has_child, Ann)`, resulting in the incorrect fact that Ann is her own sibling. The refinement of PCA confidence, called graph pattern probability model (GPro) confidence, excludes instances such as these that do not map variables to unique entities when computing confidence. As with PCA confidence, GPro confidence can be calculated on both the head and tail of the consequent.

Ebisu et al. also proposed a refinement of GPro confidence, called graph pattern entity ranking model (GRank) confidence [26], which takes into account the number of instantiations of the body of the rule for a given instantiation of the head atom. This metric is more sensitive to the number of facts per relation.

2.3.3 AMIE, AMIE+ and AMIE3

The experiments of this thesis required a rule mining algorithm capable of mining meaningful rules while maintaining acceptable runtime. The rule mining component of the experiment would be treated as a black box that only took a KG as input and produced a set of rules as output. The AMIE-algorithms do precisely this, and require no hyperparameter optimization, training, nor additional input.

AMIE is a rule mining system introduced in 2013 by Galárraga et al. [32], which was improved upon in 2015 with the release of AMIE+ [31] and in 2020 with AMIE3 [44]. This section will summarise the original rule mining algorithm, present the improvements made with AMIE+ and AMIE3, and finally justify the eventual choice of AMIE3.

The AMIE rule mining algorithm

The algorithm employs a type of breadth-first search when exploring the search space for possible rules. It starts with the empty rule, and extends it. It is extended in a way that if this breadth-first extension continues, the entire search space can be generated. Of course, this is not desired, so AMIE employs a number of methods to limit the search for rules. One can think of this as methods of pruning the breadth-first search tree. First we will look at how rules are extended and then how the search space is pruned.

A rule is treated as a sequence of atoms, where the first atom is the head, and the rest make up the body. It is considered *closed*, if every variable is shared by at least two predicates. These rules can also be thought of as *closed-path* rules, since the sequence of relations in the body of the rule form a path from the subject argument to the object argument of the head atom in the rule. The rule $has\_child(y, x) \Rightarrow has\_parent(x, y)$ is closed. If we were to add an atom to the body with a *fresh* (new and unused) variable, a resulting rule $has\_sibling(z, x) \wedge has\_child(y, x) \Rightarrow has\_parent(x, y)$, would be open. The algorithm only outputs closed rules, ensuring that rules do not contain unrelated atoms or variables. When traversing the search space, AMIE extends rules by adding one of three different atom types:

1. *Dangling atom*: An atom with a fresh variable and a shared variable is added. A shared variable is a variable that occurs in some other atom of the rule.
2. *Instantiated atom*: An atom with an entity and an argument (variable or entity) shared with the rule.
3. *Closing atom*: An atom with both arguments shared with the rule.

By only adding atoms to rules, the extension is monotonic in the sense that atoms added by one operation cannot be modified by a later operation. In the algorithm new rules are added to a queue. If, however, a generated rule is a duplicate of one already in the queue, it is not enqueued. Because the length of the rules dequeued increases monotonically, one can be sure that any potential duplicates still are in the queue – when a rule with n atoms is dequeued, no rule with $n+1$ atoms has ever been dequeued. Since equal rules have the same head coverage and PCA confidence, this limits the search space considerably when checking for duplicates before enqueuing a new rule.

The head coverage of possible rules must be at least 0.01, meaning rules that correctly predict less than 1% of the head relations are deemed insignificant. As mentioned in section 2.3.2 *head coverage* decreases monotonically as atoms are added to a rule. This allows the algorithm to safely discard and not expand upon rules that score below the head coverage threshold. With the default setting a rule also needs support ≥ 100.

For more details about the implementation details of AMIE, please refer to the original paper, *AMIE: association rule mining under incomplete evidence in ontological knowledge*

bases [31]. Note that the authors of AMIE present results with rules mined *without* constants unless it is explicitly stated. This is also the case for the AMIE+ and AMIE3 publications. The AMIE-algorithms are capable of mining rules with instantiated atoms, but at the expense of increased time required to run.

AMIE+ and AMIE3

While the original AMIE algorithm had some intelligent methods for limiting the search space, it spent 3.62min on a server with 48GB RAM and 8 CPUs [32] when mining rules with two atoms and no constants from YAGO2, which has around 1 million facts [37]. In recent years KGs have only increased in size (YAGO4 has 2 billion facts [66]), so the improvements made in AMIE+ and AMIE3 address this need for improved performance. AMIE+ includes new pruning techniques, such as the fact that rules with 100% confidence cannot be improved upon, and hence do not need to be expanded. Computing the PCA confidence of rules is a quite expensive part of the algorithm, so AMIE+ introduces a method for approximating the confidence score. This approximation has a 4% error rate, but shortens the runtime considerably. AMIE3 uses more sophisticated methods for calculating support and PCA confidence, and does not need to resort to approximations, while simultaneously improving runtime. It also employs a technique called *lazy evaluation*, which rests on the idea that if a rule is bad, there is no need to spend resources calculating exactly how bad it is.

Depending on the intended application of the mined rules, different metrics are more appropriate than others. For example, standard confidence is more appropriate if one assumes that the information in the KG is complete and one wants rules that model the data as closely as possible. If the intended use of the rules, on the other hand, is to predict new facts, then PCA, GPro or GRank confidence are more appropriate. In the release of AMIE3 all these metrics are implemented, however their implementation of GPro or GRank confidence was still in the experimental stages, therefore PCA confidence was used in this thesis.

The role of AMIE3 in experiments

The rule miner required for the experiments in this master project had to have reasonable running time and have the ability to mine rules of acceptable quality. The PCA confidence

measure, while not perfect, was deemed an adequate indication of quality, and therefore the AMIE-algorithms that employed this measure seemed appropriate for the task. The AMIE3 algorithm was the best of the three, and it performed well against other recent rule mining algorithms, such as Ontological Pathfinding [17] and RudiK [65]. Furthermore, the authors of AMIE3 had made their publication publicly available on GitHub with clear instructions on how to use it. For these reasons the AMIE3 algorithm was chosen as the rule miner for the experiments of the present project.

Chapter 3

Rule mining on extended knowledge graphs

In this chapter, we first briefly summarize the KG extension and mining process, before exploring each component separately.

The process starts with a KG. From this KG, we want to first mine rules, then extend it with new plausible facts, and lastly mine rules again on the extended KG. Candidates for additions to the KG are generated according to strategies that attempt to maximise the candidates' plausibility. Once this set of candidate triples has been generated, the best of these are added to the KG. While there is no direct way to give an absolute score to these triples, a KGE trained on the original KG can rank the candidate triples against a set of corrupted triples. With the candidates ranked, one can accept all candidates above some appropriately chosen cutoff and add them to the KG. Then rules can be mined and evaluated on both the original and extended KG, and results can be compared.

This chapter also looks at the datasets used in the experiments and the results from the model selection process. The goal of the model selection process is to produce KGE models of acceptable quality for the data extension pipeline. Since the models are merely components of the process and not experimental results, this chapter presents the model selection outcomes.

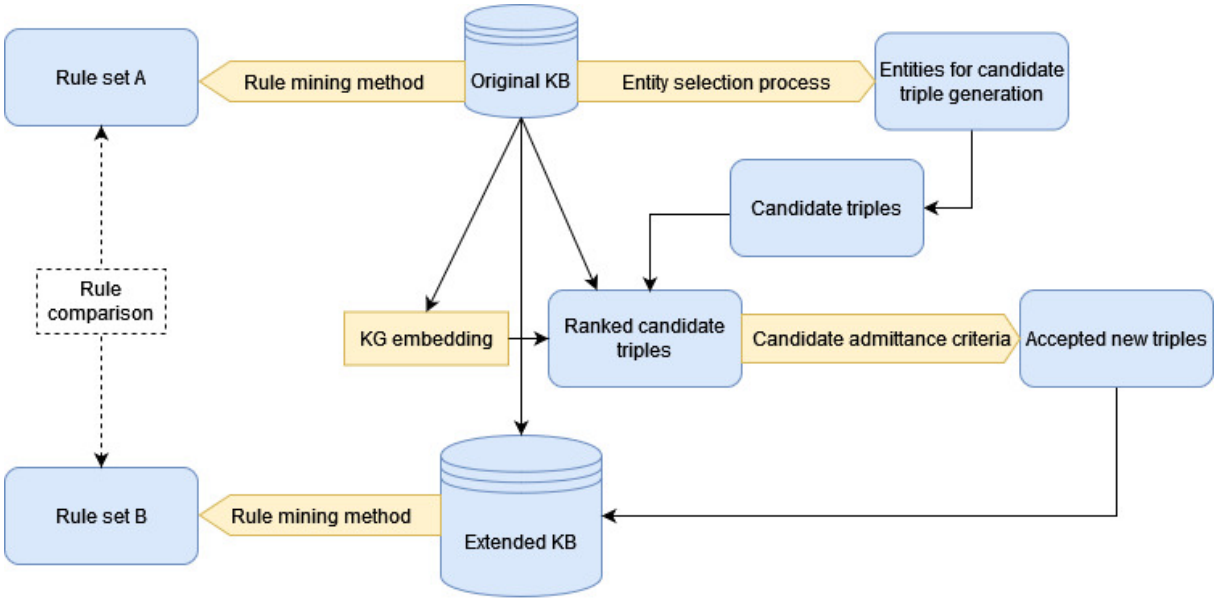


Figure 3.1: *Diagram representing KG extension and rule mining pipeline.*

3.1 KG datasets

While there are many large KGs publicly available, they are often quite complex in the number of different relations and entities used. Since we are mining rules over relations, the range of different relations was kept small. By restricting the number of relations, the resulting rules will not be overly diverse, and there will be many more candidate triples per relation when generating new plausible triples. Two different datasets were used to conduct the experiments, both of which were limited to contain only six different types of relations. We discuss them next as we look at the KGs individually.

3.1.1 Wikidata5M-family KG

Wikidata5M is a KG dataset containing over 20 million triples with over 800 types of relations. It combines information from the Wikidata KG with Wikipedia pages [75]. Wikidata5M uses the same identifier system as Wikidata, where each entity and relation is assigned a unique integer ID. The IDs of entities are prefixed with the letter `Q` and those of relations with `P`. For example, the following line

```
Q146    P279    Q39201
```

Listing 3.1: *Python dictionary converting family predicate IDs to their names.*

```
family_predicates_dict = {
    'P40': 'child',
    'P22' : 'father',
    'P25' : 'mother',
    'P26' : 'spouse',
    'P1038' : 'relative',
    'P3373' : 'sibling',
}
```

corresponds to <`house cat, subclass of, pet`>, where each entity has a corresponding Wikipedia page. This dataset was chosen due to its size and because it contained family-related information. Rules about family structure are well-known and easy to comprehend. For example, the simple rule $parent(a, b) \Rightarrow child(b, a)$ would be implicitly represented in the KG.

All triples that did not use one of the six selected family predicates were filtered out, and the IDs were converted to meaningful names so that the resulting rules mined would be easily readable. The python dictionary in listing 3.1 shows the chosen predicates. A few other family predicates were considered, but did not have enough corresponding data points or were too similar to other selected predicates. Common family-related predicates such as `grandchild` or `sister` are also not currently used in Wikidata. From a discussion in the Wikidata community, it seems that all properties considered redundant were removed [73]. Pykeen's distribution of the dataset was used for the experiments [4]. The resulting subset of Wikidata5M contained around 250 000 triples and will henceforth be referred to as the *family KG*.

3.1.2 WN18RR

WN18RR is a smaller KG with 93 003 triples and only 11 relations [23]. It is an improved version of the WN18 dataset, where it was found that there was information leakage between the training and test set of WN18. Many test triples could be identified simply by inverting triples in the training set [71]. For example, the relation `has_child` is the inverse relation of `has_parent`, meaning that if the triple `(Ann, has_parent, Carol)` is in the training set, then one can infer that `(Carol, has_child, Ann)` likely is in the test set, thereby the test data is no longer consideres "unseen" not a fair data set to evaluate the model on. WN18RR addresses this issue.

WN18RR contains triples scraped from WordNet, a lexical database for English [29]. In WordNet nouns, verbs, adjectives, and adverbs are grouped into sets of semantic synonyms called *synsets* which express a distinct concept. For example, {`man`} and {`adult male`} belong to the same synset. Synsets are connected to other synsets by semantic relations. The most commonly used relation in WN18RR is `hypernym`. A synset `X` is a hypernym of synset `Y`, if every `Y` is a kind of `X`. For example, the triple

`02121808 hypernym 02121620`

represents the information that the synset `cat` is a hypernym of the synset `house cat`. Or more simply phrased: *All house cats are cats.*

	Relation	Frequency
Included	hypernym	36873
	derivationally related form	31865
	member meronym	7912
	has part	5131
	synset domain topic of	3328
	instance hypernym	3118
Excluded	also see	1396
	verb group	1220
	member of domain region	981
	member of domain usage	673
	similar to	86

Table 3.1: *Frequency of relations in WN18RR KG and whether they were included in the final KG.*

Though it is not vital for the reader to understand the semantics behind the relations in this dataset, it may be more rewarding to understand the terms when later reading rules mined from the WN18RR KG. The relation `hypernym` was described above, so now the remaining five relations are succinctly explained.

- `derivationally related form`
  A concept A derives from another concept B. `perfectly` is the derivationally related form of `perfect`.
- `member meronym`
  A concept A is a member of a concept B. `class` is a member meronym of `student`.
- `has part`
  A whole concept A has a part B. `cat` has part `tail`.

- `synset domain topic of`
  A concept A is the scientific field which concept B belongs in. `computer science` is the synset domain topic of `deep learning`.
- `instance hypernym`
  It denotes the type of an instance. For example, `Bergen` has instance hypernym `city`.

WN18RR was chosen due to the few number of relations in it, and with a KG already containing few relations, most of the data could be included. By taking all triples containing one of the six most frequent relations, in total 88227 datapoints, 95% of WN18RR was used. This was intended to produce a more complete KG. The dataset was loaded using AmpliGraph [19].

3.2 KGE model architectures

There are many publicly available knowledge graph embedding libraries [13, 14, 4]. The library AmpliGraph was chosen due to its thorough documentation and because it is an open source library based on Tensorflow, a well-known library for the development of machine learning models [1]. The library provides many different KGE models, performance metrics, and KG datasets. Three different KGE methods were selected: TransE, DistMult, and ComplEx.

TransE was selected because it is one of the simplest and most intuitive KGE models. Due to the limited number of relations in the KGs, this architecture may be sufficient for producing a good embedding. However, as mentioned in section 2.2.3, TransE struggles with learning complex and symmetric relations, both of which are present in the chosen KGs, therefore it may not be adequate. In the family KG *spouse* and *relative* are symmetric relationships. *child* is an example of a complex relation, as a person can have many children. In WN18RR *derivationally related form* is symmetric, but not reflexive.

DistMult was selected because it is in a different family of embedding vectors, namely tensor-factorization based models. Recalling its description from section 2.2.3, DistMult is one of the least computationally-intensive models due to its use of diagonal matrices, which made it an obvious choice for the experiments of this thesis.

ComplEx has the same runtime as DistMult, and was therefore an attractive candidate. The embedding model was chosen due to this and also because it addresses DistMult's shortcomings in embedding antisymmetric relations. The relations *parent* and *child* in the family KG are antisymmetric, so it would be interesting to compare the results from these two models.

AmpliGraph also provides a baseline model, to which the three selected models were compared to. The baseline model, called RandomBaseline, assigns a pseudo-random score to each triples it is asked to evaluate. When it comes to extending the KG this would essentially have the same effect as adding noise to the data.

3.3 Model selection

Each KGE model has a number of hyperparameters that can be optimised. As the search space grows, it has been shown that random search is more optimal than grid search, where each combination needs to be tested [8]. Due to limited computational resources, this approach to hyperparameter optimisation was selected. It is not an optimal approach, but serves as a practical and effective solution that measures well against more sophisticated methods such as Baysian optimisation [47].

Hyperparameter	**Values**
Batches count	50, 100
Epocs	50, 100
Embedding dimension	50, 100, 200
η (negative sampes @ rate)	5, 10, 15
Loss function	pairwise, nll
Pairwise loss margin	0.5, 1, 2

Table 3.2: *Hyperparameter values to search through during model selection.*

The AmpliGraph documentation was the primary inspiration when deciding which hyperparameters to focus on. It was, for example, stated in the documentation that they received the best results with the adam optimizer, therefore other optimisers were not considered in the hyperparameter search [20]. For each hyperparameter combination, a learning rate randomly chosen in the range of 0.0001 - 0.01 was selected, yet another design choice borrowed from AmpliGraph.

PARAM.	WN18RR			Family		
	TransE	**DistMult**	**ComplEx**	**TransE**	**DistMult**	**ComplEx**
Batch size	50	50	100	50	100	50
Epocs	50	100	100	50	50	100
Emb. dim.	50	200	100	50	200	200
η	15	5	10	15	10	15
Loss func.	nll	nll	pairwise	nll	nll	nll
P. l. margin	-	-	0.5	-	-	-

Table 3.3: *Hyperparameter selection results. The hyperparameter at the bottom of the table, pairwise loss margin, was only applicable for those models that used a pairwise loss function, hence the missing values.*

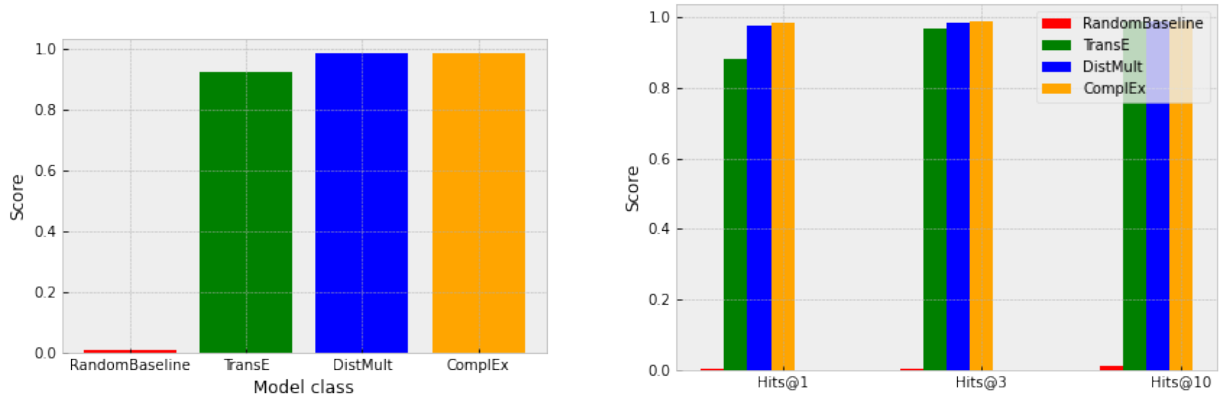
For the implementation of model selection, AmpliGraph's `select_best_mode_ranking` was used. It is a model selection routine that allows for both grid and random search. At the end of each model selection process, the final model for each embedding type is retrained on the concatenation of the train and validation set, before it is eventually evaluated on the test set. The `eta` hyperparameter denotes the number of negative examples generated at training for each positive example, a process described in section 2.2. The hyperparameters for the final selected models can be seen in table 3.3.

3.3.1 Model selection results

The final model of all three embedding architectures had good results on the WN18RR dataset, and even better results on the family KG. Each model had an MRR score above 0.9 on the family KG and around 0.6 on the WN18RR. TransE seemed to perform slightly worse than DistMult and ComplEx on the family KG. On the WN18RR dataset TransE had a lower hits@1 score, while it outperformed both DistMult and ComplEx in both hits@3 and hits@10.

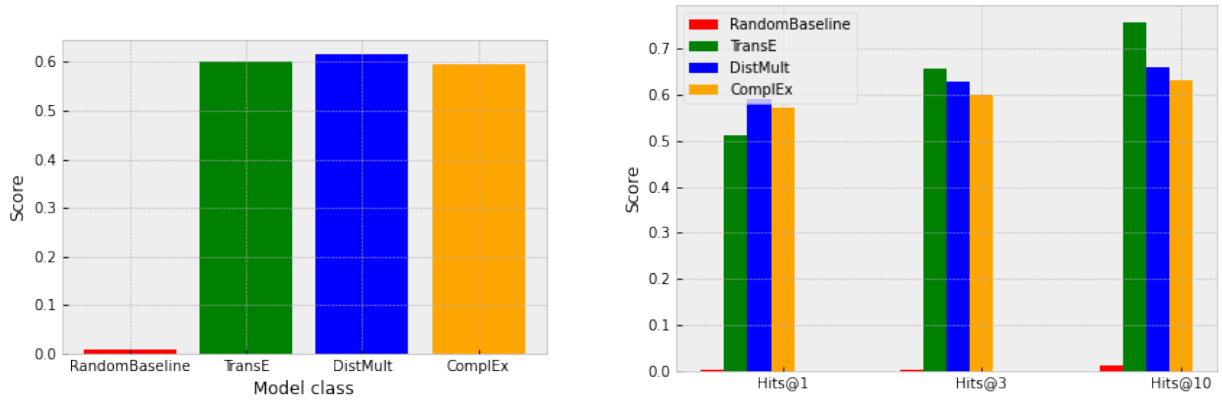
Dataset	WN18RR KG					Family KG				
Model	MR	MRR	Hits@K			MR	MRR	Hits@K		
			1	3	10			1	3	10
Random	495.32	0.01	0.00	0.00	0.01	498.72	0.00	0.00	0.00	0.10
TransE	**34.29**	0.60	0.51	**0.66**	**0.76**	**2.59**	0.93	0.88	0.97	**0.99**
DistMult	152.37	**0.62**	**0.59**	0.63	0.66	7.45	0.98	**0.99**	**0.99**	**0.99**
ComplEx	139.36	0.59	0.57	0.60	0.63	4.64	**0.99**	0.98	**0.99**	**0.99**

Table 3.4: *Results of selected models evaluated on test set, with the best results per column marked with bold font.*



(a) *MRR scores* **(b)** *Hits@k scores*

Figure 3.2: *KGE test performance results for family KG.*



(a) *MRR scores* **(b)** *Hits@k scores*

Figure 3.3: *KGE test performance results for WN18RR.*

3.4 KG extension

The goal of KG extension is to add potentially true statements to the original KG. AmpliGraph does have an implementation for this, called `discover_facts`, but due to computational limitations[1] it could not be used. However, the implementation used in our study follows the same general strategy used in `discover_facts`. It consists of two main components:

1. Candidate triple generation
2. Ranking of generated candidates

With a set of ranked candidate triples, all that remains is to decide the minimum rank for a candidate to be admitted to the KG.

3.4.1 Candidate triple generation

Both in the family KG and in WN18RR the number of potential facts is enormous, and so in order to avoid having to evaluate all of them, several strategies can be used to select plausible facts. In AmpliGraph's implementation of plausible candidate generation, they make the assumption that densely connected entities are less likely to have missing true statements. In a KGE tutorial at the European Conference on Artificial Intelligence in 2020, members of the AmpliGraph team claim that this assumption has been true for their empirical evaluations, but is not necessarily true for all datasets [21]. As stated, AmpliGraph has implemented different strategies, many of which have to do with graph clustering, but these were too computationally intensive to be used. Four simpler strategies for entity selection were instead used:

- Random selection of entities.
- Selecting the most frequent entities.
- Selecting the least frequent entities.
- Probabilistic selection based on the frequency of the entities in the dataset, where the least frequent entities are most likely to be selected.

KG			Entity	Freq.
Ann	*sibling*	*Bob*	Ann	4
Ann	*sibling*	*Carl*	Bob	3
Ann	*friend*	*Carl*	Carl	3
Bob	*friend*	*Carl*	Dave	2
Bob	*friend*	*Dave*	Eve	2
Ann	*friend*	*Eve*	Felix	1
Felix	*friend*	*Gina*	Gina	1
Dave	*friend*	*Eve*		

Table 3.5: *Example KG with frequency table of entities.*

If we look at listing 3.5 as an example KG and need to select three entities for candidate generate, then with the

- *most frequent* strategy the entity set would be {*Ann, Bob, Carl*},
- *least frequent* strategy the entity set would be {*Dave, Felix, Gina*},
- *probabilistic* strategy the entity set could be {*Gina, Eve, Felix*},

and any three candidates would be selected with the *random* strategy. Note that selection is without replacement. Once the entities were selected, all possible triples were generated with them and the six relations. Then, all the triples already included in the original KG were excluded from the resulting candidate triples. If one were to pick the *most frequent* strategy, then the resulting set of candidate triples would be that depicted in table 3.6. Of course, the more candidate triples there are to choose from, the better the extension will be because then there is a higher chance of plausible facts being suggested. Again, computational limitations restricted this. After trying different candidate set sizes, it was eventually decided that selecting 1000 entities for candidate triple generation was a feasible number. This meant that $1000 \times 6 \times 1000 = 6 \times 10^6$ triples (minus those that already appeared in the KG) were considered for each KG extension.

3.4.2 Candidate triple ranking

As discussed in Section 2.2.2, it is not easy to set an absolute score to a triple with a KGE. Therefore candidate triples are instead ranked against corrupted triples. An accurate KGE

[1] `discover_facts` uses *all* entities in the KG to generate counterexamples. In large KGs like the ones used in our experiments, this is not feasible.

Candiate Triples					
Ann	*sibling*	*Ann*	*Ann*	*friend*	*Ann*
Ann	*sibling*	*Bob*	*Ann*	*friend*	*Bob*
Ann	*sibling*	*Carl*	*Ann*	*friend*	*Carl*
Bob	*sibling*	*Ann*	*Bob*	*friend*	*Ann*
Bob	*sibling*	*Bob*	*Bob*	*friend*	*Bob*
Bob	*sibling*	*Carl*	*Bob*	*friend*	*Carl*
Carl	*sibling*	*Ann*	*Carl*	*friend*	*Ann*
Carl	*sibling*	*Bob*	*Carl*	*friend*	*Bob*
Carl	*sibling*	*Carl*	*Carl*	*friend*	*Carl*

Table 3.6: *Candidate triples for the KG in listing 3.5, using entity selection method most frequent, and entity set size 3. Triples in red are already present in the KG and are removed from the candidates set.*

model will rank well-suited candidates higher than the corrupted triples. Since only one side of each triple is corrupted at a time, the corrupted triples generated are compliant with the local closed world assumption. AmpliGraph's function `evaluate_performance` evaluates the performance of a trained KGE model by ranking a set of test triples from the KG against a set of corrupted triples. An embedding model is then evaluated based on how good it is at ranking the positive test triples higher than the negative corrupted triples. By instead passing the set of candidate triples to `evaluate_performance`, these could be ranked instead. Once the candidates have been given ranks, one simply needs to decide on a cutoff rank to consider the candidate as a true fact and add it to the KG. The rank cutoffs that were considered were 1, 4, and 7. These values were chosen as they are in the range of typical Hits@K cutoff values. Rank cutoff 1 is the strictest cutoff possible where only candidates ranked higher than *all* the corrupted triples are admitted. Hits@1 is also the strictest Hit ratio performance metric, where the triple with the highest score must be a true fact in the KG. As stated in Section 2.2.2, hit ratio cutoff values larger than 10 are rarely used. Therefore the other two rank cutoff values chosen for experiments were within this range.

3.5 Rule mining and evaluation

Rules are mined using the rule mining algorithm AMIE3 [44]. The authors of AMIE3 have made an implementation of it available on GitHub. It is released as an executable jar file that takes a KG in TSV file format as input and outputs rules accompanied by various confidence

scores. The jar file in release 3.0 was used for rule mining. It was run using the default settings used in the AMIE3 paper by Lajus et al. [44]. Since default settings change, the exact command used for rule mining was: `java -jar amie-milestone-intKB.jar -bias lazy -full -noHeuristics -ostd [TSV file]`.

As previously mentioned, the implementation of AMIE3 outputs the rules accompanied by confidence scores. Unfortunately, the scores are calculated on the same dataset the rules have been mined from. Since these datasets have been extended, they may contain datapoints that are false positives, leading to evaluation based on erroneous data. New rules that follow this incorrect data will therefore possibly receive high scores despite not making sense in the given context. To amend this problem, all mined rules are also evaluated on the original KG. This was not previously available in any release of AMIE3, which only allows rule evaluation in combination with rule mining, on the dataset from which the rules are being mined. Evaluation of rules on a custom KG was not possible until one of the authors, Jonathan Lajus, was kind enough to add a script to do exactly this.

In the final results from the experiments run, each mined rule had two different sets of confidence scores, one evaluated on the original KG and the other on the extended KG. In addition to scores, each rule was also accompanied by the parameters under which the KG was extended. These three parameters are the *entity selection method*, the *KGE model* used to rank candidates, and the *rank cutoff value* for candidate admittance. In summary, the output data points in the rule mining and evaluation process had these features:

- Rule
- Metrics
  - Head Coverage (extended and original KG)
  - PCA Confidence (extended and original KG)
  - Positive Examples (extended and original KG)
  - PCA body size (extended and original KG)
- Parameters
  - Entity seleciton method
  - KGE model
  - Rank cutoff
- Boolean indication for whether the rule was also mined from the original KG

With this information, one can look at the effect the parameters have on the number of rules mined and how the measured confidence differs when calculated on the extended KG versus the original KG.

3.6 Additional experimental setup details

The experiments were run on a server with 64 GB of RAM and an Intel Core i9-7900X 3.3GHz processor.

The implementation of AMIE used in our study is distributed under the Creative Commons Attribution-NonComercial license v3.0 by the YAGO-NAGA team and the DIG team. The program uses Javatools, a library released under the Creative Commons Attribution license v3.0 by the YAGO-NAGA team.

Chapter 4

Results

This chapter begins by presenting some of the central questions mentioned in the introduction. It continues by giving a general overview of the experimental results, before examining the findings in more detail. Finally, the discoveries are summarized, and some limitations of the experiment are highlighted.

4.1 Goals

We study how the extension of KG affects rule mining and the role of extension methods in the process. Some central questions here are:

1. Does adding new plausible facts lead to new rules being mined?
2. How does the PCA confidence of these rules compare to the rules mined from the original KG?
3. Can the rules mined from the original KG also be mined after the KG is extended?

Regarding extension methods, we looked at three parameters:

- the *entity selection strategy* for candidate generation
- the *KGE model architecture* for ranking candidates
- the *rank cutoff value* for admitting candidates to the KG

and their role in the quantity and PCA confidence of the rules that are mined.

4.2 Overview of results

This section gives a brief overview of the results and focuses on the fact that the KGE choice considerably influences the number of rules being mined. The majority of rules were mined from TransE-extended KGs, and were rules that upon inspection had little meaning to them. As speculated in section 3.2, the findings show that the TransE model was not a success, and this section justifies why results from this model are omitted in later parts of the data analysis.

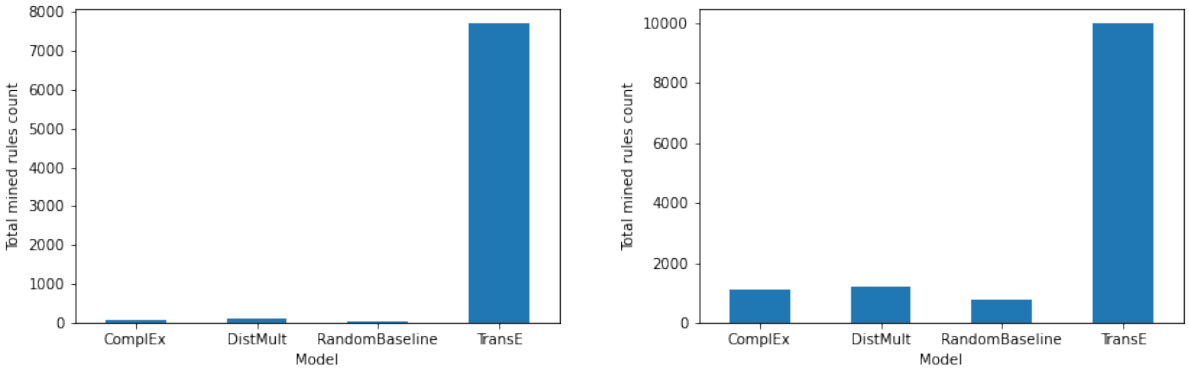
4.2.1 KG extension sizes

In the KG extension process there are 48 [1] different parameter combinations. This means that there are 48 different KG extensions for each original KG. Extension set sizes range from approximately 800 to 33500 triples, where the RandomBaseline model admitted the most candidates. This makes sense, as the models assign a random score to facts, so that many more receive a high rank. The *trained* models give most candidates a low score because most triples usually are bad candidates. Note that the extension sizes were about the same for both datasets, while WN18RR and the family KG respectively have 88 227 and 258 235 facts, therefore the extensions for WN18RR are relatively larger than for the family KG. See figure B.3 for the distribution of KG extension sizes over the KGE models used.

4.2.2 Rule set sizes and mean PCA confidence

Upon examining the number of rules mined per KG extension, one immediately sees some startling results, displayed clearly in figure 4.1. Most rule sets do not differ drastically in size, apart from those where TransE was used, where these rule sets are *exceptionally* larger. See fig. B.1 for a more detailed overview of the results for all KG extensions. When looking at the mean PCA confidence of the rules sets, one also notes that the score for the TransE sets are mostly around 0.1, while the remaining rule sets have a score on average around 0.5.

[1] 4 (KGE models) $\times$ 4 (entity selection methods) $\times$ 3 (rank cutoff) = 48 parameter combinations.



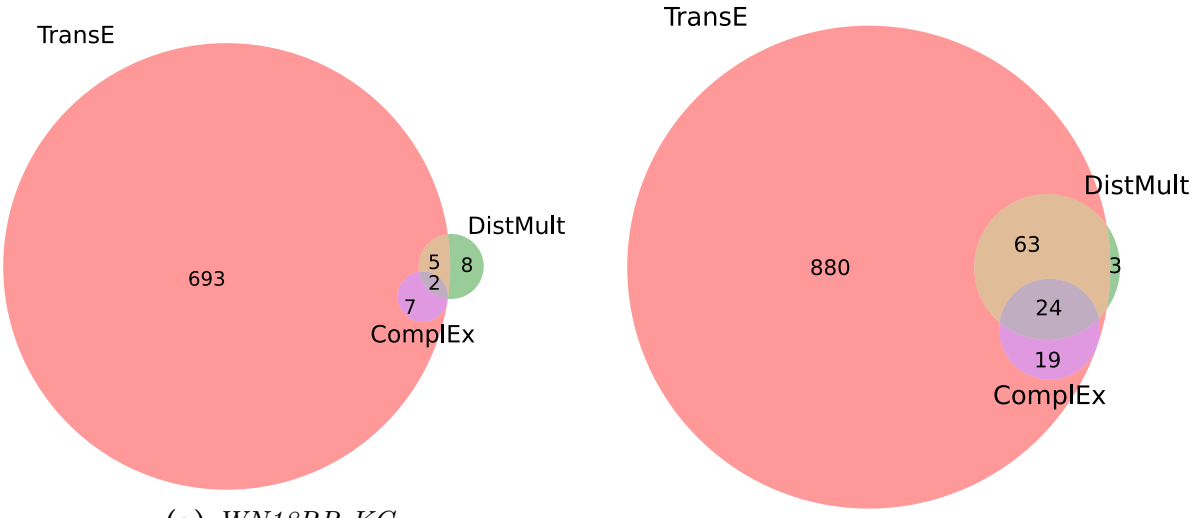
(a) *WN18RR* **(b)** *Family KG*

Figure 4.1: *Distribution of the total number of mined rules over different embedding models, so the sum of all rules mined over the different KG extensions partitioned by the embedding models used.*

As discussed in Section 2.2.3, TransE cannot learn certain relation qualities. Upon further inspection of the embedding vectors for relations in TransE, the vector values all tend toward zero (see B.1 and B.2). This can partially be explained by TransE's tendency to push the embeddings of symmetric relations toward the zero vector and its lack of ability to embed complex relations effectively. Both issues are explained in section 2.2.3. It seems that while scoring decently on the performance metrics during model selection, TransE has not properly embedded the relations in the KG and does not do well in completion for learning rules.

Interestingly, the number of rules mined from TransE-extended KGs is not proportional to the number of triples added to the original KG, which is much lower. There is something *particular* about the triples that TransE ranks highly, resulting in the introduction of more patterns in the dataset. If we look at the mentioned weaknesses of the TransE embedding model, we can reach some plausible explanations.

To illustrate this, let us again consider the KG from example 2, in which `Carol` has children `Ann` and `Bob`, and `Ann` and `Bob` are explicitly stated to be each other's siblings. Due to TransE's problems in embedding complex relations [78, 50], `Ann` and `Bob` are likely to be identically represented in the embedding space. According to the results, each relation in the family KG is represented as a null-vector in the TransE embedding. Thus all relations are considered essentially equivalent to each other. Hence, we find ourselves in a situation where `Ann` and `Bob` are equivalent, and all relations are equivalent. With these assumptions,



(a) *WN18RR KG.*

(b) *Family KG.*

Figure 4.2: *Distribution of new rules grouped according to which KGE was used to create the extended KG from which the rule was mined. Duplicates are removed.*

meaningless triples such as `(Carol, has_spouse, Ann)`, `(Carol, has_spouse, Bob)` and `(Bob, has_child, Ann)` would score highly according to our TransE model. TransE will assign a high score to all original triples with the relation swapped for another. This would lead to more triples sharing common entities, resulting in more patterns arising in the data. Therefore any combination of relations in the body and consequent would gain some degree of support in the extended KG, and hence be mined with AMIE3. The meaningless triples mentioned above are simply existing triples in the example KG with the relation switched for another that is, by the model, considered equivalent.

Some resulting nonsense rules from TransE extensions are shown in listing 4.1. Many of these would make sense if the relations were swapped for another, for example, if *relative* in the body of rule 3 were swapped with *sibling* the rule would make perfect sense. The second nonsense rule in the listing, $mother(x, y) \Rightarrow child(x, y)$, was also found by DistMult. This makes sense because for DistMult the two triples `(a, mother, b)` and `(b, mother, a)` will have the same score for all entities `a` and `b`, and the rule $mother(y, x) \Rightarrow child(x, y)$ is likely to have considerable support in the dataset. The remaining rules in listing 4.1 were only found in TransE-extended KGs.

Of the rules mined, 97% (WN18RR) and 76% (family KG) originate from KGs extended using TransE. The large number of rules produced marginalizes those mined using other embedding models. Due to this and the fact that the trained TransE model has poor relational

1 $spouse(x, y) \Rightarrow father(x, y)$
2 $mother(x, y) \Rightarrow child(x, y)$
3 $relative(y, x) \Rightarrow sibling(x, y)$
4 $spouse(x, y) \Rightarrow child(x, y)$
5 $relative(x, y) \Rightarrow sibling(x, y)$
6 $relative(x, z) \wedge sibling(z, y) \Rightarrow mother(x, y)$
7 $child(z, x) \wedge mother(z, y) \Rightarrow spouse(x, y)$
8 $relative(y, x) \wedge sibling(y, x) \Rightarrow father(x, y)$
9 $mother(z, y) \wedge mother(x, z) \Rightarrow child(x, y)$
10 $sibling(y, x) \wedge spouse(x, y) \Rightarrow father(x, y)$

Listing 4.1: *Selection of nonsense rules mined from KGs extended with TransE.*

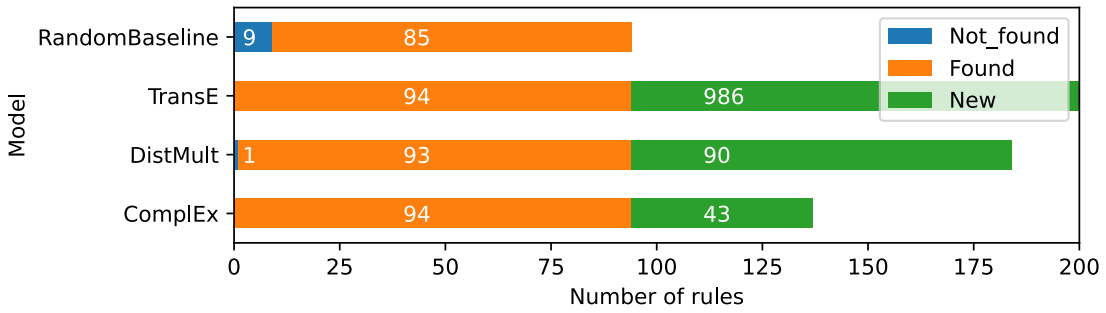
embeddings, we include rules mined from KGs extended with TransE only when examining embedding models, but not when looking at entity selection methods or rank cutoff values.

4.3 Effect of parameters

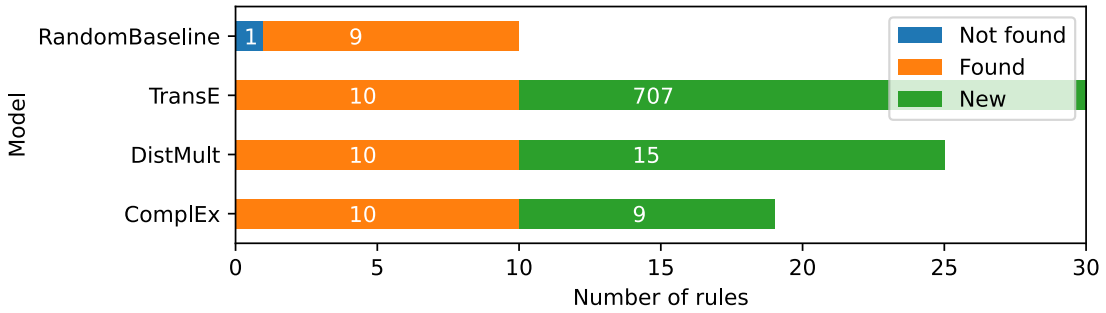
This section examines the effects of the three primary parameters in the KG extension process on the rules being mined. Note that when looking at rules mined from a KGs extended with a specific parameter fixed, the sets of rules will be a concatenation of all cases where that parameter choice was used. Consider the rules mined with ComplEx as the KGE model. This group contains rules mined from 12 different KG extensions because we need to look at all the combinations of entity selection methods and rank cutoff values ($4 \times 3 = 12$). This is differentiated from the case where we look at rules mined from a single extended KG, which we do in section 4.4.

4.3.1 Effect of KG embedding model

At first glance of the boxplots in figure 4.4 it may seem odd that the PCA confidence of rules mined from RandomBaseline extensions is so high. Boxplots represent the spread of data through their quartlies [25]. The box itself covers $Q_1 - Q_3$, with the horizontal line in the box marking the median. The dashed line across the plots indicates the median PCA confidence for the original rules. The boxplots used in this thesis also have *whiskers*, which are the lines extending vertically out of the box. Whiskers indicate the variability of the data

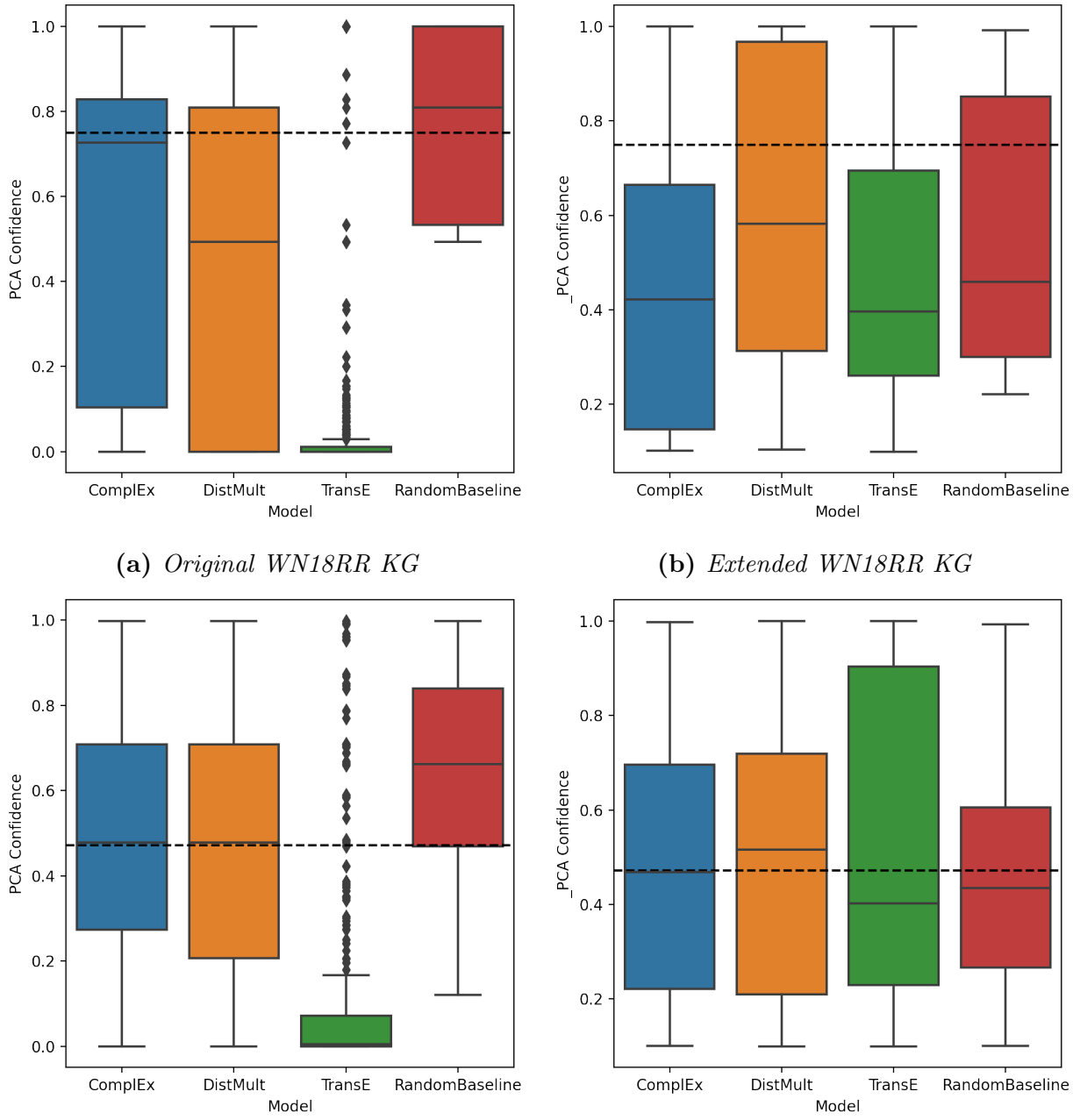


(a) *Family KG*



(b) *WN18RR KG*

Figure 4.3: *Distribution of rules over KGE models, not counting duplicate rules mined from multiple KGs. Note that the bar for TransE extends far beyond the chart, so the unusually high number of different rules mined from TransE extensions is not correctly shown here. The single original rule not found by DistMult in the family KG is* $child(x, y) \Rightarrow mother(y, x)$*. This rule has the second-lowest PCA confidence of all the original family KG rules.*



(a) *Original WN18RR KG*

(b) *Extended WN18RR KG*

(c) *Original family KG*

(d) *Extended family KG*

Figure 4.4: *Distribution of PCA confidence of mined rules by KGE models. PCA confidence scores are calculated on the original KG (4.4a, 4.4c) and the extended KG from which the rules are mined (4.4b, 4.4d). A rule can be mined from multiple KGs, therefore, this graph shows duplicates. The dashed line represents the median PCA confidence of the rules mined from the original KG.*

outside the upper and lower quartiles (respectively Q_1 and Q_3). Points beyond the whiskers are considered outliers.

By looking at figure 4.3 it becomes apparent that there are few different rules mined from the RandomBaseline extensions. In the WN18RR case, no new rules were found, and a single original rule was also never mined. The omitted rule,

$$has\_part(x, z) \wedge hypernym(y, z) \Rightarrow has\_part(x, y)$$

has the lowest PCA confidence out of all the original rules. Since there already was little support in the KG for the rule, the addition of noise may have obscured too much evidence for the algorithm to consider the rule significant. See table A.1 to see all rules mined from the original WN18RR KG, sorted by their PCA confidence.

Model	**Original rules**		**New rules**		**Orig. not found**	
	Count	% of mined	Count	% of mined	Count	% of original
Baseline	85	100%	0	0%	9	10%
TransE	94	9%	986	91%	0	0%
DistMult	93	51%	90	49%	1	1%
ComplEx	94	69%	43	31%	0	0%

Table 4.1: *Distribution of all the rules mined over KGE models. KG: family KG.*

Model	**Original rules**		**New rules**		**Orig. not found**	
	Count	% of mined	Count	% of mined	Count	% of original
Baseline	9	100%	0	0%	1	10%
TransE	10	1%	659	99%	0	0%
DistMult	10	67%	5	33%	0	0%
ComplEx	10	59%	7	41%	0	0%

Table 4.2: *Distribution of all the rules mined over KGE models. KG: WN18RR.*

The family KG produced similar results, where AMIE3 failed to find 9 original rules in the RandomBaseline-extended KGs. All of these rules were among those with the lowest PCA confidence of the original rules. Refer to table A.1 for all rules mined from the original family KG, sorted by their PCA confidence. When the rules with the lowest PCA confidence in a set are removed, the median PCA confidence naturally goes up. This explains how the RandomBaseline rules have a higher median PCA confidence than the original, shown in tables 4.1 and 4.2.

As presented in section 4.2.2, AMIE3 mines a great deal more rules from KGs extended with TransE. Due to the low mean PCA confidence for these rules (0.038 for WN18RR and 0.089 for the family KG) it is clear that there is little support for the rules in the original KGs. However, if one evaluates the TransE rules on the *extended* KGs they were mined from, the PCA confidence increases substantially, as seen in figures 4.4b and 4.4d. So even though most of these rules have little support from the original KGs, it was no mistake that AMIE3 mined so many rules from the TransE-extended KGs.

DistMult and ComplEx are closely related embedding models as they have similar scoring functions (the scoring function of ComplEx corresponds to that of DistMult but with real embeddings) and perform similarly on benchmark datasets [72]. As mentioned in section 2.2.3, DistMult cannot represent asymmetric relations. AMIE3 can only mine Horn rules, thereby a rule defining the asymmetry of a relation cannot be mined. However, DistMult's shortcomings may still affect the quality of the embedding and thus the KG extension. From the results, it seems that ComplEx is a somewhat stricter model, in the sense that it ranks candidate triples lower against corrupted triples than DistMult does. This is evidenced by the KG extension sizes being larger for DistMult than for ComplEx; DistMult is admitting more candidates. For example, of the candidates generated with the entity selection method "probabilistic" from the WN18RR KG, DistMult assigned 1468 candidates rank 1 while ComplEx only did this for 866 candidates. As a result, more rules are mined from DistMult-extended KGs.

4.3.2 Effect of entity selection method

The *probabilistic*, *random*, and the *least frequent* entity selection strategies perform relatively similarly if we look at the PCA confidence box plots in figure 4.5. The *most frequent* strategy, however, performs worse, both on the WN18RR and family KG. This is consistent with AmpliGraph's assumption that the most frequent entities are less likely to have missing facts [21].

In the WN18RR KG all original rules were found, but in the family KG only the *least frequent* strategy resulted in all the original rules being mined. It also resulted in the least new rules being mined, while the *most frequent* strategy led to the most new rules.

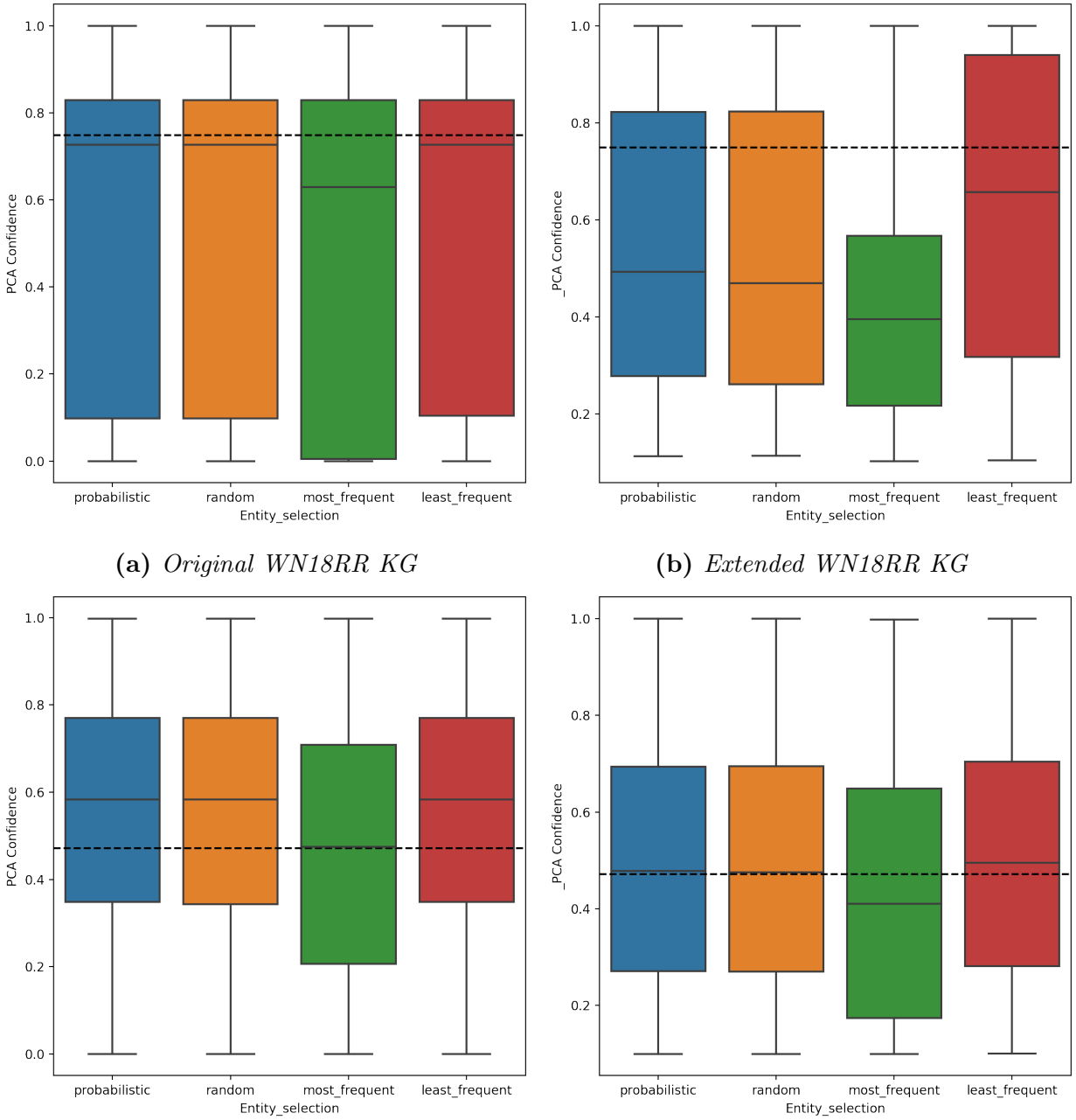
Overall it seems like *probabilistic*, *random*, and the *least frequent* entity selection strategies all are appropriate in regard to maintaining PCA confidence. If, however, the goal is to mine many new rules, then *most frequent* appears to be best suited, though at the sacrifice of PCA confidence.

Strategy	**Original rules**		**New rules**		**Orig. not found**	
	Count	% of mined	Count	% of mined	Count	% of original
Random	90	71%	36	29%	4	4%
Most freq.	88	46%	105	54%	6	6%
Least freq	94	77%	28	33%	0	0%
Probabilistic	91	73%	33	27%	3	3%

Table 4.3: *Distribution of all the rules mined over entity selection strategies for the family KG.*

Strategy	**Original rules**		**New rules**		**Orig. not found**	
	Count	% of mined	Count	% of mined	Count	% of original
Random	10	53%	9	47%	0	0%
Most freq.	10	42%	14	58%	0	0%
Least freq.	10	48%	11	52%	0	0%
Probabilistic	10	45%	12	55%	0	0%

Table 4.4: *Distribution of all the rules mined over entity selection strategies. KG: WN18RR.*



(a) *Original WN18RR KG*

(b) *Extended WN18RR KG*

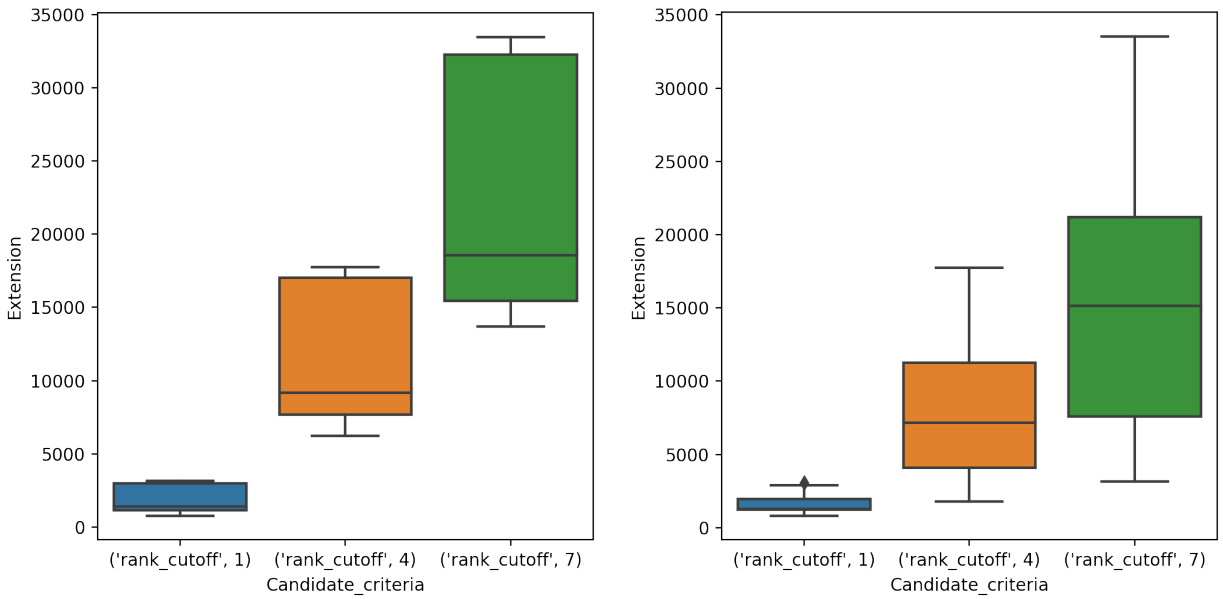
(c) *Original family KG*

(d) *Extended family KG*

Figure 4.5: *Distribution of PCA confidence of mined rules by entity selection strategies. PCA confidence scores are calculated on the original KG (4.5a, 4.5c) and the extended KG from which the rules are mined (4.5b, 4.5d). A rule can be mined from multiple KGs, therefore this graph shows duplicates. The dashed line represents the median PCA confidence of the rules mined from the original KG.*

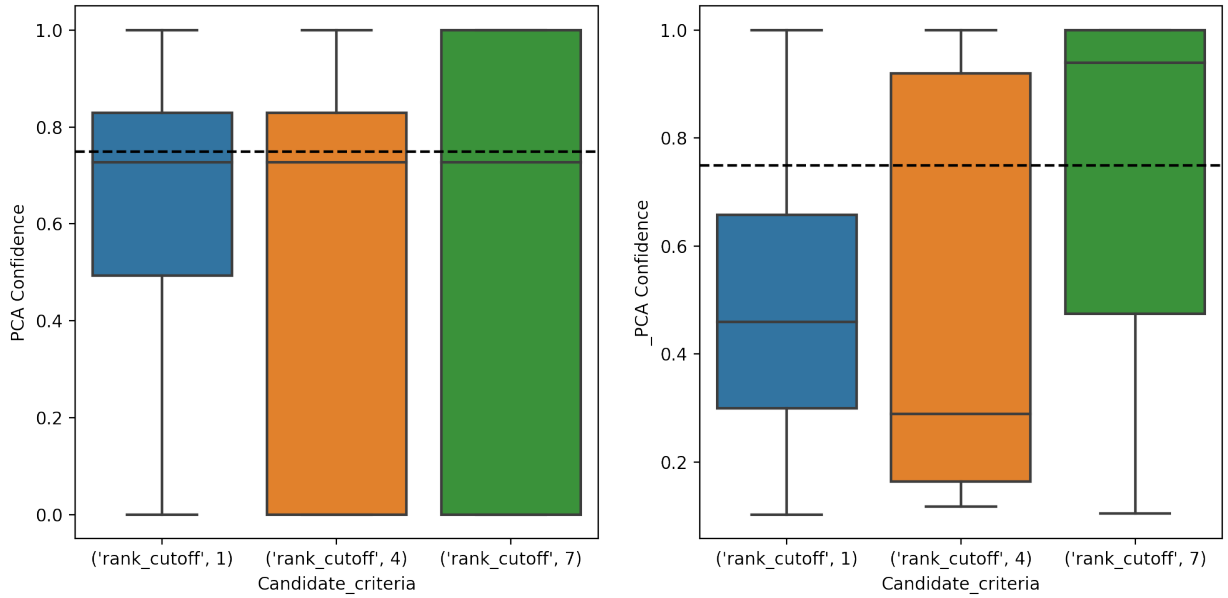
4.3.3 Effect of rank cutoff value

Rank cutoff values determine which candidates are admitted to the KG. Only those with rank on the cutoff or above are accepted. Therefore, the lower the rank cutoff, the more candidates are added to the KG. The box plots in fig. 4.6 show evidence of this. Tables 4.5 and 4.6 both show that when the rank cutoff was 1, all the original rules were found. A decrease in rank cutoff led to the mining of more new rules, but also led to some original rules not being mined, especially in the WN18RR dataset. PCA confidence-wise the rank cutoff values had a slightly different effect on the two datasets. In the WN18RR dataset there was little difference in PCA confidence over the ranks when calculated over the original KG, while for the family KG the PCA confidence was increased at rank cutoff 4 and 7, with no change at rank cutoff 1. When calculated over the KG extensions, rank 1 and 4 did worse than the mean of the original rules, while rank cutoff 7 did significantly better. On the family KG, PCA confidence calculated on the extended KGs led to little variety over the rank values. Rank 1 resulted on the same median as the original rules, rank 4 just above and rank 7 just below. Overall, if the goal is to find many new rules, then a higher rank cutoff values is more appropriate.



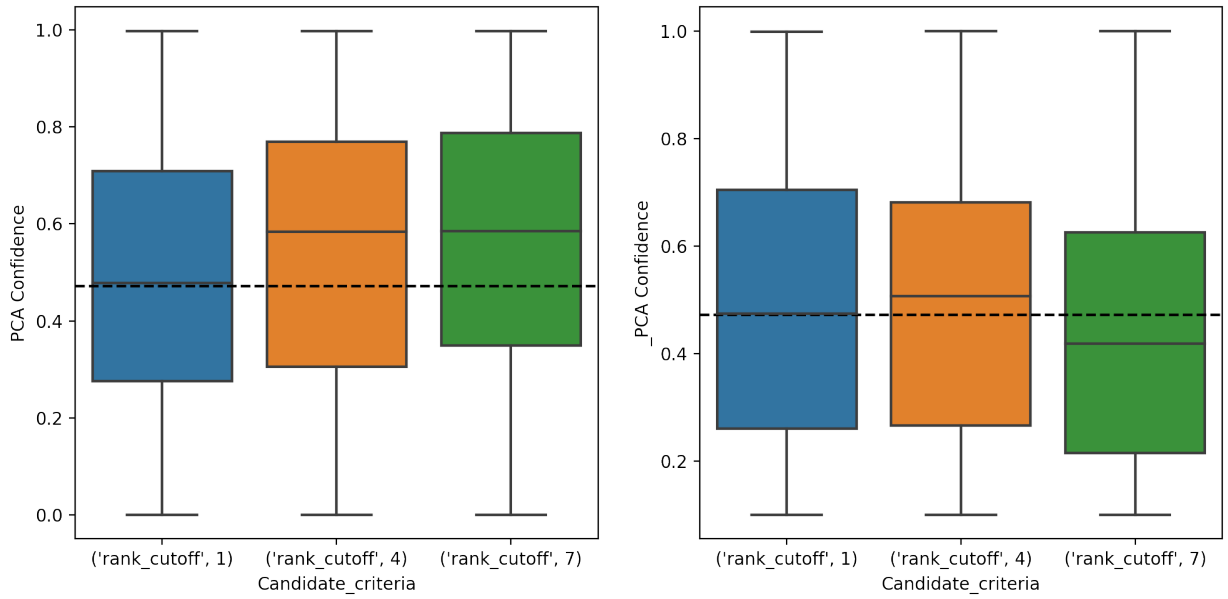
(a) *WN18RR KG.* **(b)** *Family KG.*

Figure 4.6: *Distribution of KG extension sizes over rank cutoff values.*



(a) *Original WN18RR KG*

(b) *Extended WN18RR KG*



(c) *Original family KG*

(d) *Extended family KG*

Figure 4.7: *Distribution of PCA confidence of mined rules over rank cutoff values. PCA confidence scores are calculated on the original KG (4.7a, 4.7c) and the extended KG from which the rules are mined (4.7b, 4.7d). A rule can be mined from multiple KGs, therefore this graph shows duplicates. The dashed line represents the median PCA confidence of the rules mined from the original KG.*

Rank cutoff	Orig. rules		New Rules		Orig. not found	
	Count	% of mined	Count	% of mined	Count	% of original
1	10	50%	10	50%	0	0 %
4	4	27%	11	73%	6	60%
7	3	20%	12	80%	7	70%

Table 4.5: *Distribution of all the rules mined over rank cutoff values. KG: WN18RR.*

Rank cutoff	Orig. rules		New Rules		Orig. not found	
	Count	% of mined	Count	% of mined	Count	% of original
1	94	90%	11	10%	0	0 %
4	90	51%	85	49%	4	4%
7	88	45%	108	55%	6	6%

Table 4.6: *Distribution of all the rules mined over rank cutoff values. KG: family.*

4.3.4 Summary of effect of parameters

The three parameters for KG extension were shown to have a varying degree of influence. All entity selection strategies, apart from *most likely*, performed equally well. This was consistent with the assumption of the authors of AmpliGraph, being that the most frequent entities are less likely to have missing facts [19]. Of the KGE models, TransE led to a substantially greater number of new rules being mined compared to the other models. However, these rules had very low PCA confidence when calculated over the original dataset. ComplEx and DistMult performed similarly concerning PCA confidence, with slightly more new rules being mined with DistMult due to the model giving candidate triples a higher score than ComplEx did. The rank cutoff value gave expected results, where a value that accepted more new candidate triples led to more new rules being minded. When calculated over the original KG, the PCA confidence of rules did not significantly vary among the rank cutoff values.

4.4 Rule comparison

In the WN18RR dataset there was one rule,

$$derivationally\_related\_form(y, x) \Rightarrow derivationally\_related\_form(x, y)$$

which was mined in all 48 KG extensions. It has a PCA confidence of 1, meaning that there are no data points in WN18RR that do not follow the rule. In table 4.7, all original rules sorted by their PCA Confidence are listed. How often these rules were mined from KG extensions is also listed under the "Frequencies" column. Using Spearman's rank correlation coefficient, there was found a correlation coefficient of 0.65 with p-value 0.04. These values are positively correlated with the PCA confidence, meaning that as the PCA confidence increases, the frequency also increases. From the family KG, 94 original rules were mined, and the corresponding table B.3 is listed in the appendix. For these rules, the PCA confidence and frequencies were also positively correlated with a correlation coefficient of 0.86 and p-value of $5.85e^{-29}$. The rule mined the least number of times is

$$father(x, z) \wedge sibling(z, y) \Rightarrow relative(x, y)$$

which only was mined in 10 out of the 48 extensions. Consistent with the correlation between PCA confidence and mining frequency, this is the rule with the second-lowest PCA confidence. Table B.3 also shows us that all rules with the *relative* predicate were not mined for all 48 extensions, and all with different predicates in the antecedent were mined from all extensions. This could be due to the simple fact that *relative* is the least frequently used relation in the KG, leading to less supporting data points and correspondingly lower PCA confidence.

4.4.1 PCA distribution

For both the WN18RR and family KG, the PCA confidence of the original rules is relatively spread out. A few rules have a PCA confidence of 1 or close to 1, and none have PCA confidence below 0.1. This makes sense because this would imply the absence of supporting datapoints and thus AMIE3 would never mine the rule. The majority of new rules on the

Rule	Frequencies	PCA Confidence
DRF(y, x) ⇒ DRF(x, y)	48	1.000000
IH(x, z) ∧ SDTO(z, y) ⇒ SDTO(x, y)	24	0.886525
hypernym(x, z) ∧ SDTO(z, y) ⇒ SDTO(x, y)	36	0.828871
has_part(x, z) ∧ SDTO(z, y) ⇒ SDTO(x, y)	24	0.809524
has_part(z, x) ∧ SDTO(z, y) ⇒ SDTO(x, y)	24	0.771739
hypernym(z, x) ∧ SDTO(z, y) ⇒ SDTO(x, y)	34	0.726667
has_part(x, z) ∧ IH(y, z) ⇒ has_part(x, y)	19	0.532544
DRF(x, z) ∧ SDTO(z, y) ⇒ SDTO(x, y)	24	0.492857
DRF(z, x) ∧ SDTO(z, y) ⇒ SDTO(x, y)	24	0.492857
has_part(x, z) ∧ hypernym(y, z) ⇒ has_part(x, y)	21	0.104016

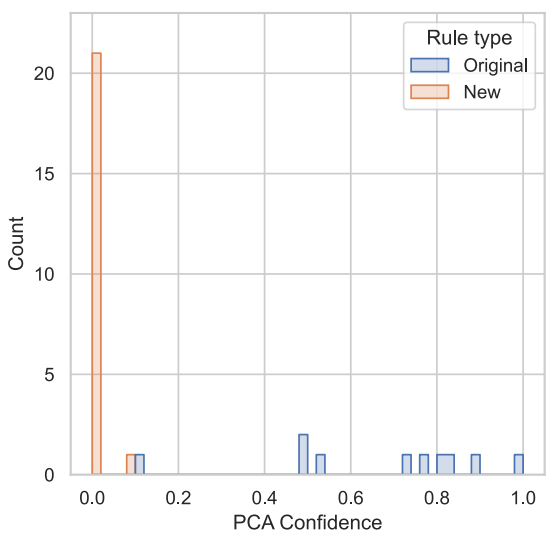
Table 4.7: *Rules mined from the original WN18RR KG, with their corresponding PCA confidences and how many times they were mined from the 48 extensions. DRF = "derivationally related form", STDO = "synset domain topic of" and IH = "instance hypernym".*

other hand *do* however have a PCA confidence of 0. This means that most, if not all, the support for new rules lies in the extension of the KG. If all sets of supporting datapoints are dependent on the extension, then when PCA confidence is measured on the original KG, this new rule would get a PCA confidence score of 0. Looking closely at figure 4.8b one can see that two new rules have PCA confidence above 0.2. However, these were also rejected by AMIE3 because the PCA body size (the number of instances of that rule) is so small that the PCA confidence calculated is not very reliable.

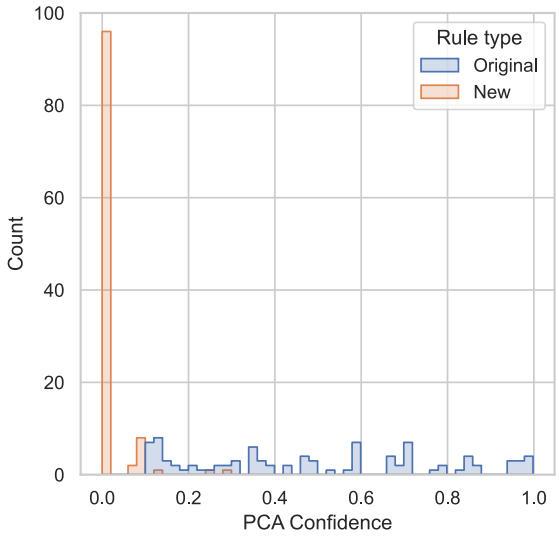
When plotting the PCA confidence of new rules calculated over the KGs from which they were mined, the new rules become somewhat more evenly distributed along the PCA confidence interval. Figures 4.8c and 4.8d show this, where the PCA confidence of the new rules here is not calculated over the original KG, but the KG from which they were mined that gave them the best PCA confidence score. This method was chosen to highlight how the KG extensions changes the PCA confidence of new rules.

4.5 Summary of findings

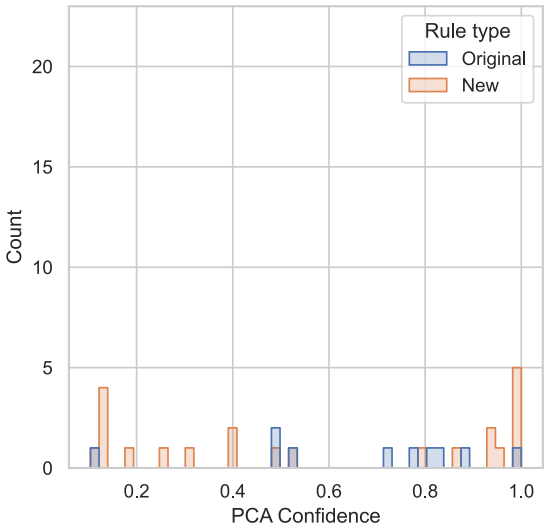
We restate the central questions for the experiment posed at the beginning of this chapter. These questions were:



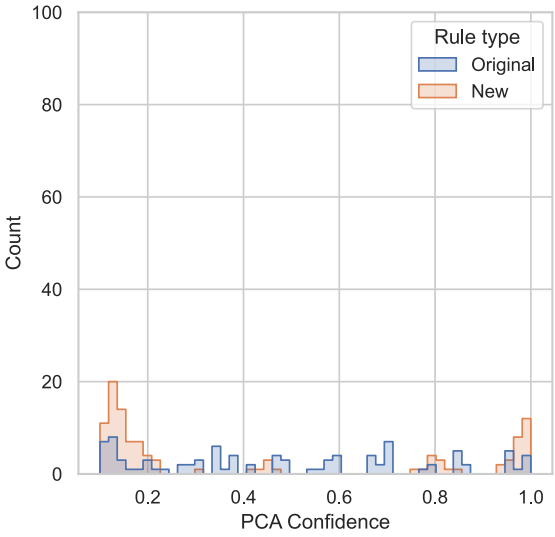
(a) *WN18RR - Original KG*



(b) *Family - Original KG*



(c) *WN18RR - Extended KG*



(d) *Family - Extended KG*

Figure 4.8: *Distribution of PCA confidence of original rules and new rules. In subfigures 4.8a and 4.8b the PCA confidence is calculated over the original KG. For subfigures 4.8c and 4.8d the PCA confidence is the best score calculated over the extended KG from which the new rules was mined. The PCA confidence of original rules is always calculated over the original KG.*

1. Does adding new plausible facts lead to new rules being mined?
2. How does the PCA confidence of these rules compare to the rules mined from the original KG?
3. Can the rules mined from the original KG also be mined after the KG is extended?

Questions 1 is the simplest, and our experiment has clearly shown that, yes, adding new plausible facts does result in new rules being mined. Examples of rules that were mined after extending the KGs are:

$$\text{relative(z, x)} \land \text{relative(y, z)} \Rightarrow \text{relative(x, y)}$$
$$\text{father(x, y)} \land \text{mother(x, y)} \Rightarrow \text{child(x, y)}$$
$$\text{father(y, z)} \land \text{mother(x, z)} \Rightarrow \text{sibling(x, y)}$$
$$\text{relative(z, y)} \land \text{spouse(x, z)} \Rightarrow \text{relative(x, y)}$$
$$\text{relative(x, z)} \land \text{relative(y, z)} \Rightarrow \text{relative(x, y)}$$
$$\text{father(x, z)} \land \text{mother(y, z)} \Rightarrow \text{sibling(x, y)}$$

At close inspection, these rules are relatively poor, and as seen in figure 4.8, the PCA confidence of new rules was most often zero or close to zero when calculated over the original KG. However, when measured on the extended KGs from which they were mined, the scores were significantly better and had similar distributions to the original rules, which answers question 2. Regarding question 3, a strong correlation was found between the PCA confidence of original rules and how often the rule was mined from extended KGs. The higher the PCA confidence, the more likely the rule will be mined from the extended KG.

4.6 Limitations of experiment

The experiment has many limitations, most related to computational restrictions. For example, when generating candidates, the optimal approach would be to generate all possible candidates and evaluate them all. This is simply not possible due to computational limitations. As mentioned in section 3.4.1, there are more intelligent strategies for entity selection, but these were also too computationally intensive to take into use. A larger and better set

of candidate triples would substantially strengthen the conclusions about the KGE model and rank cutoff values when evaluating the KG extension process parameters. When only a small fraction of the possible candidates are ranked by models, we get an incomplete picture of the model's behaviour.

The hyperparameter search for the KGE models could also be improved upon with greater computational resources, but it is not as influential as increasing the candidate set size and quality. This is because random search has been shown to measure well against grid search [47], so the effect on the performance of the model is likely to be minimal.

Chapter 5

Related work

The related work chapter is divided into three sections, KGEs, rule mining, and approaches combining the two methods. For perspective, it is worth noting that KGEs and rule mining are two knowledge graph completion (KGC) methods at the opposite ends of the rule vs. predictive spectrum, depicted in figure 5.1. On the left-hand side there are methods that prioritize rule extraction over triple prediction. As one moves from the left gradually to the right, there are methods that to an increasing degree prioritize triple prediction over rule extractions. KGEs are furthest to the right on this scale, while rule mining methods are furthest to the left. These two groups have been used in the present study and hence works within these areas are focus of the related work chapter. This figure can, however, serve as a reminder that there are many other methods in the grey area between the two extremes.

5.1 Knowledge graph embedding techniques

A wide range of statistical-based relational learning techniques have been proposed for KGC [61]. Out of these methods, vector space embedding approaches are one of the most successful due to their performance and scalability. One of the main problems with vector space embeddings, as with many other statistical machine learning methods, is that the results are not explainable [9]. Rule-based approaches alleviate this problem to some degree. Early works within vector space embeddings include TransE [10] and DistMult [82], which employ

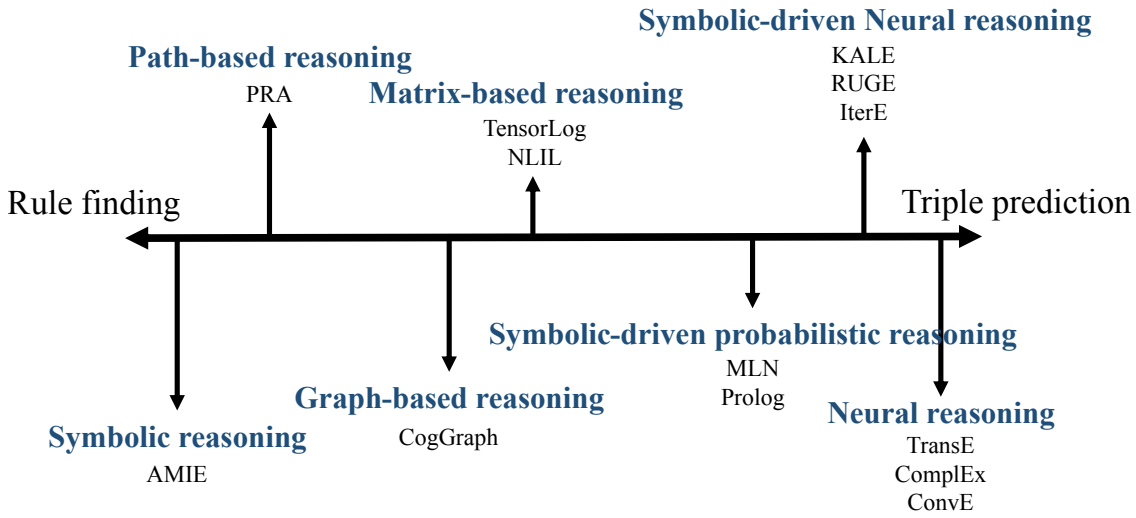


Figure 5.1: *KGC techniques arranged by the degree to which rule finding or triple prediction is the focus. Based on work by Zhang et al. [83].*

simple vector space operations for link prediction. These simple models can handle large-scale KGs, but this is often at the cost of expressiveness [23]. There have been many attempts at increasing expressiveness while maintaining simplicity, such as SimplE [42], HolE [62], and RotatE [70]. A more direct approach for increasing expressiveness is the employment of deep neural networks. This has been done using multi-layer perceptrons [24], semantic matching energy networks [11] and neural tensor networks [69]. It has however been shown that these approaches have more parameters and are prone to overfit [61]. Improving upon this, there have been many recent approaches using convolutional neural networks [23, 59, 67, 74] .

5.2 Rule mining

Traditional rule mining methods find rules of quality by statistically evaluating support and confidence in candidate rules. AMIE and its successors are part of this group and have for a while been considered at the forefront of both scalability and performance when it comes to mining first-order Horn rules from knowledge graphs. Exact statistical evaluation of rules is expensive, so there have been approaches adopting embedding methods to score rules [82, 63, 64]. The rule miner AnyBURL uses rule generation methods different from AMIE, where rules are generated by sampling paths in the KG [53]. These rules are evaluated with the same metrics, but scores are approximated based on sampling in contrast to the

exact evaluation done in AMIE3. AnyBURL used reinforcement learning to improve their sampling of paths, leading to higher quality rules being found earlier in the search. A more recent approach to rule mining with reinforcement learning takes advantage of embedding information to achieve better performance and allows scalable mining of long rules [16]. This method outperforms AMIE+, and the authors point out that the many optimizations made to AMIE+ (resulting in AMIE3) could be applied to any top-down rule mining methods, including theirs. Top-down in this context means beginning the search with the most general rules and then expanding on them, essentially performing a breadth-first search in the space of possible rules. The bottom-up approach, on the other hand, works from the data, starting with very specific rules and then generalizing them.

A KGC family similar to rule mining is path-based reasoning methods. One of the first of these was the path ranking algorithm (PRA) [45]. This approach trains a binary classifier for each relation r in the KG, that given two entities h and t determines if they are connected via a relation r. Path ranking algorithms struggle with sparseness in KGs [51], so there have been attempts to combine PRAs and embedding methods. An example of this is PTransE, which considers relation paths with multiple steps, and uses embedding methods similar to TransE to represent these paths [49].

5.3 Approached combining KGE and rule mining

When it comes to combining the rule-based and embedding-based approach there seem to be two main ideas. The first idea is to integrate the two approaches tightly. An example of this is KALE [35], which initially learns rules from a TransE embedding and thereafter retrains the embeddings on a joint model for rules and triples, resulting in a unified framework. Based on KALE, Guo et al. proposed RUGE [36], which learns entity and relation embeddings through iterative guidance from soft rules. The soft rules are queried to obtain soft labels for unlabeled triples, and these newly labeled triples are then used to update the embedding model. However, it has been noted that these systems generally do not allow for rules that contain constants [54]. KALE and RUGE infer rules once at the beginning of the process, therefore the embedding can benefit from information in the rules, but the rules are never improved upon with the embedding. IterE [84] addresses this issue and also infers new rules based on the updated embedding. The update of rules and embeddings is done iteratively, and the rules are given a score based on the embedding of the relations included in the rules.

The second idea proposes to combine the rule-based and embedding models into an ensemble. An ensemble is a machine learning method that uses multiple models to obtain a better prediction than one would with a single model. This is often done by allowing models to "vote" upon a prediction. The ensemble approach has been studied by Wang et al. [76] and was the focus of the experimental study by Meilicke et al. [52] mentioned in the introduction. The authors recently extended this study with a more up-to-date analysis. They explain "*why a naive way to combine symbolic and latent knowledge graph completion techniques works surprisingly well*" [54]. The work in the present study combines rule-based and embedding models in a different manner, where the information gained by one model is passed on to the next in the form of an extended, or more "complete", KG. While the first model influences the outcome of the next, it does not play a direct role in the final outcome; in this case the outcome being the rules mined from the KG.

Chapter 6

Discussion

Prior to developing the idea explored in this thesis, a different proposal for extracting rules from KGs was considered. It has similarities with the current work, but ultimately was deemed unachievable to implement. This chapter starts by explaining this failed idea and explain why it was unsuccessful, before summarising the findings of the more successful work described in this thesis and provide an overall evaluation. Certain design choices are assessed, and the chapter concludes with propositions for further work.

6.1 Mining ontologies through queries

This abandoned approach is similar to the KG extension and mining pipeline used for experiments in this study, but differs in the manner in which the KGEs are used to add implicit information to the KG. Instead of a rule mining algorithm that takes a KG as input, this approach uses the HORN algorithm by Angluin et al [6], which requires an "oracle" to learn from. The oracle can be viewed as a teacher who is considered an expert on the rules we would like to learn, with the idea being to use a well-trained KGE model as the oracle for the HORN algorithm. The HORN algorithm outputs rules in the form of propositional logic. If the reader is unfamiliar with propositional logic, a brief introduction covering the relevant aspects can be found in the appendix at section C. Below, we will examine the HORN algorithm, before explaining why it was incompatible with RDF-style KGs.

6.1.1 The HORN algorithm

Similarly to AMIE3, the HORN algorithm is designed to output a set of rules in the form of a Horn sentence. It does this by posing equivalence queries and membership queries. An equivalence query asks if the current hypothesis, a Horn sentence, is equivalent to the target Horn sentence that the oracle knows. If the oracle answers "*yes*", the target set of rules has been found, and the algorithm terminates. Otherwise, a "*no*" answer is accompanied with a valuation in which the target and hypothesis are evaluated to different boolean values. A *positive* valuation is one which makes the target Horn sentence true, while a *negative* valuation is one which evaluates the target to false.

A membership query asks whether a given valuation is positive or negative, to which the oracle only answers "*yes*" or "*no*". We will now focus on the implementation of the oracle and why this was incompatible with the set-of-triples data format. For more information on the HORN algorithm, please refer to the work *Learning conjunctions of Horn clauses* by Angluin et al. [6]. The important takeaway for this algorithm is that it runs in polynomial time on the size of the target and the number of variables considered.

6.1.2 The difficulty in translating triples to valuations

If the oracle were a machine learning model, it would need to be able to classify valuations as positive or negative. The equivalence query could be simulated with many membership queries, where if enough valuations are tested one can with high probability determine whether the target is logically equivalent with the hypothesis. So all one needs to simulate the oracle is a binary classifier. This approach rested on the idea that with enough valuations labeled as positive/negative, one could train a model to implicitly represent the target Horn sentence that determines if the valuations are positive or negative.

The problem with applying this idea to KGs arises immediately when trying to express all entities and relations of a KG with literals. This is necessary in order to create a set of positive valuations that collectively represent all information in the KG. The approach of assigning an individual literal to each entity and relation is not feasible. YAGO4 [66], for example, contains over 50 million distinct entities, meaning that each data point in the training set would have at least 50 million features. One can attempt to assign shared *roles*

to entities to limit the number of elements to encode, but the erasure of the unique identity of entities leads to further problems when it comes to maintaining patterns in the data. To demonstrate this, let us again use the KG from Example 1. Assume we have some intelligent way of assigning roles to all the individuals in the KG, which in this simple example will be the role of "adult" and "child". `Carol` will be an adult, and `Ann` and `Bob` children. Now we have reduced the number of entities by one, and the KG now looks like this:

Example 2. An even simpler KG.

```
<child> <hasParent> <adult>
<adult> <hasChild> <child>
<adult> <hasChild> <child>
<child> <hasSibling> <child>
<child> <hasSibling> <child>
```

Now the KG has been simplified, and almost half the data points have become redundant. If we now consider the rule

$$hasSibling(x, y) \wedge hasParent(y, z) \Rightarrow hasParent(x, z)$$

which previously predicted the new triple `(Bob, hasParent, Carol)`, it now only predicts `(child, hasParent, adult)`, which is not new information. By removing the identity of the individuals in the family, we removed the information that Ann is specifically the sibling of Bob and has Carol as a parent, which implied that Bob *also* has Carol as a parent. Hence, the valuations must represent all unique entities, or find a way to create replacement labels that maintain all the relevant information about the roles entities have in the KG. This becomes unfeasible with large KGs, such as the mentioned YAGO4.

6.2 Conclusion of findings

This study examined how adding new facts deemed plausible by a KGE affected the rules mined from the extended KG. We showed that extending KGs in such a way did in fact lead to new rules being mined and that the quality of these rules (approximated by PCA confidence) was similar to that of the original rules, but only when the confidence score was calculated over the extended KG from which the new rule is mined. It was also shown that

there was a strong correlation between the PCA confidence of an original rule and whether it was re-mined after the KG was extended.

When evaluating the effect of the three parameters in the experiment, the choice of KGE was shown to clearly be the one with the highest impact. The RandomBaseline model led to no new rules being mined and the omittance of original rules with the lowest PCA confidence (calculated over the original KG). As RandomBaseline was the only KGE that led to no new rules, we can conclude that the other embeddings found non-trivial patterns in the data, reflected by the new rules mined on the extended dataset. KGs extended with TransE led to an unusually large number of new rules, as the triples ranked highly by TransE contributed significantly to introducing new patterns in the KG. There was surprisingly little overlap between the new rules mined from KGs extended with different KGEs, underlining the impact of the choice of KGE.

6.3 Discussion of design choices

The area under the precision-recall curve (AUC-PR) is a metric appropriate for evaluating the performance of KGE models [39]. A precision-recall curve is a plot of the precision over the y-axis and recall over the x-axis. AUC-PR is thus the area under this curve and describes how well a model correctly predicts the positive class. A model that assigns a random score to a triple will result in a horizontal line based on the distribution of classified triples (positive or negative). Therefore an AUC-PR score of 0.5 is good if 99% of the classified triples are negative, but if the triples are evenly distributed then an AUC-PR score of 0.5 would indicate that the model classified no better than randomly. This metric could have been used in the study for evaluation of the KGE models, but was omitted as it requires context for interpretation.

We have used the PCA when generating and evaluating rules, which is merely an approximation of the truth. Just because only one child of a person is listed in Wikidata does not mean that this person does not have more children. These missing children were perhaps just not as famous and thus not mentioned in the database. Expanding on this idea of measuring the "truth" of rules, rule mining approaches have been especially popular due to their explainablity, and the induced knowledge can lead to interesting new knowledge. It

is, however, also prone to errors that occur when creating rules based on bias or errors in the data. For example, in Wikidata5M 99% of all football players are men, leading to the erroneous rule that if a person is a football player, they are a man. This is, of course, not an accurate rule by our understanding, but it captures the bias in the data. Therefore the fact that rules in this study are only evaluated on PCA confidence is a severe limitation when evaluating their quality.

6.4 Further work

There are many ways to expand upon the work in this thesis. As mentioned in section 4.6, the main limitations of the experiment itself are due to computational restrictions. The quality of the results could most likely be substantially improved if larger and more interesting KGs were used, as when KGs are restricted to only six relation types there is a relatively small area to create rules within. The *form* of the rules was also limited. Rules contained no entities, only variables, therefore relevant rules such as

$$won\_award(x, Spellemannprisen) \Rightarrow citizen\_of(x, Norway)$$

were never considered. Section 2.3.3 explained how AMIE3 is capable of mining such rules with instantiated variables, but at the expense of increased runtime, which is why why it was not done the experiments.

The RandomBaseline KGE assigns a random score to each triple. If the candidates are randomly generated, then the extension of a KG with the RandomBaseline would be equivalent to adding noise to the data. In the experiments of the present study it was shown that not all original rules were mined when the KG was extended with the RandomBaseline. This was, however, only a mere observation, and the extent to which the addition of noise to the data affects the rules being mined from it would be an interesting direction of further research.

As mentioned in the introduction, the process of first applying a KGE and *then* a rule mining algorithm could be swapped. In this reverse approach, a set of mined rules would be used to extend the KG, and thereafter a KGE would be trained on the extended KG. Instead of evaluating the eventual rules mined from the extended dataset, one would evaluate the

resulting KGEs trained on the extended KG. In this way, one could perhaps gain new insight into the choice that rule mining algorithms have on the KGE trained on the extended data.

List of Acronyms and Abbreviations

AUC-PR	area under the precision-recall curve.
CWA	closed world assumption.
GPro	graph pattern probability model.
GRank	graph pattern entity ranking model.
KB	knowledge base.
KG	knowledge graph.
KGC	knowledge graph completion.
KGE	knowledge graph embedding.
MR	mean rank.
MRR	mean reciprocal rank.
OWA	open world assumption.
PCA	partial completeness assumption.
PRA	path ranking algorithm.
RDF	Resource Description Framework.
W3C	World Wide Web Consortium.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.
URL: `https://www.tensorflow.org/`. Software available from tensorflow.org.

[2] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 207–216, 1993.

[3] Rajendra Akerkar and Priti Sajja. *Knowledge-based systems*. Jones & Bartlett Publishers, 2009.

[4] Mehdi Ali, Max Berrendorf, Charles Tapley Hoyt, Laurent Vermue, Sahand Sharifzadeh, Volker Tresp, and Jens Lehmann. PyKEEN 1.0: A Python Library for Training and Evaluating Knowledge Graph Embeddings. *Journal of Machine Learning Research*, 22 (82):1–6, 2021.
URL: `http://jmlr.org/papers/v22/20-825.html`.

[5] Wasif Altaf, Muhammad Shahbaz, and Aziz Guergachi. Applications of association rule mining in health informatics: a survey. *Artificial Intelligence Review*, 47(3):313–340, 2017.

[6] Dana Angluin, Michael Frazier, and Leonard Pitt. Learning conjunctions of horn clauses. *Mach. Learn.*, 9:147–164, 1992.

[7] Michael K Bergman. A common sense view of knowledge graphs, Aug 2019.
URL: https://www.mkbergman.com/2244/a-common-sense-view-of-knowledge-graphs/.

[8] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

[9] Piero Andrea Bonatti, Stefan Decker, Axel Polleres, and Valentina Presutti. Knowledge graphs: New directions for knowledge representation on the semantic web (dagstuhl seminar 18371). In *Dagstuhl Reports*, volume 8. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[10] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.
URL: https://proceedings.neurips.cc/paper/2013/file/1cecc7a77928ca8133fa24680a88d2f9-Paper.pdf.

[11] Antoine Bordes, Xavier Glorot, Jason Weston, and Yoshua Bengio. A semantic matching energy function for learning with multi-relational data. *Machine Learning*, 94(2):233–259, 2014.

[12] Ronald J Brachman and James G Schmolze. An overview of the kl-one knowledge representation system. *Readings in artificial intelligence and databases*, pages 207–230, 1989.

[13] Samuel Broscheit, Daniel Ruffinelli, Adrian Kochsiek, Patrick Betz, and Rainer Gemulla. LibKGE - A knowledge graph embedding library for reproducible research. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 165–174, 2020.
URL: https://www.aclweb.org/anthology/2020.emnlp-demos.22.

[14] Chandrahas, Aditya Sharma, and Partha Talukdar. Towards understanding the geometry of knowledge graph embeddings. In *Proceedings of the 56th Annual Meeting of the*

Association for Computational Linguistics (Volume 1: Long Papers), pages 122–131, Melbourne, Australia, July 2018. Association for Computational Linguistics.
URL: https://www.aclweb.org/anthology/P18-1012.

[15] Sophie-Grace Chappell. Plato on knowledge in the theaetetus. 2005.

[16] Lihan Chen, Sihang Jiang, Jingping Liu, Chao Wang, Sheng Zhang, Chenhao Xie, Jiaqing Liang, Yanghua Xiao, and Rui Song. Rule mining over knowledge graphs via reinforcement learning. *Knowledge-Based Systems*, 242:108371, 2022.

[17] Yang Chen, Sean Goldberg, Daisy Zhe Wang, and Soumitra Siddharth Johri. Ontological pathfinding. In *Proceedings of the 2016 International Conference on Management of Data*, pages 835–846, 2016.

[18] Edgar F Codd. A relational model of data for large shared data banks. In *Software pioneers*, pages 263–294. Springer, 2002.

[19] Luca Costabello, Sumit Pai, Chan Le Van, Rory McGrath, Nicholas McCarthy, and Pedro Tabacof. AmpliGraph: a Library for Representation Learning on Knowledge Graphs, March 2019.
URL: https://doi.org/10.5281/zenodo.2595043.

[20] Luca Costabello, Sumit Pai, Chan Le Van, Rory McGrath, Nicholas McCarthy, and Pedro Tabacof. Models — AmpliGraph 1.4.0 documentation, 2019.
URL: https://docs.ampligraph.org/en/1.4.0/ampligraph.latent_features.html#optimizers.

[21] Luca Costabello, Sumit Pai, Nicholas McCarthy, and Adrianna Janik. Knowledge graph embeddings tutorial: From theory to practice, September 2020.
URL: https://doi.org/10.5281/zenodo.4268208. https://kge-tutorial-ecai2020.github.io/.

[22] Yuanfei Dai, Shiping Wang, Neal N Xiong, and Wenzhong Guo. A survey on knowledge graph embedding: Approaches, applications and benchmarks. *Electronics*, 9(5):750, 2020.

[23] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[24] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 601–610, 2014.

[25] Stephen HC DuToit, A Gert W Steyn, and Rolf H Stumpf. *Graphical exploratory data analysis*. Springer Science & Business Media, 2012.

[26] Takuma Ebisu and Ryutaro Ichise. Graph pattern entity ranking model for knowledge graph completion. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 988–997, 2019.

[27] EdoardoRamalli. File:knowledgegraphembedding.png — wikimedia commons, the free media repository, 2021.
URL: `https://commons.wikimedia.org/w/index.php?title=File:KnowledgeGraphEmbedding.png&oldid=569452478`. [Online; accessed 24-March-2022].

[28] Lisa Ehrlinger and Wolfram Wöß. Towards a definition of knowledge graphs. *SEMANTiCS (Posters, Demos, SuCCESS)*, 48:1–4, 2016.

[29] Ingo Feinerer and Kurt Hornik. *wordnet: WordNet Interface*, 2020.
URL: `https://CRAN.R-project.org/package=wordnet`.

[30] Christiane Fellbaum. Wordnet. In *Theory and applications of ontology: computer applications*, pages 231–243. Springer, 2010.

[31] Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M Suchanek. Fast rule mining in ontological knowledge bases with amie+. *The VLDB Journal*, 24(6):707–730, 2015.

[32] Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian Suchanek. Amie: association rule mining under incomplete evidence in ontological knowledge bases. In *Proceedings of the 22nd international conference on World Wide Web*, pages 413–422, 2013.

[33] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[34] Niels Gottschalk-Mazouz. Internet and the flow of knowledge: Which ethical and political challenges will we face? 2008.

[35] Shu Guo, Quan Wang, Lihong Wang, Bin Wang, and Li Guo. Jointly embedding knowledge graphs and logical rules. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pages 192–202, 2016.

[36] Shu Guo, Quan Wang, Lihong Wang, Bin Wang, and Li Guo. Knowledge graph embedding with iterative guidance from soft rules. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[37] Johannes Hoffart, Fabian M Suchanek, Klaus Berberich, and Gerhard Weikum. Yago2: A spatially and temporally enhanced knowledge base from wikipedia. *Artificial intelligence*, 194:28–61, 2013.

[38] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, et al. Knowledge graphs. *Synthesis Lectures on Data, Semantics, and Knowledge*, 12(2):1–257, 2021.

[39] Rebecca Jahn. *Reasoning in Knowledge Graphs: Methods and Techniques*. PhD thesis, Wien, 2021.

[40] Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. Knowledge graph embedding via dynamic mapping matrix. In *Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing (volume 1: Long papers)*, pages 687–696, 2015.

[41] Rudolf Kadlec, Ondřej Bajgar, and Jan Kleindienst. Knowledge base completion: Baselines strike back. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 69–74, 2017.

[42] Seyed Mehran Kazemi and David Poole. Simple embedding for link prediction in knowledge graphs. *Advances in neural information processing systems*, 31, 2018.

[43] Richard L Kirkham. Does the gettier problem rest on a mistake? *Mind*, 93(372): 501–513, 1984.

[44] Jonathan Lajus, Luis Galárraga, and Fabian Suchanek. Fast and exact rule mining with amie 3. In *European Semantic Web Conference*, pages 36–52. Springer, 2020.

[45] Ni Lao, Tom Mitchell, and William Cohen. Random walk inference and learning in a large scale knowledge base. In *Proceedings of the 2011 conference on empirical methods in natural language processing*, pages 529–539, 2011.

[46] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic web*, 6(2):167–195, 2015.

[47] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

[48] Weiyang Lin, Sergio A Alvarez, and Carolina Ruiz. Efficient adaptive-support association rule mining for recommender systems. *Data mining and knowledge discovery*, 6(1): 83–105, 2002.

[49] Yankai Lin, Zhiyuan Liu, Huanbo Luan, Maosong Sun, Siwei Rao, and Song Liu. Modeling relation paths for representation learning of knowledge bases. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 705–714, 2015.

[50] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

[51] Jiangtao Ma, Yaqiong Qiao, Guangwu Hu, Yanjun Wang, Chaoqin Zhang, Yongzhong Huang, Arun Kumar Sangaiah, Huaiguang Wu, Hongpo Zhang, and Kai Ren. Elpkg: a high-accuracy link prediction approach for knowledge graph completion. *Symmetry*, 11 (9):1096, 2019.

[52] Christian Meilicke, Manuel Fink, Yanjie Wang, Daniel Ruffinelli, Rainer Gemulla, and Heiner Stuckenschmidt. Fine-grained evaluation of rule-and embedding-based systems for knowledge graph completion. In *International semantic web conference*, pages 3–20. Springer, 2018.

[53] Christian Meilicke, Melisachew Wudage Chekol, Manuel Fink, and Heiner Stuckenschmidt. Reinforced anytime bottom up rule learning for knowledge graph completion. *arXiv preprint arXiv:2004.04412*, 2020.

[54] Christian Meilicke, Patrick Betz, and Heiner Stuckenschmidt. Why a naive way to combine symbolic and latent knowledge base completion works surprisingly well. In *3rd Conference on Automated Knowledge Base Construction*, 2021.

[55] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[56] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

[57] Amihai Motro. Integrity= validity+ completeness. *ACM Transactions on Database Systems (TODS)*, 14(4):480–502, 1989.

[58] Francisco J. Navarrete and Antonio Vallecillo. Introducing subjective knowledge graphs. In *25th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2021, Gold Coast, Australia, October 25-29, 2021*, pages 61–70. IEEE, 2021.

[59] Dai Quoc Nguyen, Tu Dinh Nguyen, Dai Quoc Nguyen, and Dinh Phung. A novel embedding model for knowledge base completion based on convolutional neural network. In *NAACL HLT 2018: 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies-Proceedings of the Conference*, pages 327–333. Association for Computational Linguistics, 2018.

[60] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *Icml*, 2011.

[61] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1): 11–33, 2015.

[62] Maximilian Nickel, Lorenzo Rosasco, and Tomaso Poggio. Holographic embeddings of knowledge graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[63] Pouya Ghiasnezhad Omran, Kewen Wang, and Zhe Wang. Scalable rule learning via learning representation. In *IJCAI*, pages 2149–2155, 2018.

[64] Pouya Ghiasnezhad Omran, Kewen Wang, and Zhe Wang. An embedding-based approach to rule learning in knowledge graphs. *IEEE Transactions on Knowledge and Data Engineering*, 33(4):1348–1359, 2019.

[65] Stefano Ortona, Venkata Vamsikrishna Meduri, and Paolo Papotti. Robust discovery of positive and negative rules in knowledge bases. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 1168–1179. IEEE, 2018.

[66] Thomas Pellissier Tanon, Gerhard Weikum, and Fabian Suchanek. Yago 4: A reasonable knowledge base. In *European Semantic Web Conference*, pages 583–596. Springer, 2020.

[67] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pages 593–607. Springer, 2018.

[68] Olivier Sigaud and Stewart W Wilson. Learning classifier systems: a survey. *Soft Computing*, 11(11):1065–1078, 2007.

[69] Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. Reasoning with neural tensor networks for knowledge base completion. *Advances in neural information processing systems*, 26, 2013.

[70] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *International Conference on Learning Representations*, 2018.

[71] Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In *Proceedings of the 3rd workshop on continuous vector space models and their compositionality*, pages 57–66, 2015.

[72] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *International conference on machine learning*, pages 2071–2080. PMLR, 2016.

[73] Wikidata user "Sannita". Wikidata:Requests for comment/Kinship - Wikidata, 2013. **URL:** `https://www.wikidata.org/w/index.php?title=Wikidata:Requests_for_comment/Kinship&oldid=1102557673`.

[74] Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha Talukdar. Composition-based multi-relational graph convolutional networks. *arXiv preprint arXiv:1911.03082*, 2019.

[75] Xiaozhi Wang, Tianyu Gao, Zhaocheng Zhu, Zhengyan Zhang, Zhiyuan Liu, Juanzi Li, and Jian Tang. Kepler: A unified model for knowledge embedding and pre-trained language representation. *Transactions of the Association for Computational Linguistics*, 9:176–194, 2021.

[76] Yanjie Wang, Rainer Gemulla, and Hui Li. On multi-relational link prediction with bilinear models. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[77] Yanjie Wang, Daniel Ruffinelli, Rainer Gemulla, Samuel Broscheit, and Christian Meilicke. On evaluating embedding models for knowledge base completion. In *RepL4NLP@ ACL*, 2019.

[78] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, 2014.

[79] Sholom M Weiss and Nitin Indurkhya. Rule-based machine learning methods for functional prediction. *Journal of Artificial Intelligence Research*, 3:383–403, 1995.

[80] Robert West, Evgeniy Gabrilovich, Kevin Murphy, Shaohua Sun, Rahul Gupta, and Dekang Lin. Knowledge base completion via search-based question answering. In *Proceedings of the 23rd international conference on World wide web*, pages 515–526, 2014.

[81] Ludwig Wittgenstein, Gertrude Elizabeth Margaret Anscombe, Georg Henrik von Wright, Denis Paul, and Gertrude Elizabeth Margaret Anscombe. *On certainty*, volume 174. Blackwell Oxford, 1969.

[82] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*, 2014.

[83] Jing Zhang, Bo Chen, Lingxi Zhang, Xirui Ke, and Haipeng Ding. Neural, symbolic and neural-symbolic reasoning on knowledge graphs. *AI Open*, 2:14–35, 2021.

[84] Wen Zhang, Bibek Paudel, Liang Wang, Jiaoyan Chen, Hai Zhu, Wei Zhang, Abraham Bernstein, and Huajun Chen. Iteratively learning embeddings and rules for knowledge graph reasoning. In *The World Wide Web Conference*, pages 2366–2377, 2019.

Appendix A

Original rules

Rule	PCA Conf.	Found
has_part(x, z) ∧ hypernym(y, z) ⇒ has_part(x, y)	0.104	False
DRF(x, z) ∧ SDT(z, y) ⇒ SDT(x, y)	0.493	True
DRF(z, x) ∧ SDT(z, y) ⇒ SDT(x, y)	0.493	True
has_part(x, z) ∧ IH(y, z) ⇒ has_part(x, y)	0.533	True
hypernym(z, x) ∧ SDT(z, y) ⇒ SDT(x, y)	0.727	True
has_part(z, x) ∧ SDT(z, y) ⇒ SDT(x, y)	0.772	True
has_part(x, z) ∧ SDT(z, y) ⇒ SDT(x, y)	0.810	True
hypernym(x, z) ∧ SDT(z, y) ⇒ SDT(x, y)	0.829	True
IH(x, z) ∧ SDT(z, y) ⇒ SDT(x, y)	0.887	True
DRF(y, x) ⇒ DRF(x, y)	1.000	True

Table A.1: *All rules mined with AMIE3 from the original WN18RR KG, sorted by their calculated PCA confidence in ascending order. The rightmost column indicates whether this rule was found in at least one of the KG extensions made with the RandomBaseline KGE. As the table shows, the rule not found after extending the KG in such a way is that with the lowest PCA confidence.* ***DRF*** *= derivationally_related_form,* ***IH*** *= instance_hypernym and* ***SDT*** *= synset_domain_topic of.*

Rule	PCA Confidence	Found
relative(z, y) ∧ relative(x, z) ⇒ relative(x, y)	0.100	False
father(x, z) ∧ sibling(z, y) ⇒ relative(x, y)	0.101	False
mother(z, x) ∧ relative(y, z) ⇒ relative(x, y)	0.101	False

child(x, z) ∧ spouse(z, y) ⇒ relative(x, y)	0.102	False
child(x, z) ∧ spouse(y, z) ⇒ relative(x, y)	0.102	False
mother(x, z) ∧ relative(y, z) ⇒ relative(x, y)	0.104	False
father(z, y) ∧ father(x, z) ⇒ relative(x, y)	0.109	False
relative(y, z) ∧ sibling(x, z) ⇒ relative(x, y)	0.120	True
relative(y, z) ∧ sibling(z, x) ⇒ relative(x, y)	0.120	True
father(z, x) ∧ spouse(z, y) ⇒ relative(x, y)	0.122	False
child(x, z) ∧ relative(z, y) ⇒ relative(x, y)	0.124	True
father(z, x) ∧ spouse(y, z) ⇒ relative(x, y)	0.124	False
relative(z, y)∧ sibling(z, x) ⇒ relative(x, y)	0.127	True
relative(z, y) ∧ sibling(x, z) ⇒ relative(x, y)	0.128	True
child(z, y) ∧ child(x, z) ⇒ relative(x, y)	0.131	True
father(z, x) ∧ relative(z, y) ⇒ relative(x, y)	0.141	True
child(x, z) ∧ relative(y, z) ⇒ relative(x, y)	0.148	True
child(x, z) ∧ father(y, z) ⇒ relative(x, y)	0.154	True
child(z, y) ∧ father(z, x) ⇒ relative(x, y)	0.167	True
father(z, x) ∧ relative(y, z) ⇒ relative(x, y)	0.179	True
father(z, x) ∧ father(y, z) ⇒ relative(x, y)	0.196	True
relative(z, x) ∧ spouse(z, y) ⇒ relative(x, y)	0.206	True
relative(z, x) ∧ spouse(y, z) ⇒ relative(x, y)	0.206	True
mother(z, y) ∧ relative(z, x) ⇒ relative(x, y)	0.224	True
mother(y, z) ∧ relative(z, x) ⇒ relative(x, y)	0.242	True
child(y, z) ∧ relative(z, x) ⇒ relative(x, y)	0.273	True
child(z, y) ∧ relative(z, x) ⇒ relative(x, y)	0.275	True
mother(z, y) ∧ relative(x, z) ⇒ relative(x, y)	0.284	True
father(y, z) ∧ relative(z, x) ⇒ relative(x, y)	0.294	True
father(z, y) ∧ relative(z, x) ⇒ relative(x, y)	0.301	True
relative(x, z) ∧ spouse(y, z) ⇒ relative(x, y)	0.305	True
relative(x, z) ∧ spouse(z, y) ⇒ relative(x, y)	0.305	True
child(y, z) ∧ relative(x, z) ⇒ relative(x, y)	0.343	True
relative(z, x) ∧ sibling(y, z) ⇒ relative(x, y)	0.349	True
child(z, x) ∧ spouse(z, y) ⇒ father(x, y)	0.350	True
mother(y, z) ∧ relative(x, z) ⇒ relative(x, y)	0.350	True
relative(z, x)∧ sibling(z, y) ⇒ relative(x, y)	0.351	True

child(z, x) ∧ spouse(y, z) ⇒ father(x, y)	0.351	True
child(z, y) ∧relative(x, z) ⇒ relative(x, y)	0.364	True
father(z, y) ∧ relative(x, z) ⇒ relative(x, y)	0.372	True
father(y, z) ∧ relative(x, z) ⇒ relative(x, y)	0.378	True
child(z, x) ∧ spouse(y, z) ⇒ mother(x, y)	0.382	True
child(z, x) ∧ spouse(z, y) ⇒ mother(x, y)	0.386	True
child(y, z) ∧ sibling(z, x) ⇒ mother(x, y)	0.422	True
child(y, z) ∧ sibling(x, z) ⇒ mother(x, y)	0.423	True
child(y, z)∧ father(z, x) ⇒ spouse(x, y)	0.470	True
child(x, z) ∧ father(z, y) ⇒ spouse(x, y)	0.472	True
child(x, z)∧ child(y, z) ⇒ spouse(x, y)	0.472	True
child(x, z) ∧ mother(z, y) ⇒ spouse(x, y)	0.478	True
child(y, z) ∧ mother(z, x) ⇒ spouse(x, y)	0.481	True
relative(x, z) ∧ sibling(z, y) ⇒ relative(x, y)	0.483	True
relative(x, z) ∧ sibling(y, z) ⇒ relative(x, y)	0.485	True
child(y, x) ⇒ mother(x, y)	0.535	True
sibling(z, x) ∧ sibling(z, y) ⇒ sibling(x, y)	0.565	True
child(y, z) ∧ sibling(z, x) ⇒ father(x, y)	0.584	True
child(y, z) ∧ sibling(x, z) ⇒ father(x, y)	0.584	True
sibling(z, x) ∧ sibling(y, z) ⇒ sibling(x, y)	0.585	True
sibling(z, y) ∧ sibling(x, z) ⇒ sibling(x, y)	0.588	True
father(y, z) ∧ spouse(x, z) ⇒ child(x, y)	0.588	True
father(y, z) ∧ spouse(z, x) ⇒ child(x, y)	0.589	True
sibling(x, z) ∧ sibling(y, z) ⇒ sibling(x, y)	0.590	True
father(x, z) ∧ father(y, z) ⇒ sibling(x, y)	0.659	True
child(z, y) ∧ father(x, z) ⇒ sibling(x, y)	0.662	True
child(z, x) ∧ child(z, y) ⇒ sibling(x, y)	0.665	True
child(z, x) ∧ father(y, z) ⇒ sibling(x, y)	0.668	True
child(z, y) ∧ spouse(z, x) ⇒ child(x, y)	0.688	True
child(z, y) ∧ spouse(x, z) ⇒ child(x, y)	0.689	True
father(x, z) ∧ spouse(y, z) ⇒ mother(x, y)	0.702	True
mother(z, x) ∧ sibling(z, y) ⇒ child(x, y)	0.703	True
mother(z, x) ∧ sibling(y, z) ⇒ child(x, y)	0.704	True
mother(x, z) ∧ mother(y, z) ⇒ sibling(x, y)	0.707	True

father(x, z) ∧ spouse(z, y) ⇒ mother(x, y)	0.709	True
child(z, y) ∧ mother(x, z) ⇒ sibling(x, y)	0.709	True
child(z, x) ∧ mother(y, z) ⇒ sibling(x, y)	0.710	True
child(y, x) ⇒ father(x, y)	0.770	True
mother(z, y) ∧ sibling(z, x) ⇒ mother(x, y)	0.787	True
mother(z, y) ∧ sibling(x, z) ⇒ mother(x, y)	0.788	True
mother(y, z) ∧ spouse(x, z) ⇒ child(x, y)	0.838	True
mother(y, z) ∧ spouse(z, x) ⇒ child(x, y)	0.840	True
relative(y, x) ⇒ relative(x, y)	0.846	True
child(x, z) ∧ sibling(z, y) ⇒ child(x, y)	0.851	True
child(x, z) ∧ sibling(y, z) ⇒ child(x, y)	0.851	True
mother(x, z) ∧ spouse(y, z) ⇒ father(x, y)	0.868	True
mother(x, z)∧ spouse(z, y) ⇒ father(x, y)	0.873	True
father(z, x) ∧ mother(z, y) ⇒ spouse(x, y)	0.953	True
father(z, x) ∧ sibling(z, y) ⇒ child(x, y)	0.954	True
father(z, x) ∧ sibling(y, z) ⇒ child(x, y)	0.955	True
father(z, y) ∧ sibling(x, z) ⇒ father(x, y)	0.961	True
father(z, y) ∧ sibling(z, x) ⇒ father(x, y)	0.961	True
father(z, y) ∧ mother(z, x) ⇒ spouse(x, y)	0.968	True
sibling(y, x) ⇒ sibling(x, y)	0.990	True
mother(y, x) ⇒ child(x, y)	0.992	True
spouse(y, x) ⇒ spouse(x, y)	0.992	True
father(y, x) ⇒ child(x, y)	0.998	True

Table A.2: *All rules mined with AMIE3 from the original family KG, sorted by their calculated PCA confidence in ascending order. The rightmost column indicates whether this rule was found in at least one of the KG extensions made with the RandomBaseline KGE. As the table shows, the rules not found after extending the KG in such a way were among those with the lowest PCA confidence.*

Appendix B

Results

child	father	mother	relative	sibling	spouse
0.009	-0.010	-0.010	-0.001	0.000	0.000
0.002	-0.002	-0.002	0.000	0.000	0.000
-0.001	0.001	0.002	0.001	0.000	0.000
-0.005	0.005	0.005	0.000	0.000	0.000
-0.012	0.012	0.012	0.001	0.000	0.001
-0.007	0.008	0.007	0.001	0.000	-0.001
0.006	-0.006	-0.006	-0.001	0.000	0.000
-0.004	0.005	0.004	0.001	0.000	0.000
0.003	-0.002	-0.002	0.001	0.000	0.000
0.016	-0.016	-0.015	0.000	0.000	0.000
-0.002	0.002	0.002	0.000	0.000	0.000
-0.001	0.001	0.001	0.001	0.000	0.000
0.006	-0.005	-0.006	-0.001	0.000	0.000
0.003	-0.004	-0.004	0.000	0.000	0.000
-0.001	0.001	0.002	-0.001	0.000	0.000
-0.013	0.014	0.013	-0.001	0.000	0.000
0.008	-0.008	-0.008	0.000	0.000	0.000
0.006	-0.006	-0.006	-0.002	0.000	0.000
-0.013	0.013	0.013	0.000	0.000	0.000
-0.009	0.009	0.009	0.000	0.000	0.000
-0.005	0.005	0.005	0.001	0.000	0.000
-0.007	0.007	0.007	0.001	0.000	0.000

0.001	-0.001	-0.001	0.000	0.000	0.001
0.009	-0.009	-0.008	-0.001	0.000	0.000
-0.003	0.003	0.003	0.000	0.000	0.000
0.003	-0.003	-0.003	0.000	0.000	0.000
-0.014	0.014	0.014	0.001	0.000	0.001
0.005	-0.006	-0.005	0.000	0.000	0.001
-0.006	0.005	0.005	0.000	0.000	0.001
0.000	0.000	0.000	0.001	0.000	0.000
-0.003	0.003	0.003	0.000	0.000	0.000
-0.003	0.003	0.004	0.001	0.000	0.000
-0.009	0.009	0.009	-0.001	0.000	0.000
-0.011	0.011	0.010	0.001	0.000	0.000
-0.001	0.001	0.001	-0.001	0.000	0.000
-0.009	0.009	0.008	-0.001	0.000	-0.001
0.006	-0.006	-0.006	-0.001	0.000	0.000
-0.002	0.002	0.002	0.002	0.000	0.000
0.002	-0.002	-0.002	0.000	0.000	0.000
0.005	-0.005	-0.005	-0.002	0.000	0.000
-0.005	0.005	0.004	0.001	0.000	0.000
0.007	-0.007	-0.006	0.001	0.000	0.000
0.004	-0.004	-0.003	0.000	0.000	0.000
0.005	-0.006	-0.006	0.001	0.000	0.000
-0.005	0.006	0.005	0.001	0.000	0.001
0.007	-0.008	-0.008	-0.001	0.000	0.000
0.005	-0.005	-0.004	0.000	0.000	-0.001
0.002	-0.002	-0.002	-0.001	0.000	0.000
0.003	-0.003	-0.002	-0.001	0.000	0.001
0.007	-0.007	-0.007	0.001	0.000	0.000

Table B.1: *TransE's 50-dimensional embedding vectors for the different relations in the family dataset, rounded to the third decimal. This table shows that the vectors for all relations are close to the zero-vector.*

DRF	**has_part**	**hypernym**	**IH**	**MM**	**SDT**
0.000	-0.002	-0.003	-0.004	-0.027	-0.028

0.000	0.000	0.012	0.081	-0.001	0.010
0.000	-0.003	0.007	0.010	-0.012	0.009
0.000	-0.003	0.003	-0.148	-0.042	0.000
0.000	0.017	0.003	0.005	0.002	0.007
0.000	-0.001	0.002	0.139	-0.014	0.003
0.000	-0.002	-0.002	-0.115	0.031	-0.006
0.000	-0.006	0.005	0.005	-0.015	0.008
0.000	-0.001	0.001	0.150	-0.021	0.010
0.000	-0.007	-0.011	0.119	0.012	0.019
0.000	-0.004	0.013	0.118	-0.019	0.012
0.000	-0.004	-0.002	-0.069	0.042	-0.001
0.000	-0.002	0.002	-0.165	-0.025	0.002
0.000	0.000	-0.012	-0.017	0.007	-0.012
0.000	-0.018	-0.002	-0.111	0.008	0.015
0.000	0.021	0.001	0.006	0.003	-0.041
0.000	0.001	-0.015	0.005	0.043	-0.026
0.000	-0.002	0.005	-0.152	-0.017	-0.006
0.000	0.002	0.002	-0.082	-0.002	-0.009
0.000	-0.002	0.001	0.134	-0.048	0.044
0.000	-0.001	0.007	0.142	-0.003	0.016
0.000	-0.001	-0.001	-0.134	-0.060	-0.009
0.000	0.018	-0.003	-0.142	-0.020	-0.016
0.000	0.002	-0.008	-0.084	0.010	-0.030
0.000	-0.004	0.009	-0.062	-0.007	-0.019
0.000	0.005	0.005	0.067	0.000	0.011
0.000	-0.003	0.005	0.113	-0.009	0.028
0.000	0.002	-0.005	0.002	-0.038	-0.010
0.000	0.007	0.001	-0.127	0.000	-0.010
0.000	-0.002	0.000	0.036	-0.007	0.013
0.001	-0.005	0.002	0.002	0.009	-0.010
0.000	-0.004	0.004	-0.075	-0.045	0.017
-0.001	0.016	0.001	0.110	0.018	-0.024
0.000	0.009	0.000	0.156	-0.002	-0.020
0.001	0.000	0.007	0.023	-0.043	-0.001

0.000	-0.002	-0.003	-0.103	0.001	0.000
0.000	-0.008	0.007	0.167	-0.005	0.019
0.000	0.000	-0.017	0.021	0.012	-0.020
0.000	-0.007	-0.003	-0.001	0.025	-0.002
0.000	-0.008	-0.006	0.029	0.047	0.006
-0.001	0.002	0.000	-0.001	0.047	-0.004
0.000	-0.002	0.014	0.139	-0.036	0.005
0.000	0.001	0.010	0.010	-0.056	0.003
0.000	-0.001	0.001	-0.005	0.000	0.019
0.000	-0.002	-0.001	-0.128	0.005	0.007
0.000	-0.001	0.002	0.131	0.001	-0.002
0.000	-0.004	0.011	0.137	-0.008	0.024
0.000	0.005	-0.002	-0.092	-0.045	-0.006
0.000	0.001	0.011	-0.058	-0.005	-0.023
0.000	0.004	-0.013	-0.041	0.021	-0.046

Table B.2: *TransE's 50-dimensional embedding vectors for the different relations in the WN18RR dataset, rounded to the third decimal. This table shows that the vectors for all relations are close to the zero-vector.* ***DRF*** *= derivationally_related_form,* ***IH*** *= instance_hypernym,* ***MM*** *= member_meronym and* ***SDT*** *= synset_domain_topic of.*

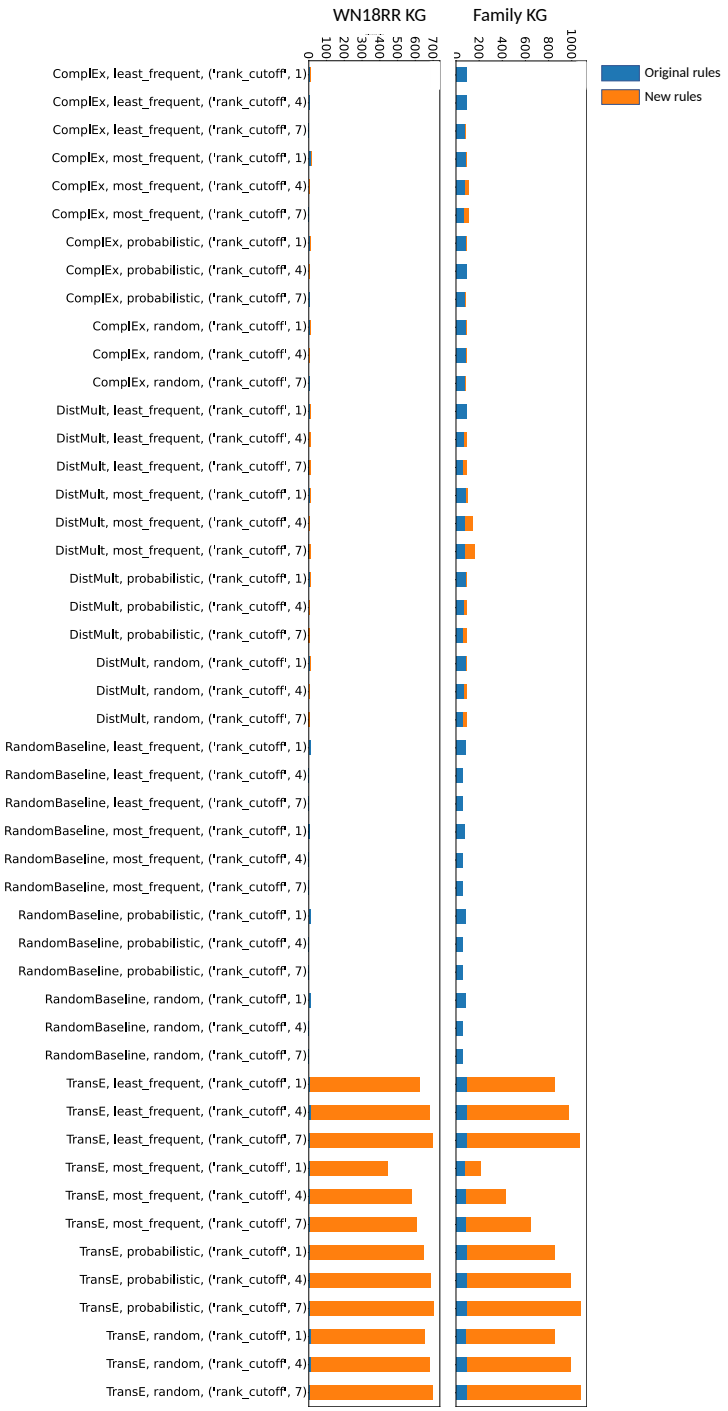


Figure B.1: *Distribution of all mined rule sets. Each rule set is labeled with the entity selection strategy, embedding model, and rank cutoff used for that particular KG extension. As mentioned in section 4.2.2, we can see that when the KGs are extended using TransE, the number of new rules that are mined increases drastically.*

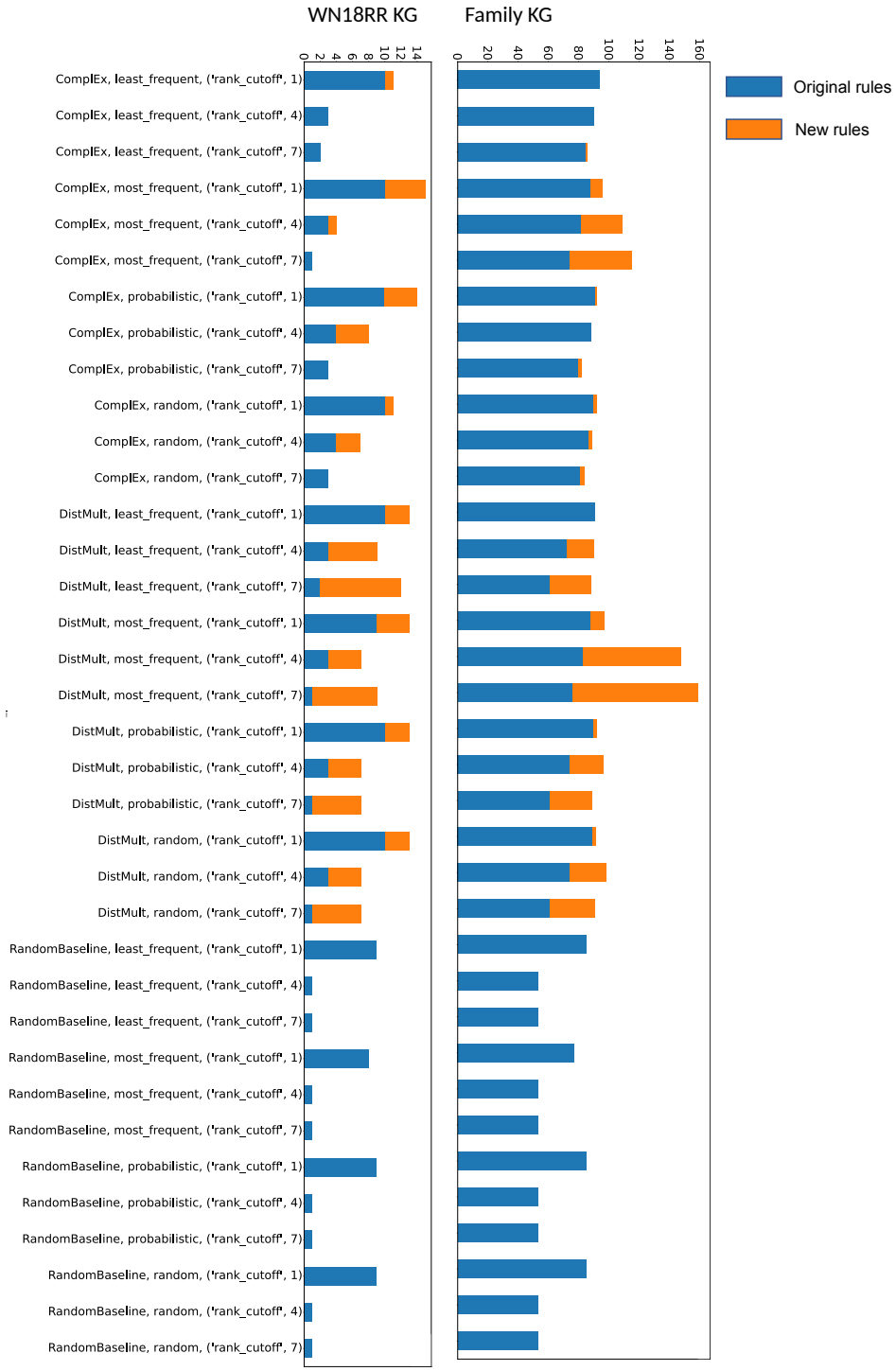


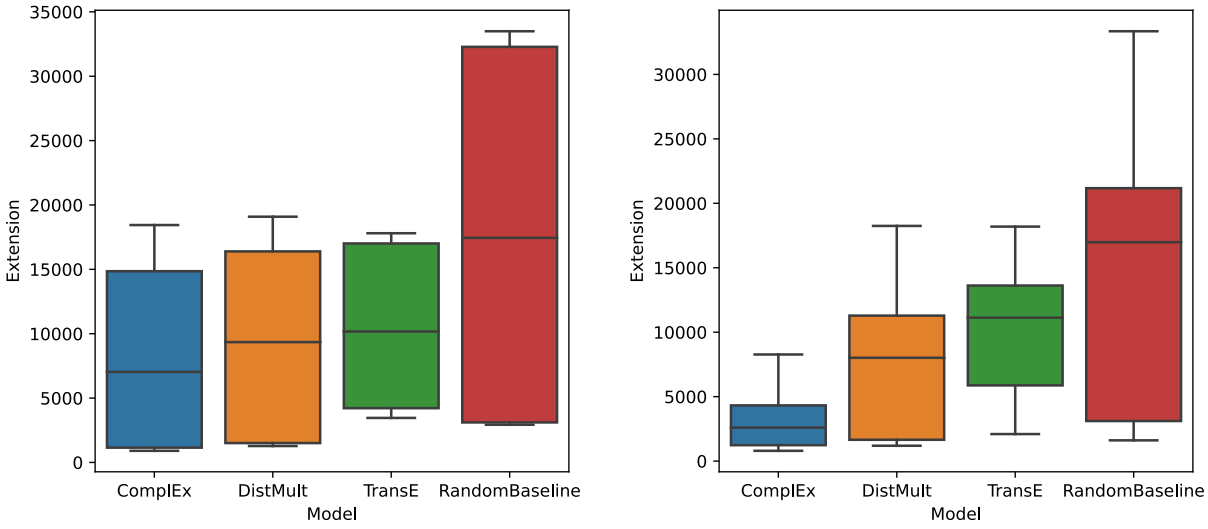
Figure B.2: *Distribution of all mined rule sets, excluding the sets mined from KGs extended using TransE. Each rule set is labeled with the entity selection strategy, embedding model, and rank cutoff used for that particular KG extension. We can observe that no new rules were mined when the RandomBaseline was used to extend the KGs.*

Rule	Frequencies	PCA Confidence
father(y, x) ⇒ child(x, y)	48	0.998071
spouse(y, x) ⇒ spouse(x, y)	48	0.992355
mother(y, x) ⇒ child(x, y)	48	0.991906
sibling(y, x) ⇒ sibling(x, y)	48	0.989634
father(z, y) ∧ mother(z, x) ⇒ spouse(x, y)	48	0.967952
father(z, y) ∧ sibling(z, x) ⇒ father(x, y)	48	0.961199
father(z, y) ∧ sibling(x, z) ⇒ father(x, y)	48	0.961023
father(z, x) ∧ sibling(y, z) ⇒ child(x, y)	48	0.954603
father(z, x) ∧ sibling(z, y) ⇒ child(x, y)	48	0.954328
father(z, x) ∧ mother(z, y) ⇒ spouse(x, y)	48	0.953170
mother(x, z) ∧ spouse(z, y) ⇒ father(x, y)	48	0.872686
mother(x, z) ∧ spouse(y, z) ⇒ father(x, y)	48	0.867655
child(x, z) ∧ sibling(y, z) ⇒ child(x, y)	48	0.851012
child(x, z) ∧ sibling(z, y) ⇒ child(x, y)	48	0.850519
relative(y, x) ⇒ relative(x, y)	48	0.846306
mother(y, z) ∧ spouse(z, x) ⇒ child(x, y)	48	0.839947
mother(y, z) ∧ spouse(x, z) ⇒ child(x, y)	48	0.838288
mother(z, y) ∧ sibling(x, z) ⇒ mother(x, y)	48	0.787816
mother(z, y) ∧ sibling(z, x) ⇒ mother(x, y)	48	0.787401
child(y, x) ⇒ father(x, y)	48	0.769746
child(z, x) ∧ mother(y, z) ⇒ sibling(x, y)	48	0.709805
child(z, y) ∧ mother(x, z) ⇒ sibling(x, y)	48	0.709139
father(x, z) ∧ spouse(z, y) ⇒ mother(x, y)	48	0.708957
mother(x, z) ∧ mother(y, z) ⇒ sibling(x, y)	48	0.707149
mother(z, x) ∧ sibling(y, z) ⇒ child(x, y)	48	0.704016
mother(z, x) ∧ sibling(z, y) ⇒ child(x, y)	48	0.703385
father(x, z) ∧ spouse(y, z) ⇒ mother(x, y)	48	0.701812
child(z, y) ∧ spouse(x, z) ⇒ child(x, y)	48	0.688723
child(z, y) ∧ spouse(z, x) ⇒ child(x, y)	48	0.687653
child(z, x) ∧ father(y, z) ⇒ sibling(x, y)	48	0.668446
child(z, x) ∧ child(z, y) ⇒ sibling(x, y)	48	0.664745
child(z, y) ∧ father(x, z) ⇒ sibling(x, y)	48	0.662206

father(x, z) ∧ father(y, z) ⇒ sibling(x, y)	48	0.658865
sibling(x, z) ∧ sibling(y, z) ⇒ sibling(x, y)	48	0.590059
father(y, z) ∧ spouse(z, x) ⇒ child(x, y)	48	0.588533
father(y, z) ∧ spouse(x, z) ⇒ child(x, y)	48	0.588257
sibling(z, y) ∧ sibling(x, z) ⇒ sibling(x, y)	48	0.588151
sibling(z, x) ∧ sibling(y, z) ⇒ sibling(x, y)	48	0.584857
child(y, z) ∧ sibling(x, z) ⇒ father(x, y)	48	0.584069
child(y, z) ∧ sibling(z, x) ⇒ father(x, y)	48	0.583590
sibling(z, x) ∧ sibling(z, y) ⇒ sibling(x, y)	48	0.564657
child(y, x) ⇒ mother(x, y)	48	0.535350
relative(x, z) ∧ sibling(y, z) ⇒ relative(x, y)	40	0.484528
relative(x, z) ∧ sibling(z, y) ⇒ relative(x, y)	40	0.483459
child(y, z) ∧ mother(z, x) ⇒ spouse(x, y)	48	0.480628
child(x, z) ∧ mother(z, y) ⇒ spouse(x, y)	48	0.477785
child(x, z) ∧ child(y, z) ⇒ spouse(x, y)	48	0.471973
child(x, z) ∧ father(z, y) ⇒ spouse(x, y)	48	0.471666
child(y, z) ∧ father(z, x) ⇒ spouse(x, y)	48	0.469716
child(y, z) ∧ sibling(x, z) ⇒ mother(x, y)	48	0.422559
child(y, z) ∧ sibling(z, x) ⇒ mother(x, y)	48	0.422363
child(z, x) ∧ spouse(z, y) ⇒ mother(x, y)	48	0.386296
child(z, x) ∧ spouse(y, z) ⇒ mother(x, y)	48	0.382227
father(y, z) ∧ relative(x, z) ⇒ relative(x, y)	37	0.378085
father(z, y) ∧ relative(x, z) ⇒ relative(x, y)	37	0.372340
child(z, y) ∧ relative(x, z) ⇒ relative(x, y)	37	0.364472
child(z, x) ∧ spouse(y, z) ⇒ father(x, y)	48	0.351479
relative(z, x) ∧ sibling(z, y) ⇒ relative(x, y)	40	0.351014
mother(y, z) ∧ relative(x, z) ⇒ relative(x, y)	36	0.350490
child(z, x) ∧ spouse(z, y) ⇒ father(x, y)	48	0.350117
relative(z, x) ∧ sibling(y, z) ⇒ relative(x, y)	40	0.348765
child(y, z) ∧ relative(x, z) ⇒ relative(x, y)	37	0.343180
relative(x, z) ∧ spouse(z, y) ⇒ relative(x, y)	37	0.305043
relative(x, z) ∧ spouse(y, z) ⇒ relative(x, y)	37	0.305043
father(z, y) ∧ relative(z, x) ⇒ relative(x, y)	37	0.301061
father(y, z) ∧ relative(z, x) ⇒ relative(x, y)	37	0.294304

mother(z, y) ∧ relative(x, z) ⇒ relative(x, y)	33	0.283544
child(z, y) ∧ relative(z, x) ⇒ relative(x, y)	37	0.275454
child(y, z) ∧ relative(z, x) ⇒ relative(x, y)	37	0.273356
mother(y, z) ∧ relative(z, x) ⇒ relative(x, y)	33	0.241645
mother(z, y) ∧ relative(z, x) ⇒ relative(x, y)	30	0.224490
relative(z, x) ∧ spouse(y, z) ⇒ relative(x, y)	35	0.206044
relative(z, x) ∧ spouse(z, y) ⇒ relative(x, y)	32	0.205761
father(z, x) ∧ father(y, z) ⇒ relative(x, y)	30	0.196078
father(z, x) ∧ relative(y, z) ⇒ relative(x, y)	33	0.179144
child(z, y) ∧ father(z, x) ⇒ relative(x, y)	26	0.166667
child(x, z) ∧ father(y, z) ⇒ relative(x, y)	27	0.153584
child(x, z) ∧ relative(y, z) ⇒ relative(x, y)	32	0.148287
father(z, x) ∧ relative(z, y) ⇒ relative(x, y)	30	0.140871
child(z, y) ∧ child(x, z) ⇒ relative(x, y)	25	0.130759
relative(z, y) ∧ sibling(x, z) ⇒ relative(x, y)	39	0.127978
relative(z, y) ∧ sibling(z, x) ⇒ relative(x, y)	39	0.126881
father(z, x) ∧ spouse(y, z) ⇒ relative(x, y)	21	0.124101
child(x, z) ∧ relative(z, y) ⇒ relative(x, y)	31	0.123643
father(z, x) ∧ spouse(z, y) ⇒ relative(x, y)	17	0.122083
relative(y, z) ∧ sibling(z, x) ⇒ relative(x, y)	38	0.120448
relative(y, z) ∧ sibling(x, z) ⇒ relative(x, y)	38	0.120360
father(z, y) ∧ father(x, z) ⇒ relative(x, y)	16	0.108774
mother(x, z) ∧ relative(y, z) ⇒ relative(x, y)	21	0.104492
child(x, z) ∧ spouse(y, z) ⇒ relative(x, y)	13	0.102020
child(x, z) ∧ spouse(z, y) ⇒ relative(x, y)	13	0.102020
mother(z, x) ∧ relative(y, z) ⇒ relative(x, y)	18	0.101064
father(x, z) ∧ sibling(z, y) ⇒ relative(x, y)	10	0.100971
relative(z, y) ∧ relative(x, z) ⇒ relative(x, y)	29	0.100440

Table B.3: *Rules mined from the original family KG, with their corresponding PCA confidences and how many times they were mined from the 48 extensions. Note that all rules with the relative predicate in the consequent have lower PCA confidence and were mined less frequently.*



(a) *WN18RR KG.* **(b)** *Family KG.*

Figure B.3: *Distribution of KG extension sizes over KGE models.*

Appendix C

Propositional logic

Propositional logic (PL) is a branch of logic that deals with statements, or "propositions", that can be either true or false. Propositions can be combined using boolean operators, for example, "and" $\wedge$ (conjunction) and "or" $\vee$ (disjunction). Atomic propositions are propositions without operators, and are also called *literals*. Literals are boolean variables v or their negation $\neg v$. For example, two literals v_1, v_2 can be combined into one proposition P with the "or" operator:

$$P = \neg v_1 \vee v_2$$

Now P evaluates to true if v_1 is evaluated to false or if v_2 is evaluated to true. The proposition P will only evaluate to false if v_1 is true *and* v_2 is false. A *clause* is a disjunction of literals, so P is a clause. The proposition P is also considered a *Horn clause*, which is a clause where at most one literal is not negated.

The implication operator is denoted by $\rightarrow$, and can be thought of as an "if-then" operator. The proposition $v_1 \rightarrow v_2$ can be read as "*if* v_1 *is true, then* v_2 *is true*". The statement is thus only falsified if v_1 is true and v_2 is false. If v_2 is true, then it does not matter what v_1 evaluates to, and similarly, if v_1 is false, then it does not matter what v_2 evaluates to. If we compare this to the circumstances under which P is evaluated to true or false, we also see that $P = v_1 \rightarrow v_2 = \neg v_1 \vee v_2$. Thus, Horn clauses can be treated as rules, where all the negated literals are in the antecedent (on the left side of the implication arrow), and the single non-negated literal is the consequent (on the right side of the implication arrow). For example the proposition

$$Q = v_1 \wedge v_3 \wedge v_4 \rightarrow v_2 = \neg v_1 \vee \neg v_3 \vee \neg v_4 \vee v_2$$

is also a Horn clause. The rule mining algorithm AMIE3 used in this thesis mined rules in the form of Horn clauses. One could also say that it mines a single *Horn sentence*, which is a conjunction of Horn clauses. A Horn sentence is true only if every Horn clause within is evaluated to true.

The truth value of propositions is determined by the *valuation* of the literals used. A valuation assigns truth values to each literal. For example, under the valuation

$$\mathcal{V}_+ = \{v_1 = false, v_2 = false, v_3 = true, v_4 = true\}$$

Q would be true, while under the valuation

$$\mathcal{V}_- = \{v_1 = true, v_2 = false, v_3 = true, v_4 = true\}$$

Q would be false.