Experiments & Analyses Paper

Evaluation of Sampling Methods for Discovering Facts from Knowledge Graph Embeddings

Rama Widyadhana Bhagaskoro
Technische Universität Berlin
Berlin, Germany
ramawidyadhana@gmail.com

Volker Markl
Technische Universität Berlin
DFKI GmbH
Berlin, Germany
volker.markl@tu-berlin.de

Zoi Kaoudi
IT University of Copenhagen
Copenhagen, Denmark
zoka@itu.dk

ABSTRACT

Knowledge graphs (KGs) are being used in many real-world application domains, ranging from search engines to biomedical data analysis. Even if there is a large corpus of KGs available, they are inherently incomplete due to the incompleteness of the sources based on which they were constructed. Knowledge graph embeddings (KGEs) is a very popular technique to complete KGs. However, they are only capable of answering true or false to a given fact. Thus, users need to provide a concrete query or some test data. Unfortunately, such queries or data are not always available. There are cases where users want to discover all (or as many as possible) missing facts from an input KG. Given a KGE model, users should thus provide to the KGE model candidate facts consisting of the complement of the KG. This is infeasible even for small graphs simply due to the size of the complement graph. In this paper, we define the problem of discovering missing facts from a given KGE model and refer to it as *fact discovery*. We study sampling methods to get candidate facts and then using KGEs to retrieve the most plausible ones. We extensively evaluate different existing sampling methods and provide guidelines on when each one of them is most suitable. We also discuss the challenges and limitations that we encountered when investigating the different techniques. With these insights, we expect to shed light and attract more researchers on this unexplored direction.

1 INTRODUCTION

Knowledge graphs (KGs) are an already established tool for many real-world applications, such as question answering, web search, and fact-checking. In addition, they are increasingly being used in domain-specific fields, such as bioinformatics and healthcare for precision medicine analysis and drug discovery among others [3, 10, 15, 26]. A knowledge graph consists of a set of facts (a.k.a. true triples) in the form of (s, r, o), where s and o are nodes in the graph and represent entities, while r is a labeled directed edge from s to o and denotes the relation between these entities.

Despite the large scale of available KGs (e.g., Wikidata, DBpedia, NELL), most of them are inherently incomplete, i.e., there are a lot of missing facts. This leads to loss of information and sub-optimal results in the aforementioned applications. A very popular and successful way to tackle this problem is using knowledge graph embeddings (KGEs). A KGE is a latent representation of entities and relations in a low dimensional continuous space which can be used for checking the plausibility of triples. In other words, KGEs are models trained on a given KG (training data). These models can be used to predict whether a triple is true or false. Similar to other machine learning models, once a KGE model is built, the only way it can be used is by providing it with some kind of inference or test data, i.e., a set of candidate facts in the case of KGs. In the literature, when evaluating KGE models, a given KG is split into training and test data, and the test data is used to evaluate the accuracy of the model by asking it to reply true or false (triple classification). In other cases, test data are formed into specific queries, i.e., triples where one of the entities is unknown, and are given to the evaluation process. The process first imputes the unknown entity with all entities that exist in the input KG and ranks the triples based on their plausibility. The plausibility of a triple being true is provided by the KGE model. This task is commonly referred to as *link prediction*.

However, there are cases where test data or specific queries are not readily available, a common situation in real-world applications such as biomedical research [25] where new facts in a KG need to be discovered. In such cases, a biomedical scientist using KGEs to discover relations between drugs, diseases, and proteins might not have specific queries to input into the model. While the scientist could employ link prediction to identify the disease that a specific drug targets, they may also want to uncover entirely new relationships or properties within the already defined entities of the graph. *Fact discovery*, i.e., finding triples that are true in a KG without any input, can be especially advantageous in such scenarios where the primary goal is to explore undiscovered connections, a common objective in fields such as biomedical research. Starting without predefined queries allows for a broader exploration of potential connections, facilitating exploratory analysis and serendipitous discoveries, aligning with the nature of scientific inquiry where investigations often pursue knowledge expansion rather than hypothesis confirmation. Thus, it becomes crucial for a biomedical scientist to use an existing KG and enable the KGE model to autonomously infer missing facts, which could lead to groundbreaking insights and innovations.

In this paper, we focus on the problem of fact discovery rather than link prediction. Our approach leverages the inherent structure and properties of the KGs to infer missing information, even in the absence of explicit queries. A naive way to achieve this is to follow an exhaustive approach: compose all non-existing triples, i.e., the complement of the KG, as candidate facts (*generation phase*) and pass them to the KGE model (*inference phase*). In the case of a very large KG, just constructing the complement of the graph can be extremely expensive, especially because the graph is directed and heterogeneous, i.e., different types of edges exist. For instance, assume a KG $\mathcal{G}$ with $\mathcal{E}$ the set of entities, $\mathcal{R}$ the set of relations, and $|\mathcal{G}|$ the total number of triples in $\mathcal{G}$. Then, the complement graph $\mathcal{G}'$ contains $|\mathcal{E}|^2 \times |\mathcal{R}| - |\mathcal{G}|$ edges. For a moderate size KG, such as YAGO3-10 [20] with approximately $120K$ entities and 37 relations, this number grows to 533×10^9 edges. Thus, enumerating all non-existing triples can take a significant amount of time depending on the size of the graph. To add to this, the invocation of the KGE model should happen for

each edge in $\mathcal{G}'$. Given that each call to a KGE model typically requires more than a couple of seconds [34], the inference phase for the YAGO3-10 would require thousands of years!

The problem of fact discovery from KGEs has been largely ignored in the literature. The only work that touches upon this matter is [6]. However, the authors in [6] assume an exhaustive candidate generation process followed by a filtering step. This may reduce the cost of the inference step but cannot scale the generation phase, especially for large KGs. The only available solution is AmpliGraph[1], an open-source KGE library which provides fact discovery strategies that leverage different sampling algorithms for the generation phase. However, it is not clear when each sampling method should be used as there has been no comprehensive study on their performance, benefits, and drawbacks.

In this paper, we present an extensive experimental evaluation of existing sampling methods for fact discovery from KGEs. From our analysis, we found that sampling methods based on entity frequency or popularity yielded better results. Similarly, we were able to discover more facts in KGs that were relatively dense. Leveraging our insights, we also shed light on challenges and open problems that remain to be solved, such as discovering facts for long-tail entities.

2 PRELIMINARIES & PROBLEM DEFINITION

Let $\mathcal{E} = \{e_1, e_2, \ldots, e_N\}$ be the set of entities and $\mathcal{R} = \{r_1, r_2, \ldots, r_K\}$ be the set of all relation types of a knowledge graph. A triple is of the form (s, r, o), where $s \in \mathcal{E}$ is the subject, $o \in \mathcal{E}$ is the object and $r \in \mathcal{R}$ is the relation between them. Let $\mathcal{T} = \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ be the set of all possible triples. A knowledge graph $\mathcal{G} \subseteq \mathcal{T}$ is a subset of all possible triples with $N = |\mathcal{E}| \geq 2$ entities, $K = |\mathcal{R}| \geq 1$ relations, and $M = |\mathcal{G}|$ triples.

2.1 Knowledge Graph Completion

Given a KG $\mathcal{G}$, the problem of *knowledge graph completion* or *missing link prediction* has been defined as finding the probability of any triple $t \in \mathcal{T}$ to belong in $\mathcal{G}$. By setting a threshold on the probability, one can determine whether a triple is true or not and label it by $\{-1, 1\}$.

KGE models learn an embedding of all entities and relations in the graph in a low-dimensional space. These models predict the existence of a triple $t = (s, r, o)$ via a scoring function $f(t; \Theta)$ which represents the model's confidence that t exists given the model parameters Θ. Θ consists of the learned latent representations of the entities s, o and relation r of t. We denote these representations as $\mathbf{s} \in \mathbb{R}^l$, $\mathbf{o} \in \mathbb{R}^l$, and $\mathbf{r} \in \mathbb{R}^l$, respectively, where $l \in \mathbb{N}$ is the embedding size of the model. KGEs aim to represent the semantics of each entity and relation using its latent representation and to use this representation to correctly predict the scores of triples. Below, we outline the scoring functions of five popular embedding models that we consider in this paper.

RESCAL. RESCAL [28] is a bilinear factorization-based model that associates each entity with a vector and each relation with a matrix to capture its latent semantics. Its scoring function is: $f_t^{RESCAL} = \mathbf{s^T r o}$.

TransE. TransE [5] is a translation-based model inspired by the Word2Vec algorithm [22]. It represents a relation as a translation operation on the entities and uses a scoring function that measures the distance of the two entities with respect to the relation of the triple: $f_t^{TransE} = -d(\mathbf{s} + \mathbf{r}, \mathbf{o})$ where $d(x, y)$ can be any distance measure, e.g., L1 or L2 norm.

DistMult. DistMult [37] can be seen as a more compact and less expressive variant of RESCAL [28]. It adds a diagonality constraint on the relation matrix and can thus, model only symmetric relations. Its scoring function is: $f_t^{DistMult} = \mathbf{s}^T diag(\mathbf{r}) \mathbf{o}$.

HolE. Leveraging the idea of associative memory, HolE [27] uses a circular correlation operation between the two entities' vectors in its scoring function: $f_t^{HolE} = \mathbf{r}_k^T (\mathbf{e}_i \star \mathbf{e}_j)$ where $(\mathbf{e}_i \star \mathbf{e}_j)_k = \sum_{t=1}^{l} e_{it} e_{j((k+t-2 \bmod l)+1)}$.

ComplEx. ComplEx [33] extends DistMult by using complex numbers and the Hermitian dot product. Its scoring function is: $f_t^{ComplE} = Real(\mathbf{s}^T diag(\mathbf{r}) \mathbf{o})$ where $Real()$ is a function that returns the real part of a complex number. ComplEx has been proven to be equivalent to HolE.

Training. Using these embeddings, we can train a machine learning model by optimizing a loss function $\min_{\Theta} \mathcal{L}(\Theta)$ dependant on the embedding method. Among the various optimization methods, the most widely-used ones include stochastic gradient descent (SGD), Adagrad [14], and Adam [16]. In this paper, all optimizations were done using Adam. Adam is one of the most popular optimization methods as it is straightforward, computationally efficient, and has low memory requirements, among many other advantages.

Testing. Typically, the performance of the embedding model is evaluated on a test dataset. For each of the triples in the test dataset, a list of corruption triples is generated: For a triple (s, r, o), we replace the o entity in the triple with every other entity in the dataset. The original triple is then ranked against its corruptions. The performance is evaluated with the help of aggregate metrics such as the mean reciprocal rank (MRR), which we further explain in Section 3.3.

2.2 Problem Definition

The goal of fact discovery is to find triples belonging to the complement of the input KG having a high probability to be true and, thus, it can be considered to be a fact. Having a high probability can be defined in multiple different ways, e.g., setting a probability threshold or getting triples that rank within the top k of a ranked list. We formally define the problem as follows:

Definition 2.1. Given an incomplete KG $\mathcal{G}$, find triples $t \in \mathcal{G}'$ belonging to the complement KG $\mathcal{G}'$ having a probability $P(t) > b$ to be true based on a KGE model built from $\mathcal{G}$, where b is a pre-defined probability threshold.

3 FACTS DISCOVERY

We detail on the fact discovery process, the different strategies we use, and the evaluation procedure we followed. We then present the metrics we used to conclude the different strategies.

3.1 Fact Discovery Process

We now provide a detailed explanation of the facts discovery algorithm and the different sampling methods used. In general, the algorithm works as follows. For each relation r existing in a given KG $\mathcal{G}$, it generates new fact candidates (s, r, o) by *sampling* entities s and o existing in $\mathcal{G}$. It then ranks the newly generated

[1] https://docs.ampligraph.org

candidates using a KGE model against its corruptions. Candidates ranking within the top n and do not already belong to $\mathcal{G}$ are then returned as facts.

Algorithm 1: Discover Facts

Input: M: a trained KGE model
G: KG data used to train the model M
top_n: max rank of candidates
$max\_candidates$: max number of facts generated per relation
$relations$: relations to discover facts for
$strategy$: sampling strategy of choice
Output: $facts$: an array of discovered facts
$ranks$: the ranks of all triples in $facts$

1 $facts \leftarrow [\,]$;
2 $ranks \leftarrow [\,]$;
3 $relations \leftarrow$ All unique relations in M;
4 $sample\_size \leftarrow \sqrt{max\_candidates} + 10$
5 **for** r *in* $relations$ **do**
6 $local\_facts \leftarrow [\,]$
7 $s\_w, o\_w \leftarrow compute\_weights(strategy)$; /* subject, object weights */
8 **while** $len(local\_facts) < max\_candidates$ || *a maximum of 5 iterations* **do**
9 Select $s\_samples$ of $sample\_size$ with s_w as sampling probability
10 Select $o\_samples$ of $sample\_size$ with o_w as sampling probability
11 Generate triples using NumPy mesh grid of $s\_samples, r, o\_samples$
12 Filter out seen entities in G out of generated triples
13 Append generated triples to $local\_facts$
14 $local\_ranks \leftarrow$ ranks of each triple in $local\_facts$ based on M
15 Filter out triples from $local\_facts$ with rank $> top\_n$ and the corresponding ranks from $local\_ranks$
16 Append $local\_facts$ to $facts$
17 Append $ranks\_iter$ to $ranks$
18 **return** $facts, ranks$

3.1.1 Discover Facts Algorithm. Algorithm 1 shows the pseudocode of the `DiscoverFacts` method. It receives as input a KGE model M, the input KG G used to build the model M, the hyperparameters top_n and $max\_candidates$, and a sampling method. Note that the algorithm is independent of the sampling method.

Algorithm 1 starts by first iterating over all relations in G (line 5). For each relation r, it assigns sampling weights to all subject and object entities according to the chosen sampling strategy (line 7). It then samples subjects and objects for r (lines 9&10). Given that the algorithm creates a mesh grid using the sampled entities (subjects or objects), the sample size is approximately equal to the square root of the $max\_candidates$ (line 4). It then generates a set of triples using a NumPy mesh grid of the sampled subject ($s\_samples$) and object ($o\_samples$) entities, and relation r. The sampling probability is determined by the chosen strategy. This generation phase is repeated until $max\_candidates$ facts are generated or 5 iterations have passed, which is a default constant. Although this could arguably be treated as another hyperparameter, we did not explore it further and took the constant value as is. Then, the algorithm filters out all the triples that model M ranks lower than top_n (i.e., $rank > top\_n$) and stores the remaining triples and their ranks in $facts$ and $ranks$. The algorithm terminates once we have iterated through every relation.

It is important to note that $max\_candidates$ limits the amount of facts generated for each relation and top_n sets the maximum rank of the generated triples per relation. Also, as the algorithm iterates over each existing relation in the KG, the runtime scales with the number of relations used in the KG.

The differentiation among different fact discovery strategies comes from the sampling method used. We discuss the existing sampling methods in the following.

3.1.2 Sampling methods. We explain the implementation of the $compute\_weights()$ in the pseudocode and how the sampling probabilities are defined by each of the strategies. The $compute\_weights()$ function first retrieves all unique entities on the subject side and all unique entities on the object side and, then, assigns weights to each one of them. These weights denote the sampling probability of the entities. We consider six different sampling strategies, namely: Uniform Random, Entity Frequency, Graph Degree, Clustering Coefficient, Clustering Triangles, and Clustering Squares.

Uniform Random Sampling. This sampling strategy assigns all the weights among the entities equally. That means every entity on each side has an equal probability of being sampled. The weights of each entity can be formulated as:

$$weight_{\text{UNIFORM\_RANDOM}}(x, side) = \frac{1}{len(side)} \tag{1}$$

where x is an entity on the $side$ side of the triple with $side \in \{subject, object\}$. Note that the weights of an entity x that exists on both subject and object sides may not be equal.

Entity Frequency Sampling. This strategy sets the weight of an entity by calculating its count relative to the total number of entities present on the same side. Consequently, entities that appear more frequently are assigned higher weights, increasing their likelihood of being selected during the sampling process. This can be formulated as:

$$weight_{\text{ENTITY\_FREQUENCY}}(x, side) = \frac{count(x, side)}{len(side)} \tag{2}$$

where $count(x, side)$ is the count of entity x as $side \in \{subject, object\}$. Similar to Uniform Random, the weights of an entity x that exists on both sides may not be equal.

Graph Degree Sampling. This strategy uses the degree of the nodes, which is normalized by the sum of all node degrees, as weights. The weights can be formulated as:

$$weight_{\text{GRAPH\_DEGREE}}(x) = \frac{deg(x)}{\sum_{v \in V} deg(v)} \tag{3}$$

where $deg(x)$ denotes the sum of in- and out-degree of node x and V is the set of nodes in the graph. This strategy does not differentiate between the entity sides, i.e., the sampling probability of an entity x that appears both as a subject and object is equal. This method prioritizes popular nodes in the graph, i.e., nodes with many connections to other nodes regardless of the direction of connections.

Clustering Triangles Sampling. This sampling method leverages the number of triangles that an entity participates in.

It is defined as:

$$weight_{\text{TRIANGLES}}(x) = \frac{T(x)}{\sum_{v \in V} T(v)}. \quad (4)$$

where $T(v)$ is the number of triangles that includes v as a node (local triangles count). $T(v)$ is computed as:

$$T(v) = |\{e_{uw} : u, w \in N_v,\ e_{uw} \in E\}|$$

where u and w are neighbour nodes of v and e_{uw} represents an edge between u and w and E is the set of all edges in the graph. Note that triangles are computed as if the graph is homogeneous and undirected. The more triangles an entity is part of the more probability it has to be sampled. Local triangle counting forms another measurement of node popularity.

Clustering Coefficient Sampling. This strategy assigns weights based on the local clustering coefficient of a node. It is defined as:

$$weight_{\text{CLUSTER\_COEFFICIENT}}(x) = \frac{c(x)}{\sum_{v \in V} c(v)} \quad (5)$$

where $c(\cdot)$ is the local clustering coefficient defined as:

$$c(v) = \frac{2T(v)}{deg(v)(deg(v) - 1)}$$

where $T(v)$ is the local triangles count of node v and $deg(v)$ is the degree of v [36]. The clustering coefficient is a measure of the density of nodes surrounding x. The higher the coefficient of a node, the larger the probability that this node will be sampled. Again, this sampling method assumes an undirected homogeneous graph.

Clustering Squares Sampling. This method assigns weights using the squares clustering coefficient $c_4(\dot{})$. The weights are defined as:

$$weight_{\text{CLUSTER\_SQUARES}}(x) = \frac{c_4(x)}{\sum_{v \in V} c_4(v)} \quad (6)$$

Unlike the clustering coefficient, the squares clustering coefficient calculates the fraction of possible squares (a cycle with 4 nodes) that goes through a node. The square clustering coefficient is defined as [39]:

$$c_4(v) = \frac{\sum_{u=1}^{k_v} \sum_{w=u+1}^{k_v} q_v(u, w)}{\sum_{u=1}^{k_v} \sum_{w=u+1}^{k_v} [a_v(u, w) + q_v(u, w)]}$$

where k_v is the number of nodes adjacent to v, $q_v(u, w)$ is the number of common neighbours of u and w excluding v, and

$$a_v(u, w) = (k_u - (1 + q_v(u, w) + \theta_{uv})) + (k_w - (1 + q_v(u, w) + \theta_{uw}))$$

is the number of non-existing squares around v, where $\theta_{uw} = 1$ if u and w are connected, and 0 otherwise. Intuitively, the squares clustering coefficient provides valuable insight into the likelihood that two neighbors of a given node are connected through a mutual neighbor, distinct from the given node itself. This metric helps us understand the level of interconnectedness within the local network structure and the potential for indirect relationships between neighboring nodes.

3.2 Evaluation Process

Figure 1 illustrates the process we followed for our experimental analysis. It consists of the following steps:

Dataset Selection. There is a wide variety of datasets for KGEs, each with different amounts of entities, relations, and triples, as well as distinct graph properties. During the dataset selection, we consider these factors and strive to choose datasets with varying sizes to test the scalability of the fact discovery algorithms. We kept away from datasets with a common source to make the results as general as possible and avoid bias towards a particular dataset. Aside from the technical factors, we took into account the popularity of a dataset, i.e., how often a certain dataset is used by the research community as a benchmark. Choosing a popular dataset would be more convenient as the community is more accustomed to it, and comparisons to previous and future works would also be more easier.

KGE Algorithm Selection. There is a multitude of KGE algorithms each with their perks, some of which we have mentioned in Section 2. We take into account their strengths, weaknesses, and type of embedding method (e.g., translation-based, geometric-based, etc.) when choosing the methods for the experiments. We strive to deliver a conclusion on the strategies pertaining to a broad selection of embedding methods. Similar to the dataset selection, we also considered the popularity of the embedding methods within the research community.

Model Training. After the dataset and algorithm selection, we conduct hyperparameter tuning on all possible combinations of datasets and embedding algorithms to obtain the optimal embedding models. Let m and n be the number of datasets and embedding algorithms, respectively. We tune the parameters for all $m \times n$ embedding models. As evaluating embedding models is not the focus of this paper, we do not pay particular attention to the methodology of obtaining the optimal embedding models. In this regard, we are open to hyperparameters used by prior research as well as doing our own tuning, for instance through grid search. We then use the trained embedding models with optimal hyperparameters for the fact discovery algorithms.

Discover Facts. We use the fact discovery methods provided by Ampligraph's Discover Facts functionality[2]. This requires an embedding model, the KG used to train the model, the relations for which we want to discover facts, a sampling strategy of choice, and the hyperparameters *top_n* and *max_candidates*. It then delivers an array of discovered facts and their ranks evaluated by the standard evaluation protocol as per [5].

3.3 Evaluation Metrics

To evaluate the different sampling methods for discovering new facts we take the following steps. First, each method outputs a set of discovered facts together with their ranks against their corrupted triples. These ranks show the plausibility of the facts and can thus be used for comparing the quality of the different sampling method strategies. We thus leverage them to compute the mean reciprocal rank (MRR) of each produced set of facts. MRR is also one of the most important metrics used to evaluate the performance of an embedding model. The MRR formula is as follows:

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i} \quad (7)$$

where Q is a set of triples and $rank_i$ is the rank of the i_{th} triple against its corrupted triples. MRR is usually more favored as compared to the simple mean rank as it is more robust to outliers.

Apart from measuring the quality of the generated facts, we are also interested in the efficiency, i.e., how fast a strategy can output a certain amount of such facts. We evaluate the efficiency

[2]https://docs.ampligraph.org/en/2.0.0/generated/ampligraph.discovery.discover_facts.html

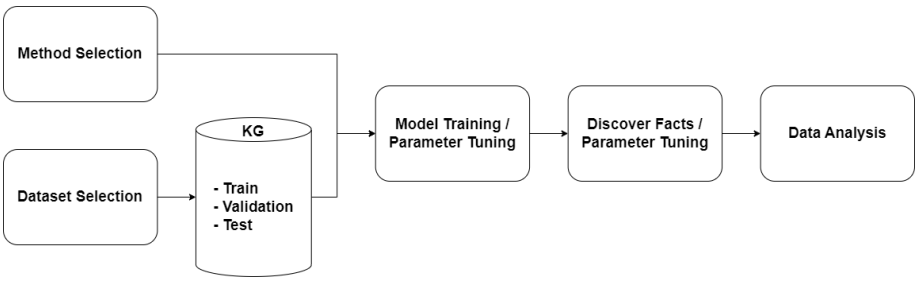


Figure 1: Experimental workflow.

of a strategy by dividing the amount of generated facts by the total runtime of the algorithm, which is made up of fact discovery (i.e., the sampling process) and fact evaluation (i.e., the filtering of triples with a rank lower than n in top_n). In contrast to the runtime which simply intuits how the algorithm of the strategies behaves, efficiency represents the throughput of the algorithm.

4 EXPERIMENTAL STUDY

We examined a variety of fact discovery methods and KGEs within an adaptable and robust framework. Central to this framework's design is its inherent flexibility, which allows for the seamless integration of diverse datasets and an array of scorer models, notably for the computation of metrics such as Mean Reciprocal Rank (MRR). The framework's capability to pair each dataset with a corresponding fact discovery strategy is crucial, as it significantly influences the sampling probability of each node in the phase of triple construction. The code underpinning these experiments is publicly accessible on GitHub[3], and while it is currently optimized to augment the LibKGE library [9] and its associated models, the architecture of our framework is not constrained to this setup. We deliberately design it to allow straightforward modifications in the source code, thereby enabling the integration of alternative scoring models beyond those provided by LibKGE. This level of customization and adaptability offers researchers and practitioners the flexibility to explore new datasets, sampling strategies, and scoring models.

We seek to answer the following questions: (i) What is the fact discovery throughput and runtime of each method? (ii) How does the quality of the discovered facts differ for different algorithms? (iii) How do different KGE models interact with the different sampling methods? Does the use of different KGE models affect the quality and throughput of each method? (iv) How do the algorithms are affected by the different KG datasets?

Before describing the experimental setup and results of our study in detail, we briefly summarize our key findings:

- Sampling methods based on node frequency or popularity yielded positive results.
- All sampling methods tend to yield better results with dense datasets, i.e., measured by the global clustering coefficient.
- Entity Frequency and Graph Degree performed well in runtime and excelled in discovering high-ranked triples.
- Uniform Random and Clustering Coefficient performed poorly in the quality of discovered facts.
- Clustering Triangles was the top performer in terms of throughput, consistently yielding more facts than other methods.
- Clustering Squares was very inefficient in runtime so it could not be compared with the other methods for the datasets we considered.

4.1 Experimental Setup

In this section, we provide a detailed overview of the foundational components underpinning our experimental design. Such transparency is essential for ensuring the reproducibility and robustness of our results. We initiate our discussion by outlining the specific hardware and software configurations employed, offering insights into the computational environment that facilitated our research. Subsequently, we turn our attention to the datasets we leveraged, elaborating on their origins and their relevance to our study's objectives.

4.1.1 Hardware and Software. Our experiments were carried out on a robust hardware configuration, comprising an Intel(R) Xeon(R) Gold 5115 CPU with 40 cores, an NVIDIA Tesla V100-PCIE-16GB GPU, 188GB of RAM, and a 140GB SSD for storage, all running on the Ubuntu 22.04 LTS operating system. We used the open-source library LibKGE [9] to train our models. We chose LibKGE for several reasons. First and foremost, LibKGE is incredibly modular, allowing us to easily incorporate new elements (e.g., datasets) into the library. This is particularly important for our study as we worked with a variety of data sources and needed a library that could adapt to our needs. Furthermore, LibKGE provides a yaml job-like definition of the experiments, which allows for a simple and high degree of customization. This feature made it easy to define and run experiments, helping to streamline the research process. Additionally, the library provides a grid search syntax suitable for running multiple experiments sequentially and testing parameters, which was important in optimizing our models. Lastly, the library is actively maintained and updated, which ensures that any bugs or issues encountered will likely be addressed promptly. We adopted the fact discovery strategies implemented in the Discovery API of Ampligraph[4] and re-implemented them to suit LibKGE.

4.1.2 Datasets. For our evaluation, we used four diverse datasets commonly used for the evaluation of knowledge graph embeddings: FB15K-237 [32], WN18RR [12], YAGO3-10 [20], and CoDEx-L [30]. An overview of the metadata of the datasets can be found in Table 1. For each dataset, the train, test, and validation splits are available.

[3] https://github.com/ramesesz/fact_discovery

[4] https://docs.ampligraph.org/en/1.4.0/ampligraph.discovery.html

Table 1: Metadata of the datasets.

Dataset	Training	Validation	Test	Entities	Relations
FB15K-237	272,115	17,535	20,429	14,541	237
WN18RR	86,835	3,034	3,134	40,943	11
YAGO3-10	1,079,040	5,000	5,000	123,182	37
CoDEx-L	550,800	30,600	30,600	77,951	69

FB15K-237. The FB15K dataset was first created by Bordes et al. [5] as a smaller-sized experimental dataset based on Freebase [4], which is a vast collaborative knowledge base capturing diverse real-world entities and relationships, spanning topics from biographical data of individuals to detailed information about media, places, organizations, and more. Later on, Toutanova and Chen [32] pointed out that the dataset contains test leakage through inverse relations. This means that a large number of test triples can be obtained by inverting triples in the training set. For instance, the test set contains (t, r^{-1}, h), whereas the train set contains (h, r, t), where r^{-1} denotes the inverse relation of r. Such occurrences are detrimental when evaluating KGE models, as they are very easily deducted. To get rid of this property, FB15K-237 [32] was created by removing inverse triples.

WN18RR. WordNet[5] [23], is a lexical database containing semantic relations between words in over 200 languages. Nouns, verbs, adjectives, and adverbs are grouped into synsets, a group of words expressing a unique concept. Phrases that express the same or similar concepts are grouped into the same synset. Synsets are connected to other synsets through edges that illustrate their relationship, including but not limited to hyponyms, hypernyms, troponyms, meronyms, and entailments. WN18 was first introduced by Bordes et al. [5] by scraping 18 relations from WordNet. Similar to FB15K, WN18 was later found to contain an inverse relation test leakage. Dettmers et al. [12] then created WN18RR, a subset of WN18, to fix this issue.

YAGO3-10. YAGO3-10[20] is a subset of YAGO3[20] (which itself is an extension of YAGO[31]), a dataset composed of the WordNet dataset and data extracted from Wikipedia[6]. The 10 in YAGO3-10 refers to the fact that this dataset only contains entities of YAGO3 that have a minimum of 10 different relations, resulting in a denser dataset. YAGO3-10 contains roughly 1.1 million triples, describing attributes of people, organizations, places, and other general information.

CoDEx-L. CoDEx [30] is a KG extracted from Wikipedia and Wikidata [35] created as an improvement to KG completion benchmarks both in scope and level of difficulty. The dataset includes multilingual descriptions of entities and relations, as well as tens of thousands of hard negative triples, which are plausible but proven to be false. CoDEx comprises three KGs differing in size and structure, namely CoDEx-S, CoDEx-M, and CoDEx-L with 36k, 206k, and 612k triples, respectively. All CoDEx versions have a 90:5:5 ratio for train, validation, and test triples, created such that there are no unseen triples on the validation and test set, and inverse relations have been removed to prevent test leakage. We used CoDEx-L for our experiments.

[5]https://wordnet.princeton.edu/
[6]https://www.wikipedia.org/

4.2 Sampling Methods Comparison

We analyzed the results of our experiments and compared the performance of the different strategies. In this aspect, we observed the strategies from three different dimensions: runtime, fact quality, and fact discovery efficiency (i.e., the amount of discovered facts per time unit). Due to Clustering Coefficient being extremely inefficient in our preliminary experiments, we exclude the strategy in our comparative experiments. We detail the reason in Section 4.3.

4.2.1 Runtime. We define runtime as the total time it takes for an execution of the discovery algorithm to terminate. As shown in Figure 2, Uniform Random, Entity Frequency, and Graph Degree had similar runtimes, as do Clustering Coefficient and Clustering Triangles. This is particularly evident in the datasets FB15K-237, YAGO3-10, and CoDEx-L (Figure 2a, c, and d, where Clustering Coefficient and Clustering Triangles took significantly longer to terminate than the other strategies.

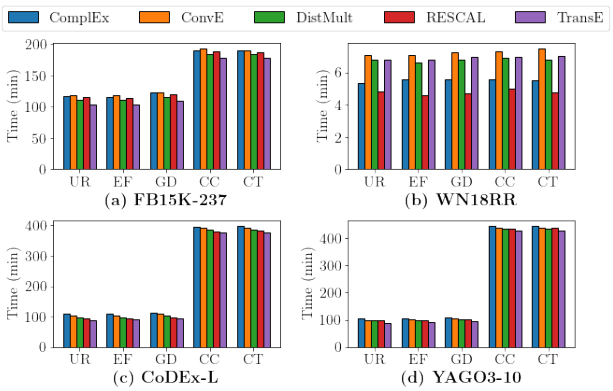


Figure 2: Runtime of the discovery algorithm. The experiments are grouped based on the strategies on the x-axis, where Uniform Random, Entity Frequency, Graph Degree, Clustering Coefficient, and Clustering Triangles are abbreviated, respectively from left to right.

The distinction between the two groups stemmed from their complexity. Both the Clustering Coefficient and Clustering Triangles calculate the number of triangles around each node using $T(v)$, which has a complexity of $O(n^3)$. This increased the runtime of both strategies by a significant amount. In contrast, Uniform Random assigns equal probability to all entities, and Entity Frequency and Graph Degree retrieve their sampling probabilities from graph properties which can be calculated in linear time.

It is, however, noticeable that the distinction between the two groups was somewhat blurred when dealing with the WN18RR dataset in Figure 2(b). Furthermore, every single experiment on WN18RR exhibited abnormally short runtimes, terminating in under 10 minutes.

There are three main factors contributing to this phenomenon, the first being the high sparsity of the dataset. Figure 3 shows that WN18RR nodes have considerably lower clustering coefficients (i.e., the average of the clustering coefficient of the nodes) than in other datasets, indicating a higher sparsity. The dataset also only has around 90k triples despite having over 40k entities. As each triple has two entities, it can be inferred that entities of WN18RR have an average of 4.5 relations. This is a significantly lower count than the other three datasets.

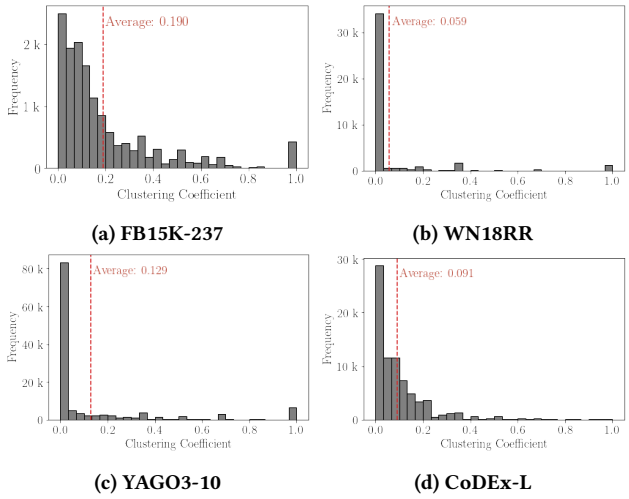


Figure 3: The distribution of the clustering coefficients of all nodes across the datasets. The red line displays the average clustering coefficient value of all nodes within the dataset.

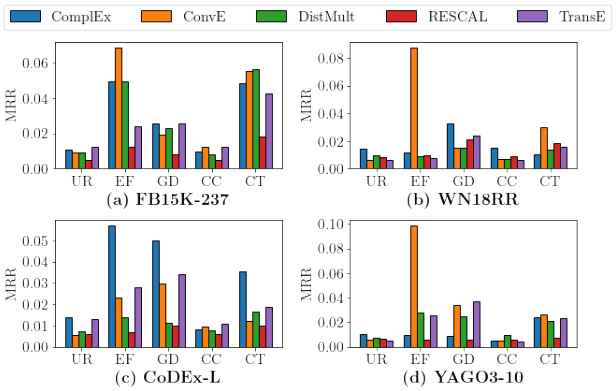


Figure 4: MRR of the discovery algorithm. The experiments are grouped based on the strategies on the x-axis, where Uniform Random, Entity Frequency, Graph Degree, Clustering Coefficient, and Clustering Triangles are abbreviated, respectively from left to right.

Secondly, triangles-based algorithms such as the Clustering Coefficient and Clustering Triangles in conjunction with the sparsity of WN18RR demonstrate the effect of fixed-parameter tractability [13], leading to a substantial reduction in runtime duration. The third reason for which the WN18RR dataset showed very short runtimes is the fact that the discovery algorithm iterates over all relations. WN18RR consists of only 11 different relations. While this leads to fewer triples being discovered, the runtime in turn becomes very short.

Contrary to the strategies employed, the choice of the KGE model has a negligible influence on the runtime of the discovery algorithm. The margin in runtime across different models is minimal, suggesting that the models have little to no substantial impact on runtime.

4.2.2 Quality. This section describes the quality of the experiments concerning the *top_n* parameter, which sets the quality threshold for the discovered facts. To reiterate, the *top_n* parameter filters out triples that ranked lower than n against its corruptions. In the following experiments, we set $top\_n = 500$. This sets a theoretical MRR threshold of 0.002 in the case where all discovered facts are exactly ranked 500.

Our baseline strategy, Uniform Random, performed relatively poorly, being one of the bottom two strategies alongside Clustering Coefficient. Uniform Random assigns equal sampling probability to all nodes. This means that a 'bad' node has an equal chance of being chosen as a 'good' node. A good node refers to one that appears frequently in a KG, and a bad node the opposite. When building triples, choosing a frequent node tends to result in a high-scoring triple. In the case of KGs, bad entities vastly outnumber the good ones, explaining why Uniform Random delivered subpar triple quality.

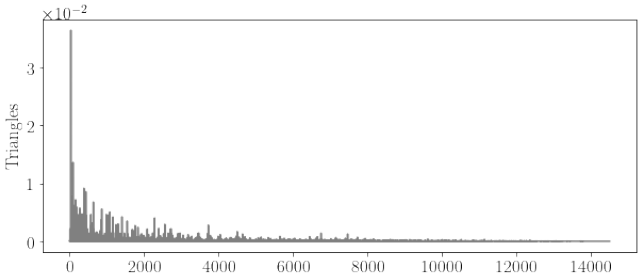
Entity Frequency outperformed Uniform Random for almost every single model. This strategy assigns a higher sampling probability to nodes that frequently appear, which more often than not correlate to good nodes. Entity Frequency performed outstandingly in FB15K-237 and showed especially high affinity with ConvE, which is most apparent in Figure 6a, b, and d, outperforming the other models within the same strategy group in terms of quality.

We discovered that Entity Frequency performed exceptionally, however abnormally well when paired with ConvE. While this might simply mean that the ranking distribution of the facts leaned more towards high ranks, we hypothesized popularity bias to play a role in exaggerating the results. Popularity bias refers to a phenomenon where the score of triples containing popular entities and relations is amplified way more than necessary. While these deliver better results, popularity bias is undesirable as it indicates that the model fails to capture the real-world semantics within the KG.

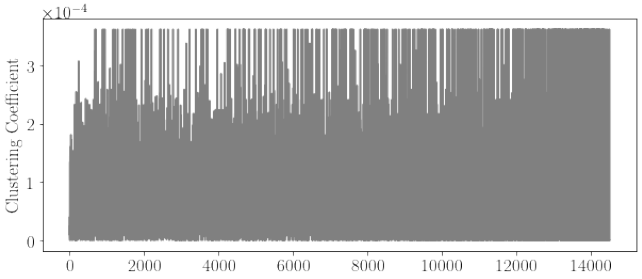
In terms of quality, Entity Frequency outperformed its algorithmically similar counterpart, Graph Degree, in Figure 4a, while performing similarly well in the other datasets. This signifies that keeping the sampling weights of the head and tail sides separate tends to result positively in small and medium-sized datasets similar to FB15K-237. Despite that, Graph Degree showed more stable results concerning the disparity within the same strategy group, indicating a more evenly spread distribution of the fact quality.

Clustering Triangles uses a more complex popularity metric as compared to the 'naive' Entity Frequency and Graph Degree. Clustering Triangles especially excelled in the dense FB15K-237 (Figure 4(a)), while also showing consistent above-average performance in the other datasets. Following the same line of thought in the previous paragraph, Clustering Triangles showed itself to be more stable than Graph Degree. This also suggested that the more complex popularity metric employed by Clustering Triangles is more robust towards popularity bias than the more naive Entity Frequency and Graph Degree. It is also noteworthy that Clustering Triangles outperformed its triangle algorithm-based counterpart, Clustering Coefficient, by a wide margin.

In the case of Clustering Triangles, the triangle value of a node, and by extension its sampling probability, correlates to the popularity or frequency of said node. In contrast, the Clustering Coefficient strategy could potentially penalize popular nodes. For example, in a star graph embedded within a KG, the central node is highly popular (measured by node degree) but has a clustering coefficient of 0. Due to the nature



(a) Triangle values



(b) Clustering coefficients.

Figure 5: The distribution of the clustering coefficients of all nodes across the datasets. The x-axis shows the index of the nodes within the dataset.

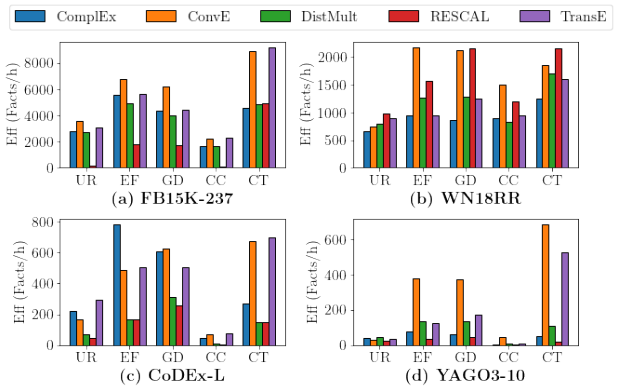


Figure 6: Efficiency of the discovery algorithm. The experiments are grouped based on the strategies on the x-axis, where Uniform Random, Entity Frequency, Graph Degree, Clustering Coefficient, and Clustering Triangles are abbreviated, respectively from left to right.

of the Clustering Coefficient strategy, there is little to no correlation between the popular nodes and sampling probability. This resulted in the quality of the triples averaging in the lower end of the spectrum, occasionally even lower than Uniform Random. Figure 5 shows the clustering coefficient and triangle values of all nodes in FB15K-237. By observing the same index of the x-axis, we can compare the triangle value of a node in Figure 5a with its clustering coefficient in Figure 5b. At large, the clustering coefficient of a node tends to fluctuate regardless of its triangle value. This lack of correlation between these two values reinforces our prior argument.

4.2.3 Efficiency. This subsection elaborates on the discovery output of the experiments we conducted, termed *efficiency*. Note that the efficiency of an experiment is not unrelated to its fact quality, as the fact discovery algorithm only allows fact candidates ranking among the top n (in our experiments $n = 500$), meaning a certain level of quality is required for a fact candidate to be outputted as a fact.

Reflecting the results on the fact quality, Uniform Random and Clustering Coefficient were the two bottom performers concerning discovery efficiency.

In a similar trend to the Section 4.2.2, Entity Frequency outperformed the baseline Uniform Random. However, contrary to Figure 4, we did not observe the stark difference displayed by the Entity Frequency and ConvE pair in Figure 6. This suggested that Entity Frequency and ConvE resulted mostly in facts in the higher end of the spectrum with regards to *top_n*, however in lower quantities. As opposed to this observation, Entity Frequency and Graph Degree with ComplEx showed similarly good performance both in Figure 4(c) and Figure 6(c). This indicates that the Graph Degree-ComplEx combination yielded high fact quality while also maintaining similarly high throughput.

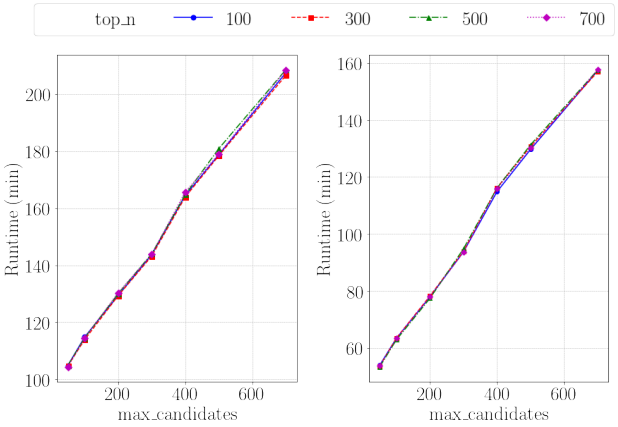
Overall, Graph Degree exhibited similar efficiency to Entity Frequency, having a slightly better output in Figure 6(b) and (c), while the latter has a slight advantage over the former in Figure 6(a). Amongst all configurations, Clustering Triangles delivered the most facts per hour on average. Despite performing rather mediocrely in Figure 4, Clustering Triangles showed strong efficiency, especially in Figure 6(a) and (b).

Size and sparsity were two traits that heavily affect efficiency. We discovered that the density of a KG needs to scale together with its size in order to deliver good efficiency. The small WN18RR had a clustering coefficient average of 0.059 (Figure 3b) but showed exceptional efficiency. On the other hand, YAGO3-10 showed much lower efficiency despite having a higher density (Figure 3(d)). Although its entities have a minimum of 10 different relations, YAGO3-10 (Figure 6 (d)) showed the lowest efficiency amongst the datasets due to its massive scale. The best-performing model of YAGO3-10 delivered only 600 facts/hour, a low number compared to the average efficiency of the other datasets. In the contrary, the small WN18RR had a relatively higher efficiency despite having fewer relations per node.

Sparsity is a classical problem of KGs which tends to get worse the larger the graph. The more entities a KG contains, the more likely the chance that we sample long-tail entities that are infrequent and score poorly.

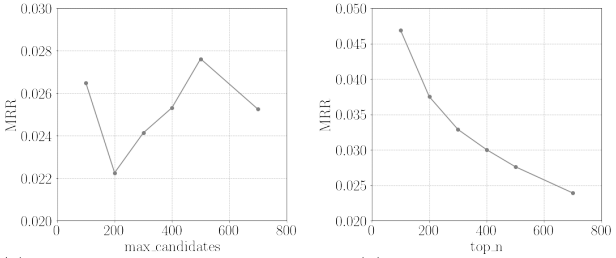
4.2.4 Summary of Findings. Based on our experiments using the Discovery algorithm, we have uncovered the following. The runtime of fact discovery highly depends on the number of relations a dataset contains and its sparsity. Fewer relations and a sparser dataset yielded a shorter runtime. As for the strategies, Clustering Coefficient and Clustering Triangles took longer to terminate due to their complexity.

Clustering Triangles and Entity Frequency are top performers with respect to fact quality. However when opting for consistency across different models, Graph Degree and Clustering Triangles are the most suitable options. We discovered that it is imperative that a strategy assigns a probability to a node that correlates with how frequently it appears in the dataset. Metrics or strategies that exhibit a strong positive correlation with the frequency or popularity of a node tend to produce favorable



(a) Clustering Triangles (b) Uniform Random

Figure 7: The runtime of the fact discovery algorithm on FB15K-237 with TransE. The different lines represent different *top_n* values.



(a) Impact of *max_candidates* on MRR (b) Impact of *top_n* on MRR

Figure 8: The quality of the fact discovery algorithm on FB15K-237 with TransE on the Clustering Triangles strategy. (a) Development of MRR with varying *max_candidates* values and *top_n* = 500. (b) Development of MRR with varying *top_n* values and *max_candidates* = 500.

outcomes, benefiting both the quality of facts and the efficiency of discovery. Notably, this correlation holds true for metrics such as Entity Frequency, Graph Degree, and Clustering Triangles. However, it is less evident in the case of Uniform Random and Clustering Coefficient, as their sampling probabilities do not align well with the popularity of nodes in the network.

4.3 Hyperparameter Analysis and Tuning

In this section, we elaborate on the tuning process of the parameters *top_n* and *max_candidates*, demonstrate their effect, and argue on the value we used for our experimentation in Section 4.2. Recall that *max_candidates* determines the maximum number of fact candidates that the discovery algorithm can generate and *top_n* sets the quality threshold, i.e., a fact candidate needs to rank higher than *n* against its corruption to be considered as a fact and thus be outputted by the algorithm.

Each execution of Discover Facts is very time-consuming. Preliminary experimentation of all strategies on FB15K-237 with TransE took 2-3 hours on average, excluding the strategy Clustering Squares. Clustering Squares took close to 54 hours while only discovering a meager 5268 facts, translating to 98 facts per hour. Due to being severely inefficient, we decided to exclude this strategy from our experiments.

4.3.1 Hyperparameter Analysis. The scope of experiments in this paper covers the combination of four datasets, five embeddings, and five strategies, resulting in a total of 100 experimental configurations. As analyzing the entirety of the configurations individually would be exceedingly time-consuming, we conducted hyperparameter analysis on two configurations: using the dataset FB15K-237 on TransE using the baseline Uniform Random and Clustering Triangles, which showed a promising amount of triple output in our preliminary experiments. To uncover how the *max_candidates* and *top_n* parameters affect the discovery algorithm, we conducted a grid search on *max_candidates*, exploring the values {50, 100, 200, 300, 400, 500, 700} and examined the values {100, 200, 300, 400, 500, 700} for the parameter *top_n*. We mainly observed the following metrics during the grid search: The runtime of the algorithm, the number of discovered facts, the quality of the discovered facts, and the discovery efficiency.

Based on the runtime data collected, *top_n* had practically no visible impact on the runtime of the algorithm, as visualized by the overlapping line graphs in Figure 7. On the other hand, *max_candidates* showed a linear increase in runtime as the value increases. As can be seen from its role in the algorithm (Algorithm 1), *top_n* simply acts as a filter on a list of triples. Increasing its value did not change the runtime in any way, as it is required to iterate through all triples to apply the filter regardless of its value. In contrast, *max_candidates* defines the amount of triple candidates being generated by the algorithm, increasing the amount of triples that have to be evaluated. A higher *max_candidates* value increased the evaluation time, and by extension also the runtime of the discovery algorithm. With respect to quality, Figure 8 revealed that increasing the *top_n* value reduced the MRR of the discovered facts, whereas *max_candidates* showed a stable MRR within a certain range as its value increased.

To summarize, increasing *max_candidates* resulted in more facts being discovered without compromising the quality of the facts, however at the cost of a longer runtime. While increasing *top_n* yielded more facts without increasing runtime, it is important to note that the general quality of the facts deteriorated as the parameter value increased.

4.3.2 Hyperparameter Tuning. The experiments conducted for the hyperparameter analysis also served to be a basis for hyperparameter tuning. We analyzed the data, mainly with respect to efficiency, to choose the most ideal values of *top_n* and *max_candidates* to be used in the experimental evaluation of the sampling strategies.

We first attempted to fix the *top_n* value. As shown in Figure 9a, the development of efficiency with Clustering Triangles starts to plateau after *top_n* reaches 200, whereas the efficiency increases more steeply after the same value in Figure 9b. While it might be in our favor to take this elbow point as the fixed parameter value, setting *top_n* = 200 generated too few triples in our subsequent experiments, which could potentially lead to higher variance within the results. We settled with *top_n* = 500 to ensure that the amount of facts discovered can capture the essence of the experimental configuration.

We then set the *top_n* value as a pivot and observed how *max_candidates* affected efficiency. The results are shown in Figure 10. The efficiency increase starts to level at 500 for Clustering Triangles and is rather unpredictable for Uniform Random. At this junction, we leaned towards Figure 10a, as we considered

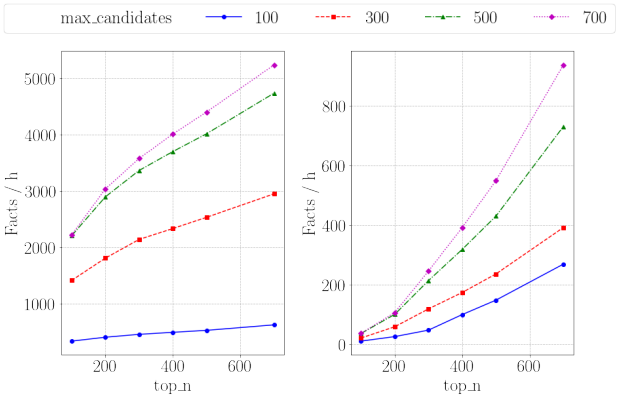


Figure 9: The impact of *top_n* to discovery efficiency. The different lines represent varying *max_candidates* values.

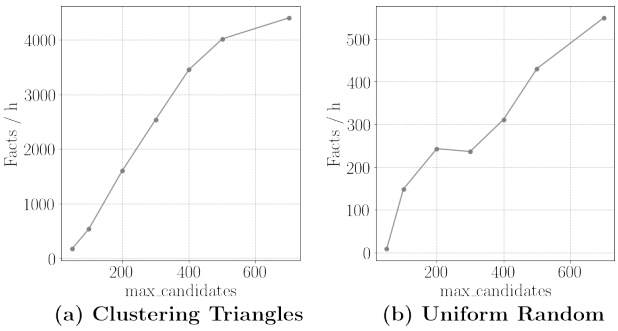


Figure 10: The impact of *max_candidates* to discovery efficiency, *top_n* = 500.

the fundamental randomness of Uniform Random to be less reliable when deciding the parameter values. Therefore, we decided on 500 for the *max_candidates* value.

5 RELATED WORK

In recent years, research on KGs is increasingly gaining support due to their versatility in solving real-world problems in various industries. Although there is vast literature on KGEs, there is a pitifully low amount of research regarding fact discovery, and this subtask of KG completion remains relatively unexplored. We review the most important works on both areas in the following.

5.1 Fact Discovery

The only work that focused on fact discovery is CHAI [6], which defines KG completion as a workflow consisting of three subtasks: Candidate generation, candidate filtering, and fact-checking, with them connected in a linear, iterative workflow. The workflow begins with candidate generation, what we call fact discovery in this paper. This step is an exhaustive one, i.e., the entire missing set of edges in a KG (the complement of the KG) has to be generated.

Candidate generation is followed by candidate filtering, which is the focus of Borrego et al. in [6]. Borrego et al. proposed a rule-based filtering tool to remove unreasonable triples (e.g., a person being a child of an object). After the filtering process, the remaining triples are passed to a fact-checking model. The authors suggest Ontological Graph Fact-checking (OGCF) [19], a rule-based model which distinguishes the truthfulness of a triple. Another notable alternative is SciCheck [8], a neural-based classification model that accomplishes the same task as OGCF.

When observed as individual KG completion tasks, CHAI takes arguably considerably less time than most fact-checking models as no trained models are required. However, CHAI lacks the capability to capture the ontology of a KG, which can be addressed through the intervention of KGE techniques. Also, CHAI as presented in [7] assumes the entire missing set as input, which may pose challenges when dealing with large KGs.

The fact discovery method discussed in this paper combines fact-checking with intelligent candidate generation. In contrast to the exhaustive method, we evaluated sampling strategies to generate candidates in a more efficient way. Furthermore, by using a trained KGE model to filter the discovered facts, we ensure that the remaining facts represent the ontology of the KG. In conclusion, the rule-based filtering method CHAI [7] would potentially be a good complement to the discussed fact discovery.

5.2 Link Prediction

The field of link prediction in knowledge graphs has seen a considerable shift towards diverse application areas, highlighting the adaptability and significance of this research in various domains. Recent studies reflect a broad spectrum of applications, ranging from social networks to predicting interactions between proteins [11, 18, 21].

A prominent area of application is in social networks, as explored in several studies [11, 18]. This research highlights the complexity of social structures and interactions that link prediction models must navigate. In these networks, predicting links goes beyond mere data analysis; it requires an understanding of social dynamics, user behavior, and community structures. The ability to accurately forecast social connections has profound implications for areas such as social media analytics, recommendation systems, and targeted advertising.

Another critical application is in scientific research domains, such as bioinformatics, where predicting interactions between proteins or genes can lead to significant breakthroughs [21]. In such contexts, the accuracy and efficiency of link prediction models are paramount, as they directly influence the potential for scientific discovery and innovation. Furthermore, link prediction techniques are increasingly being employed in more diverse and complex scenarios, including e-commerce, cybersecurity, and network analysis [18].

These varied application domains underscore the evolving nature of link prediction research. While initial studies focused on developing foundational models [5, 33, 38], current research is increasingly application-driven, seeking to address the specific needs and challenges of different domains. This shift towards application-specific research underlines the growing importance and impact of link prediction in a wide array of fields, extending far beyond its traditional boundaries. Many works experimentally evaluate different KGE models [1, 2, 17, 29] in various settings.

6 LESSONS LEARNED AND FUTURE DIRECTIONS

During the implementation, evaluation, and analysis of the several sampling methods for fact discovery, we have made several observations that we believe are important for advancing this research area. We discuss them in the following:

- *Fact discovery focuses on dense areas of KGs.* The metrics used in the sampling methods we analyzed, such as entity frequency, clustering coefficient, and graph degree, measure somehow the density of the graph. Thus, all but the Uniform Random sampling methods extract facts from the densely-populated areas of a KG, i.e., entities that are highly popular with many connections to other entities. Oddly, this leads to leaving out long-tail entities where the need for discovering new facts is higher. This is an issue that has also been discussed for KGEs themselves [24], where popularity-aware metrics have been proposed for the evaluation of KGEs.
- *KGE models are assumed to be accurate.* In our current setting, the discovery algorithm filters out all triples that are ranked low by the corresponding KGE model (see line 15 in Algorithm 1). Although the threshold, *top_n*, used for filtering out is user-specified, it still implicitly assumes that the KGE model is correct in accurate in determining whether a triple is true or false and thus, ranking the triples. This, however, is far from reality. Typically current KGE models contain a large error: if we observe their MRR or Hits@k values that are a bit over 50% [17].
- *No evaluation protocols.* Currently, there are no protocols for how the evaluation of fact discovery methods can be done. Following the standard training/validation/testing split protocol used for KGE models does not work for two main reasons. First, the fact discovery process is not exhaustive as we cannot extract all possible plausible facts which is computationally expensive. Second, the fact that a triple does not exist in the test dataset does not necessarily mean that it cannot be true.

Based on our above observations there are three future directions that the research community should focus on. One is the development of new fact discovery methods and sampling strategies that explore the sparse areas of KGs. This resembles the exploration vs. exploitation dilemma always encountered in recommendation systems. In the current KGE and sampling methods, the exploration part is undermined. Second, there is a need for devising different pruning mechanisms for faster iterating through the exhaustive list of generated candidates. A first attempt towards this direction has been done in [6] where the authors devise rules for pruning 'illogical' triples. Last but not least, new evaluation protocols and metrics need to be devised for the fact discovery problem. Ideas can be drawn from the deductive reasoning world which focuses on inferring new triples based on a set of entailment rules.

7 CONCLUSION

In this paper, we evaluated sampling strategies for the problem of fact discovery from KGEs, i.e., extracting missing facts from an incomplete KG based on a given KGE model. Our experiments covered the datasets FB15K-237, WN18RR, YAGO3-10, and CoDEx-L, as well as the KG embeddings ComplEx, ConvE, DistMult, RESCAL, and TransE. We implemented the fact discovery algorithm using the Discover Facts API of AmpliGraph and investigated the performance of the algorithm on different configurations, hyperparameter values, and strategies. Our experiments showed that sampling strategies that assign probabilities correlating to a node's frequency or popularity yielded more positive results. Uniform Random and Clustering Coefficient underperformed and thus are less suitable choices to discover facts. Entity Frequency and Graph Degree are strong options, performing well across the board and excelling in discovering high-ranked triples. While slightly worse with fact quality, Clustering Triangles was the top performer with respect to discovery output, consistently yielding more facts than the other strategies. There are still many open problems for the fact discovery problem. We believe this paper to encourage more researcher to further explore this direction.

ACKNOWLEDGMENTS

We gratefully acknowledge funding from the German Federal Ministry of Education and Research under the grant BIFOLD23B.

REFERENCES

[1] Farahnaz Akrami, Mohammed Samiul Saeef, Qingheng Zhang, Wei Hu, and Chengkai Li. [n.d.]. Realistic Re-evaluation of Knowledge Graph Completion Methods: An Experimental Study. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). 1995–2010.

[2] Mehdi Ali, Max Berrendorf, Charles Tapley Hoyt, Laurent Vermue, Mikhail Galkin, Sahand Sharifzadeh, Asja Fischer, Volker Tresp, and Jens Lehmann. 2022. Bringing Light Into the Dark: A Large-Scale Evaluation of Knowledge Graph Embedding Models Under a Unified Framework. *IEEE Trans. Pattern Anal. Mach. Intell.* 44, 12 (2022), 8825–8845.

[3] Mehdi Ali, Charles Tapley Hoyt, Daniel Domingo-Fernández, Jens Lehmann, and Hajira Jabeen. 2019. BioKEEN: a library for learning and evaluating biological knowledge graph embeddings. *Bioinformatics* 35, 18 (02 2019), 3538–3540. https://doi.org/10.1093/bioinformatics/btz117

[4] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD '08)*. Association for Computing Machinery, New York, NY, USA, 1247–1250. https://doi.org/10.1145/1376616.1376746

[5] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating Embeddings for Modeling Multi-relational Data. In *Advances in Neural Information Processing Systems*, C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger (Eds.), Vol. 26. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2013/file/1cecc7a77928ca8133fa24680a88d2f9-Paper.pdf

[6] Agustín Borrego, Daniel Ayala, Inma Hernández, Carlos R. Rivero, and David Ruiz. 2019. Generating Rules to Filter Candidate Triples for their Correctness Checking by Knowledge Graph Completion Techniques. In *Proceedings of the 10th International Conference on Knowledge Capture, K-CAP 2019, Marina Del Rey, CA, USA, November 19-21, 2019*. 115–122. https://doi.org/10.1145/3360901.3364418

[7] Agustín Borrego, Daniel Ayala, Inma Hernández, Carlos R Rivero, and David Ruiz. 2019. Generating rules to filter candidate triples for their correctness checking by knowledge graph completion techniques. In *Proceedings of the 10th International Conference on Knowledge Capture*. 115–122.

[8] Agustín Borrego, Danilo Dessì, Inma Hernández, Francesco Osborne, Diego Reforgiato Recupero, David Ruiz, Davide Buscaldi, and Enrico Motta. 2022. Completing Scientific Facts in Knowledge Graphs of Research Concepts. *IEEE Access* 10 (2022), 125867–125880.

[9] Samuel Broscheit, Daniel Ruffinelli, Adrian Kochsiek, Patrick Betz, and Rainer Gemulla. 2020. LibKGE - A Knowledge Graph Embedding Library for Reproducible Research. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. 165–174. https://www.aclweb.org/anthology/2020.emnlp-demos.22

[10] Payal Chandak, Kexin Huang, and Marinka Zitnik. 2023. Building a knowledge graph to enable precision medicine. *Scientific Data* 10, 1 (2023), 67. https://doi.org/10.1038/s41597-023-01960-3

[11] Nur Nasuha Daud, Siti Hafizah Ab Hamid, Muntadher Saadoon, Firdaus Sahran, and Nor Badrul Anuar. 2020. Applications of link prediction in social networks: A review. *Journal of Network and Computer Applications* 166 (2020), 102716.

[12] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. 2018. Convolutional 2D Knowledge Graph Embeddings. *Proceedings of the AAAI Conference on Artificial Intelligence* 32, 1 (Apr. 2018). https://doi.org/10.1609/aaai.v32i1.11573

[13] Rod G Downey and Michael R Fellows. 1995. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on computing* 24, 4 (1995), 873–921.

[14] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research* 12, 7 (2011).

[15] Fan Feng, Feitong Tang, Yijia Gao, Dongyu Zhu, Tianjun Li, Shuyuan Yang, Yuan Yao, Yuanhao Huang, and Jie Liu. 2022. GenomicKB: a knowledge graph for the human genome. *Nucleic Acids Research* 51, D1 (11 2022), D950–D956. https://doi.org/10.1093/nar/gkac957

[16] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[17] Adrian Kochsiek and Rainer Gemulla. 2021. Parallel training of knowledge graph embedding models: a comparison of techniques. *Proceedings of the VLDB Endowment* 15, 3 (2021), 633–645.

[18] Ajay Kumar, Shashank Sheshar Singh, Kuldeep Singh, and Bhaskar Biswas. 2020. Link prediction techniques, applications, and performance: A survey. *Physica A: Statistical Mechanics and its Applications* 553 (2020), 124289.

[19] Peng Lin, Qi Song, and Yinghui Wu. 2018. Fact checking in knowledge graphs with ontological subgraph patterns. *Data Science and Engineering* 3, 4 (2018), 341–358.

[20] Farzaneh Mahdisoltani, Joanna Biega, and Fabian Suchanek. 2014. Yago3: A knowledge base from multilingual wikipedias. In *7th biennial conference on innovative data systems research*. CIDR Conference.

[21] Víctor Martínez, Fernando Berzal, and Juan-Carlos Cubero. 2016. A survey of link prediction in complex networks. *ACM computing surveys (CSUR)* 49, 4 (2016), 1–33.

[22] Tomas Mikolov, Kai Chen, Greg S. Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. http://arxiv.org/abs/1301.3781

[23] George A. Miller. 1995. WordNet: A Lexical Database for English. *Commun. ACM* 38, 11 (nov 1995), 39–41. https://doi.org/10.1145/219717.219748

[24] Aisha Mohamed, Shameem Parambath, Zoi Kaoudi, and Ashraf Aboulnaga. 2020. Popularity agnostic evaluation of knowledge graph embeddings. In *Conference on Uncertainty in Artificial Intelligence*. PMLR, 1059–1068.

[25] Sameh K Mohamed, Vít Nováček, and Aayah Nounu. 2020. Discovering protein drug targets using knowledge graph embeddings. *Bioinformatics* 36, 2 (2020), 603–610.

[26] David N. Nicholson and Casey S. Greene. 2020. Constructing knowledge graphs and their biomedical applications. *Computational and Structural Biotechnology Journal* 18 (2020), 1414–1428. https://doi.org/10.1016/j.csbj.2020.05.017

[27] Maximilian Nickel, Lorenzo Rosasco, and Tomaso Poggio. 2016. Holographic embeddings of knowledge graphs. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 30.

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

[29] Daniel Ruffinelli, Samuel Broscheit, and Rainer Gemulla. 2020. You CAN Teach an Old Dog New Tricks! On Training Knowledge Graph Embeddings. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*.

[30] Tara Safavi and Danai Koutra. 2020. CoDEx: A Comprehensive Knowledge Graph Completion Benchmark. *CoRR* abs/2009.07810 (2020). arXiv:2009.07810 https://arxiv.org/abs/2009.07810

[31] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: A Core of Semantic Knowledge. In *Proceedings of the 16th International Conference on World Wide Web (WWW '07)*. Association for Computing Machinery, New York, NY, USA, 697–706. https://doi.org/10.1145/1242572.1242667

[32] Kristina Toutanova, Danqi Chen, Patrick Pantel, Hoifung Poon, Pallavi Choudhury, and Michael Gamon. 2015. Representing text for joint embedding of text and knowledge bases. In *Proceedings of the 2015 conference on empirical methods in natural language processing*. 1499–1509.

[33] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Eric Gaussier, and Guillaume Bouchard. 2016. Complex Embeddings for Simple Link Prediction. In *Proceedings of The 33rd International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Maria Florina Balcan and Kilian Q. Weinberger (Eds.), Vol. 48. PMLR, New York, New York, USA, 2071–2080. https://proceedings.mlr.press/v48/trouillon16.html

[34] Angelica S Valeriani, Guido Walter Di Donato, and Marco D Santambrogio. 2021. Exploring the Runtime Performance of Knowledge Graph Embedding Methods. In *2021 IEEE 6th International Forum on Research and Technology for Society and Industry (RTSI)*. IEEE, 463–468.

[35] Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Commun. ACM* 57, 10 (2014), 78–85.

[36] Duncan J Watts and Steven H Strogatz. 1998. Collective dynamics of 'small-world'networks. *nature* 393, 6684 (1998), 440–442.

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

[38] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2014. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. https://doi.org/10.48550/ARXIV.1412.6575

[39] Peng Zhang, Jinliang Wang, Xiaojia Li, Menghui Li, Zengru Di, and Ying Fan. 2008. Clustering coefficient and community structure of bipartite networks. *Physica A: Statistical Mechanics and its Applications* 387, 27 (2008), 6869–6875. https://doi.org/10.1016/j.physa.2008.09.006