Extending TWIG: Zero-Shot Predictive Hyperparameter Selection for KGEs based on Graph Structure

Jeffrey Sardina[1,†], John D. Kelleher[1] and Declan O'Sullivan[1]

[1]Trinity College Dublin, the University of Dublin, College Green Dublin, Ireland

Abstract

Knowledge Graphs (KGs) have seen increasing use across various domains – from biomedicine and linguistics to general knowledge modelling. In order to facilitate the analysis of knowledge graphs, Knowledge Graph Embeddings (KGEs) have been developed to automatically analyse KGs and predict new facts based on the information in a KG, a task called "link prediction". Many existing studies have documented that the structure of a KG, KGE model components, and KGE hyperparameters can significantly change how well KGEs perform and what relationships they are able to learn. Recently, the Topologically-Weighted Intelligence Generation (TWIG) model has been proposed as a solution to modelling how each of these elements relate. In this work, we extend the previous research on TWIG and evaluate its ability to simulate the output of the KGE model ComplEx in the cross-KG setting. Our results are twofold. First, TWIG is able to summarise KGE performance on a wide range of hyperparameter settings and KGs being learned, suggesting that it represents a general knowledge of how to predict KGE performance from KG structure. Second, we show that TWIG can successfully predict hyperparameter performance on unseen KGs in the zero-shot setting. This second observation leads us to propose that, with additional research, optimal hyperparameter selection for KGE models could be determined in a pre-hoc manner using TWIG-like methods, rather than by using a full hyperparameter search.

Keywords

Knowledge Graphs, Knowledge Graph Embeddings, Relational Learning, Link Prediction, Simulation

1. Introduction and Preliminaries

Knowledge Graphs (KGs) are graph-based databases that model information as a set of nodes, which represent concepts, and edges, which represent the relationships between them [1]. Knowledge Graph Embedding (KGE) models learn to predict new facts based on the information contained in a knowledge graph – formally, this is called the link prediction task. [2, 3, 4]. As a result of their success in link prediction, KGE models have become increasingly used in a large variety of domains – from modelling health sciences data [5, 6, 7, 8, 9] to general knowledge [10, 11].

While previous studies provided detailed benchmarking of various KGE models [10, 11, 4, 12, 13], explored the effects of specific KGE model components [11, 10, 14, 15], and explored the effects of KG structure on learning [5, 16, 17], no study known to the authors has attempted to create a system in which KGE models, model components, graph structure, and link prediction performance can be understood as part of a common analytic framework. In each of these areas, analysis of KGEs remain incompletely characterised in terms of the others. For example, the manner by which different KGE model components affect the learnability of various graph (sub-)structures has not been explored in detail in the literature known to the authors.

However, recent developments in the Topologically-Weighted Intelligence Generation (TWIG) approach for analysing KGE models have opened the door to characterising KGE models, KG structure, and link prediction performance in a common framework [18]. In this work, we extend the TWIG model to simulate KGEM output on multiple KGs at the same time. We provide an empirical analysis of the accuracy this new TWIG model and show that it can accurately predict the overall performance of KGE models even on previously unseen KGE hyperparameter settings. Finally, we show that it can

AICS'24: 32nd Irish Conference on Artificial Intelligence and Cognitive Science, December 09–10, 2024, Dublin, Ireland
*Corresponding author.
Jeffrey.Sardina@gmail.com (J. Sardina); John.Kelleher@tcd.ie (J. D. Kelleher); Declan.OSullivan@tcd.ie (D. O'Sullivan)
0000-0003-0654-2938 (J. Sardina); 0000-0001-6462-3248 (J. D. Kelleher); 0000-0003-1090-3548 (D. O'Sullivan)

further predict hyperparameter preference and KGE model performance on entirely unseen knowledge graphs; i.e. in the zero shot setting.

The following sections provide a background and motivation for this work. All code can be found at https://anonymous.4open.science/r/TWM-4D1F/README.md, and data files can be found at https://figshare.com/s/13dc93087c97cbf7cca1.

1.1. Knowledge Graphs and Knowledge Graph Embeddings

Knowledge Graphs represent data as atomic facts (also called "triples") consisting of labelled nodes and the directed, labelled relations that occur between them. Triples in a KG are denoted as (s, p, o), where s represents the subject (or "head") node, o represents the object (or "tail") node, and p is the "predicate" that describes the relationship between s and o [1]. The intrinsically networked nature of Knowledge Graphs leads them to very naturally represent a variety of real world data, from biological pathways and biomedical data [19, 20, 21] to linguistics [22] and general knowledge [23, 22].

Knowledge Graph Embeddings are the machine learning counterpart to KGs – they aim to automatically learn to represent all of the knowledge in a KG as latent vector embeddings of each node and edge [1, 2, 3]. These embeddings can then be used to predict new statements that should be present in a KG, allowing the inference of new knowledge from the knowledge already present in a KG – a task called "link prediction". The link prediction task is formally defined as answering a "link prediction query" in the form $(s, p, ?)$ or $(?, p, o)$, where $?$ represents the subject or object entity that should be predicted such that the triple would be true.

To solve this task, KGE models learn to calculate a plausibility score for all triples. This scoring function takes the form:

$$f(e_s, e_p, e_o) \rightarrow score_{s,p,o}$$

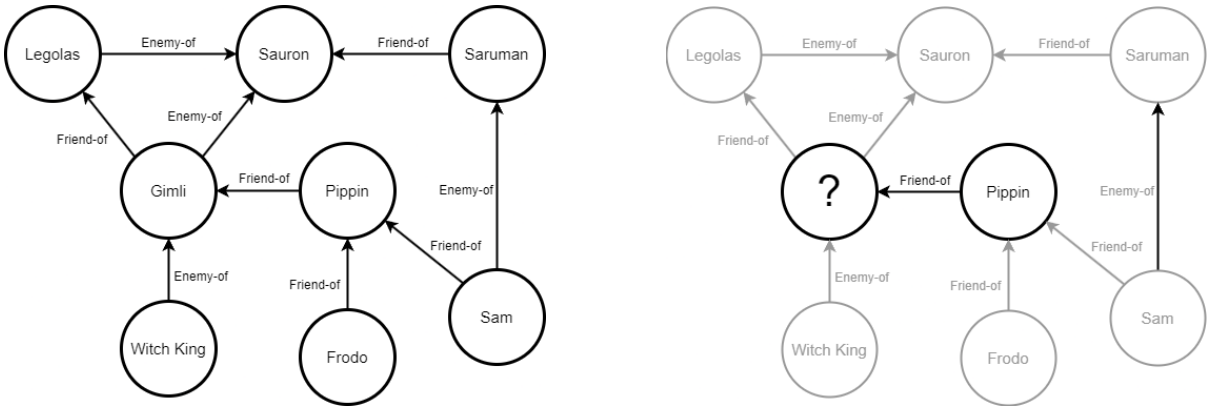
where e_s represents the embedding of the subject node, e_p represents the embedding of the predicate, e_o represents the embedding of the object node, and $score_{s,p,o}$ is a scalar-valued plausibility score output by the function for the given triple.

An example Knowledge Graph is given in Figure 1a; in this graph, all nodes represent people and all relations indicate if the people consider others to be friends or enemies. In such a graph, the link prediction task represents asking if a certain person is the friend of, or the enemy of, another. An example of link prediction in this KG is shown in Figure 1b. In this case, the query triple can be represented as $(Pippin, Friend\text{-}of, ?)$. The remainder of the KG is not part of the query, but is used as training examples for KGE models – in other words, it represents the background knowledge that is used by KGE models to answer the posed query.

Evaluation of KGE models is based on how well the model is able to assign higher plausibility scores to known-true triples (in a KG's hold-out test set) and to assign lower plausibility scores to all other triples (not observed in the KG). A schematic overview of this procedure is shown in Figure 2, and an algorithmic specification follows.

At an algorithmic level, in order to evaluate the performance of a KGE model on a given triple (s, p, o), two "link prediction queries" are posed based on that triple: $(s, p, ?)$ and $(?, p, o)$. For each query, all possible entities in the KG are substituted for the unknown entity $?$, and all resultant triples $(s, p, \hat{o_i})$ and $(\hat{s_i}, p, o)$ are scored. The scores of all link prediction queries are then sorted into two ranked lists: the list of answers to $(s, p, ?)$, and the list of answers to $(?, p, o)$; both lists are sorted by score such that triples with higher plausibility scores come first in the list. The rank of the correct answers (s^*, p, o) and (s, p, o^*) in the sorted list is then calculated. Lower ranks (closer to 1) indicate that the correct answer to the link prediction query is predicted to be more plausible than its incorrect alternatives, whereas higher ranks (further from 1) indicate that the correct answer is predicted to be less plausible than its incorrect alternatives.

This procedure is done for all triples in the KG's hold-out test set. Once all ranks are obtained for all link prediction queries, overall performance is measured using the standard Mean Reciprocal Rank



(a) An example Knowledge Graph where nodes represent people and edges represent whether they are friends or enemies.

(b) An example of a query for link prediction in the knowledge graph (in black) based on training data (in grey).

Figure 1: A sample Knowledge graph (left) and an example of link prediction (right).

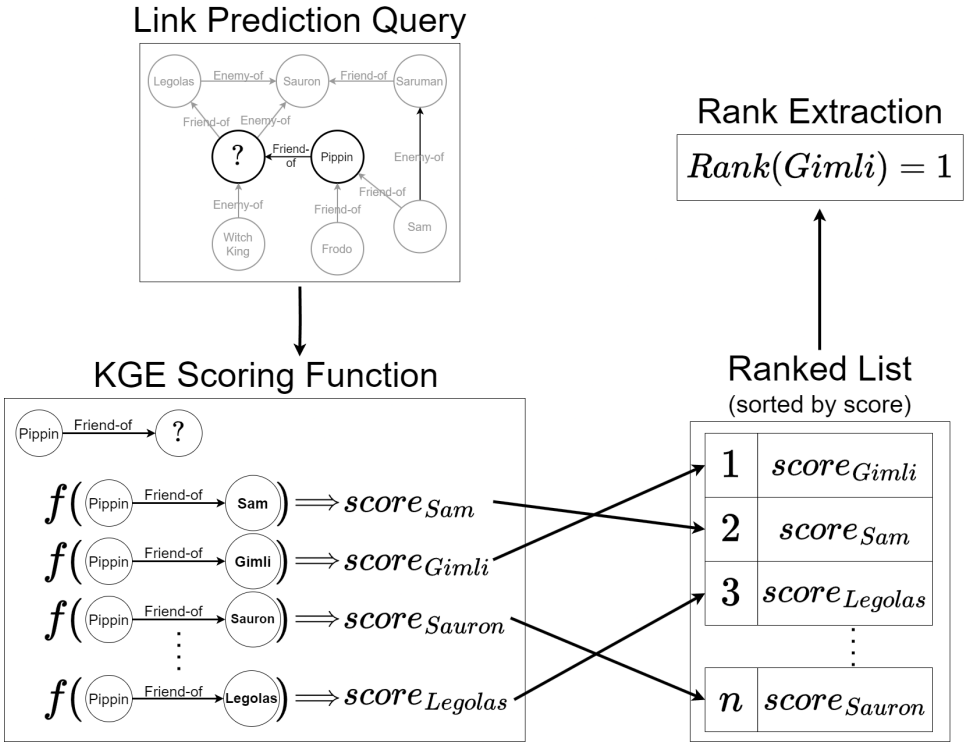


Figure 2: An schematic overview of how rank-based evaluation of KGE models is performed.

(MRR) metric [10, 11]. Specifically, MRR calculates the mean of the reciprocal of the ranks assigned to the correct answers of all link prediction queries. Mathematically, this is expressed as:

$$MRR = \frac{\sum_{i=1}^{n} \frac{1}{rank_i}}{n}$$

where n is the total number of link prediction queries posed and $rank_i$ is the rank of the correct answer to the i^{th} link prediction query. MRR values are bounded on the interval $(0, 1]$. Values closer to 1 indicate better performance, and values closer to 0 indicate worse performance.

1.2. Properties of KGE Models

KGE models have three main components

- a **scoring function** that uses embeddings to assign a plausibility score to a triple,
- a **negative sampler** that produces counter-examples (in the form of fake triples) during training, and
- a **loss function** that forces the KGE model to assign higher scores to true triples and lower scores to "negative" / fake triples.

On top of these, every KGE model has various hyperparameters, such as its learning rate, the size of the embeddings, its regularisation coefficients, and the number of negatives triples to generate during training. All of these hyperparameters affect how KGE learning proceeds.

Existing analysis of KGE-based link predictors has shown that the optimal choices for various KGE model components [11, 10, 18], such as the hyperparameters [18, 24], the loss function [15, 11, 10, 18], and the negative sampler [14, 11, 10, 18] can be understood in terms of the KGE model in use and of the structure of the knowledge graph that is being learned.

Kotnis et al. show that the optimal negative sampler choice depends on KG structure, and suggest that this structural dependence is a significant source of hyperparameter preference in KGE models [14]. However, they do note that the KGE model used also has an impact on the optimal negative sampler, citing that less expressive KGE models such as TransE cannot always benefit from more robust negative sampling protocols [14]. In a similar vein, Sameh et al. show that optimal loss function choice varies both by the KGE model used and by the KG being learned [15]. Ruffinelli et al. and Ali et al. both conducted mass KGE benchmark studies, performing ablations of a wide variety of KGE model components on various KGs of differing structures [10, 11]. Overall, Ruffinelli et al. and Ali et al. both found evidence of complex systems of preference for various KGE components and KGE hyperparameters when learning different KGs, but did relatively little analysis on those systems or what gave rise to them [10, 11].

Finally, a distinct body of works has explored the effectiveness of KGE models directly as a function of graph structure [5, 4, 18, 24]. Bonner et al. showed that so-called "super-hub" nodes of extremely high degree could substantially impact learning, often leading to worse performance in link prediction by being over-represented in model predictions [5]. They also highlight that nodes with very low degree (near 1, meaning that the node only connects to a select few other nodes in the graph) are much harder to learn during link prediction [5].

Rossi et al. showed that KGE performance can be modelled as a function of the properties of nodes and relationships – such as the how often certain nodes and relationships co-occur in the graph [4].

Finally, Sardina et al. showed that the rank assigned to link prediction queries during evaluation, as well as the overall performance of KGEs, can be accurately modelled using a simulation method called Topologically-Weighted Intelligence Generation (TWIG) [18]. While these results are only evaluated on a single dataset, they show a notable ability to summarise hyperparameter preference and KG learnability using a simple neural network model [18].

Taken together, these results suggest that the performance of KGEs can be understood largely as a function of their hyperparameters / components and the structural properties of the knowledge graphs that they learn. The result of this is the theoretical possibility of predicting hyperparameter preference and KGEM performance not only in the supervised setting, but also in the few-shot and zero-shot settings – in other words, to perform predictive hyperparameter optimisation using TWIG as a substitute to a hyperparameter search.

2. Methodology

The methodology of our work can be divided into two parts – training and evaluating TWIG in a multi-KG setting, and how we define the few-shot and zero-shot evaluations for TWIG.

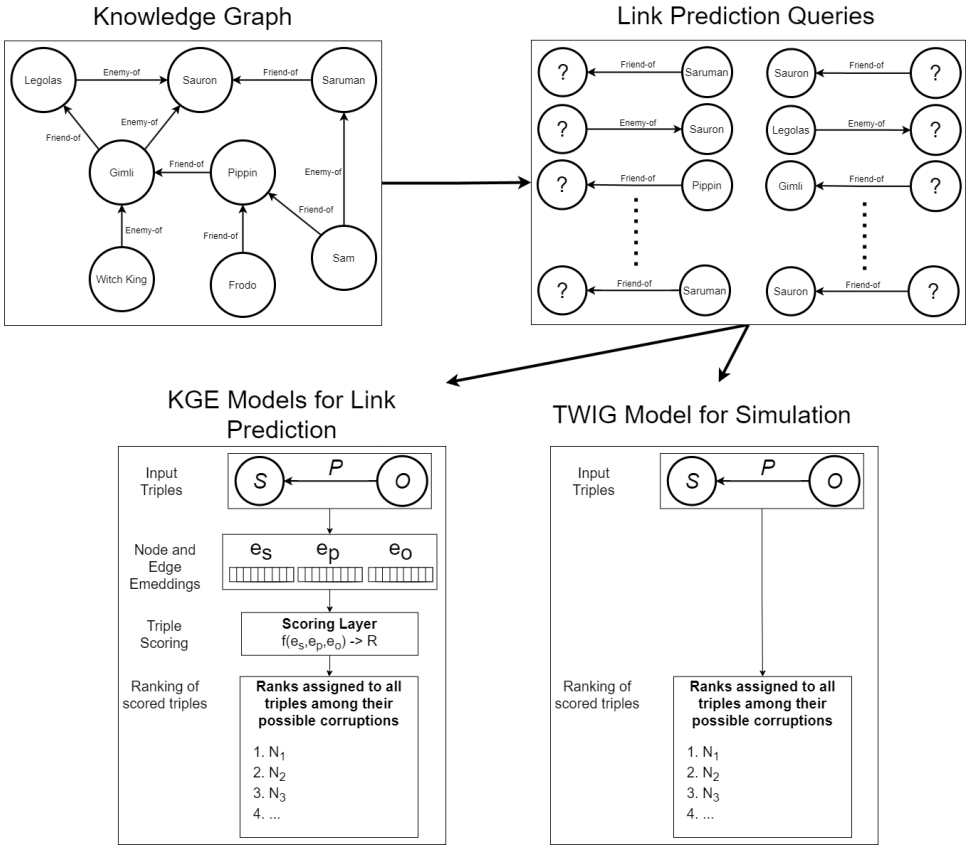


Figure 3: An overview of link prediction (using KGEs) and of KGE simulation (using TWIG).

2.1. The TWIG Model

The job of TWIG is to simulate a single knowledge graph embedding model (such as ComplEx [25]) by predicting the ranks that the KGE model would assign to link prediction queries in the form $(s, p, ?)$ or $(?, p, o)$. To do this, we use the TWIG neural network as published in the original TWIG paper [18]. The TWIG neural network has three major learning components – a hyperparameter learning component, a graph structure learning component, and an integration component. The first two components learn to implicitly represent KGE hyperparameters and graph structure respectively, and the integration component combines their information to produce the final output of predicted ranks. An overview of the TWIG simulation pipeline is given in Figure 3, and the architecture of the TWIG neural network is given in Figure 4.

In terms of input, TWIG gathers the values of all hyperparameters, as well as the specific negative sampler and loss function being used, directly as input. It further collects fine-grained structural information on every triple in the KG, as well as aggregate statistics on each triple's local neighbourhood. Full details on all hyperparameter and structural features used by TWIG can be found in Table 1; note that these are the same features used in the original TWIG paper [18].

For simplicity, a visual depiction of the fine-grained and coarse-grained structural information that TWIG collects is shown in Figure 5. As a final note, we highlight that all structural features are calculated based only on the relations present in the KG's training set to avoid data leakage.

2.2. Data and Datasets

We selected five KGs to use – CoDExSmall, DBpedia50, Kinships, OpenEA, and UMLS [26, 27, 28, 29, 19], all taken from the publicly-available PyKEEN repository [30]. We choose these graphs for two reasons. First, they represent a diverse set of domains – notably, biology (UMLS [19]), family trees (Kinships [28]), and general knowledge (CoDExSmall, DBpedia50, and OpenEA [27, 26, 29]). Second, all datasets

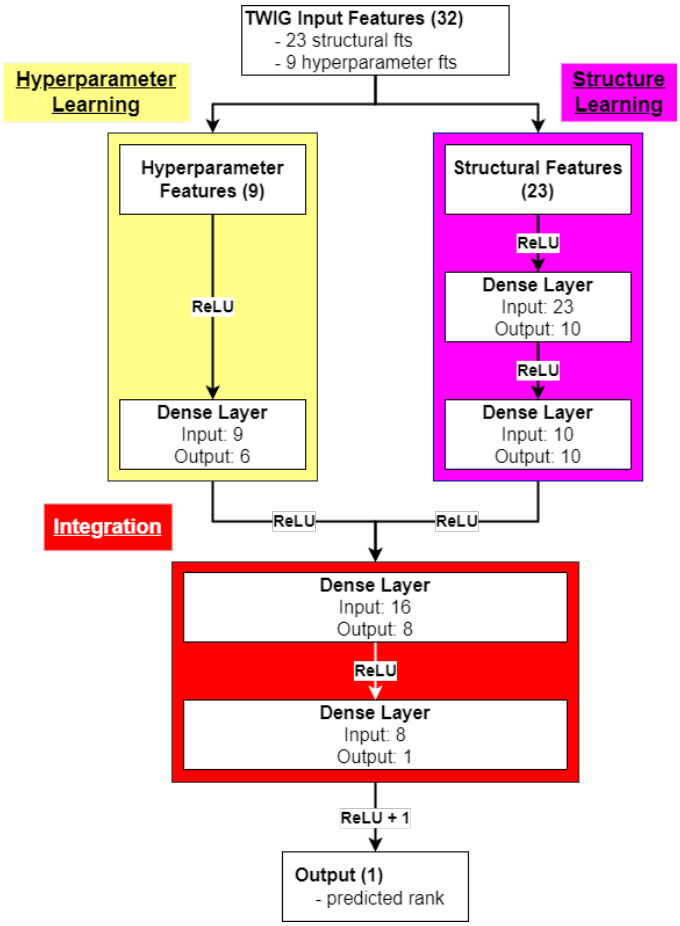


Figure 4: A visualisation of the TWIG neural network architecture. KG structure (in magenta) and hyperparameter influence (in yellow) are first learned in separate blocks, then combined in an integration component (in red) before predicted ranks are output.

are relatively small, meaning that performing large hyperparameter benchmarking experiments to produce ground-truth data for TWIG could be done in a feasible amount of time.

In this work, we choose to simulate the state-of-the-art KGE model ComplEx, as it is among the strongest KGE models and has consistent use in many applications [25, 13, 10, 11]. To gather information on ComplEx's performance for TWIG, we train ComplEx on a large hyperparameter grid on all five KGs, and record the ranks it assigns to all link prediction queries, as well as the overall MRR ComplEx achieves on each KG, for all hyperparameter settings. This is done a total of four times, to produce 4 replicates (differing only by random initialisations) of ranked lists to simulate for each KG and each hyperparameter combination.

We use the same hyperparameter grid as used in the original TWIG paper for training ComplEx; that grid is shown in Table 2. The meaning of each hyperparameter can be found in Table1.

2.3. Evaluation of TWIG

We evaluate TWIG in two settings: simulation of KGE performance on unseen hyperparameters, and simulation of KGE performance on unseen KGs and on unseen hyperparameters simultaneously.

2.3.1. Evaluation of TWIG on Unseen Hyperparameters

When evaluating TWIG on unseen hyperparameters, we define our hold-out test set as a random 10% of all hyperparameter combinations for each KG TWIG is trained on. This is done such that the exact same hyperparameter combinations are held out on each different KG TWIG is trained on, so that when

Feature	Meaning
Hyperparameter Fts	
Negative Sampler	The negative sampling strategy used
#Negatives per Positive	The number of negatives sampled for each triple in training
Loss Function	The loss function used
Margin (if applicable)	The margin value used in loss calculation (if applicable)
Learning Rate	The learning rate for the Adam optimiser
Embedding dimension	The dimension of KGE embeddings
Regularisation Coefficient	The coefficient of the regulariser
KG Structural Fts	
is_head	True if the link prediction query is $(?, p, o)$; else false
s_deg	The degree of the subject node
o_deg	The degree of the object node
p_freq	The frequency of the predicate
s-p cofreq	The number of times the given subject and predicate co-occur
o-p cofreq	The number of times the given object and predicate co-occur
s-o cofreq	The number of times the given subject and object co-occur
s min deg neighbnour	The degree of the lowest-degree neighbour of the subject
s max deg neighbnour	The degree of the highest-degree neighbour of the subject
s mean deg neighbnour	The degree of the mean-degree neighbour of the subject
o min deg neighbnour	The degree of the lowest-degree neighbour of the object
o max deg neighbnour	The degree of the highest-degree neighbour of the object
o mean deg neighbnour	The degree of the mean-degree neighbour of the object
s num neighbnours	The total number of neighbours the subject has
o num neighbnours	The total number of neighbours the object has
s min freq edge	The frequency of the least-frequent edge linked to the subject
s max freq edge	The frequency of the most-frequent edge linked to the subject
s mean freq edge	The mean frequency of edges linked to the subject
o min freq edge	The frequency of the least-frequent edge linked to the object
o max freq edge	The frequency of the most-frequent edge linked to the object
o mean freq edge	The mean frequency of edges linked to the object
s num edges	The total number of edges incident on the subject
o num edges	The total number of edges incident on the object

Table 1
A description of all features used in the TWIG model.

Hyperparameter	Values Searched
Negative Sampler	Basic, Bernoulli, Pseudo-Typed
#Negatives per Positive	5, 25, 125
Loss Function	Margin Ranking, Binary Cross Entropy, Cross Entropy
Margin (if applicable)	0.5, 1, 2
Learning Rate	1e-2, 1e-4, 1e-6
Embedding dimension	50, 100, 250
Regularisation Coefficient	1e-2, 1e-4, 1e-6
Epochs	100 (constant)

Table 2
The grid of hyperparameters used to train the KGE model ComplEx simulated by TWIG.

it is tested, it is tested on hyperparameter combinations it has never seen before. The remaining 90% of hyperparameter combinations are used as the training set.

TWIG is then trained to simulate the output of all hyperparameter combinations on all five knowledge graphs for the KGE model ComplEx. This training is run exactly as in the original TWIG paper; that is, training is done in two phases, with 5 epochs for the first phase and 10 epochs for the second phase [18]. In the first phase TWIG learns using KL-divergence loss alone to teach it to match the expected distribution of ranks in its output; in the second phase TWIG learns using both KL-divergence loss and

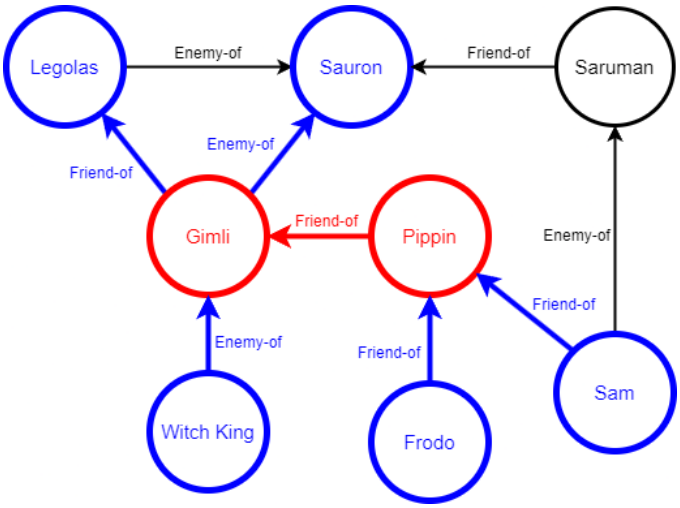


Figure 5: A visualisation of the region around a triple that TWIG uses to calculate structural features. Details statistics are gathered for the main triple (shown in red), and aggregate statistics are gathered for all triples connecting to that triple (shown in blue). Other triples (shown in black) are ignored.

Mean Squared Error (MSE) loss to teach it to more exactly match the values of the ranks it is meant to predict while also maintaining the expected distribution of ranks [18]. Full details of this training setup, and the reasons behind 2-phase training, can be found in the TWIG paper [18].

After training TWIG, we evaluate performance by its ability to correctly predict the Mean Reciprocal Rank (MRR) value that the KGE model ComplEx would achieve on each KG under each hyperparameter setting in the hold-out test set. To do this, we first use all of TWIG's predicted ranks for each hyperparameter combination to produce a predicted MRR score. We then calculate the $R2$ metric between ground-truth MRR values and TWIG's predicted MRR values. Note that the $R2$ metric is bounded on $(-\infty, 1]$, with higher values (closer to 1) indicating better performance and lower values indicating worse performance.

2.3.2. Evaluation of TWIG on Unseen KGs and Unseen Hyperparameters

When evaluating TWIG on unseen KGs, we define one of the five KGs as a hold-out set test, such that TWIG never sees the output of ComplEx on that KG during training. This is used as the first hold-out test set. For the remaining 4 KGs, we again remove a random 10% of the hyperparameter combinations as a second hold-out test set.

We then train on the training set of the remaining 4 KGs, which consists of 90% of the hyperparameter combinations for each KG. In order to evaluate how well TWIG can simulate the performance of ComplEx on unseen KGs, we evaluate TWIG on *all* hyperparameter combinations of the hold-out KG – which includes hyperparameter combinations it has previously seen in training, as well as hyperparameter combinations never seen during training. Once again, evaluation is done in terms of the $R2$ metric, this time between predicted MRR for each hyperparameter setting on the hold-out KG and on the true MRR value achieved by ComplEx on the same hyperparameter settings for the hold-out KG. This evaluation setting, in which an entire KG is held out, is referred to as the "0-shot" evaluation setting.

As a generalisation of 0-shot evaluation, we also explore how TWIG performs when it is trained on a small percent of hyperparameter combinations in the unseen KG. We phrase this as a transfer learning problem, where we use the TWIG model trained on four of the five KGs as a pretrained model, and then finetune that model onto the hold-out KG. We do this in two cases – where TWIG can see a random 5% of the hold-out KG during finetuning, and where TWIG can see a random 25% of the hold-out KG during finetuning. In both cases, the remaining 95% or 75% of the hyperparameters are reserved as a hold-out test set. We refer to these finetuning experiments as the "5%-shot and 25%-shot" settings respectively.

Finally, we also evaluate TWIG's ability to simulate ComplEx on each of the four seen KGs on their 25% hold-out hyperparameter combinations, exactly as done when testing TWIG on unseen hyperparameters only. The results of all of these tests – on seen KGs and on and unseen KGs in the 0-shot, 5%-shot, and 25%-shot evaluation settings, are given in the Results section.

3. Results and Discussion

In this section, we first demonstrate that the TWIG model we use simulates the KGE model ComplEx with high accuracy on both unseen hyperparameter combinations and on unseen KGs. We further show that TWIG can be effectively finetuned, allowing it to be re-purposed to new KGs with minimal effort and very little data for finetuning.

3.1. Evaluation on Unseen Hyperparameters

Table 3 gives the results of TWIG when trained on 90% of all hyperparameter combinations of all 5 KGs, and tested on the remaining 10% of hyperparameter combinations on the same KGs.

	CoDExSmall	**DBpedia50**	**Kinships**	**OpenEA**	**UMLS**
TWIG R^2	0.95	0.72	0.98	0.8	0.96

Table 3
$R2$ values achieved by TWIG for each KG when trained jointly on the training sets of all KGs. $R2$ is calculated between observed and predicted MRR values for each hyperparameter combination on each KG.

These results indicate that TWIG can generalise across various KGs and effectively simulate the outputs of KGEs on many different hyperparameter settings and many different KGs. The $R2$ values of TWIG on each dataset all lie between 0.72 at the lowest (on DBpedia50) and 0.98 at the highest (on Kinships), indicating that it can accurately predict the performance of ComplEx on all KGs under various hyperparameter settings. We further highlight that as this evaluation is done on hold-out hyperparameters, that it indicates that TWIG is able to predict the efficacy of hyperparameter combinations it has never seen.

3.2. Evaluation on Unseen KGs

Table 4 gives all results of TWIG when trained on 90% of all hyperparameter combinations of 4 KGs, and tested:

- on the remaining 10% of hyperparameter combinations on the 4 KGs seen during training (shown on the left)
- on all hyperparameter combinations on a hold-out KG never seen during training (shown on the right)

Table 4 further shows the results of finetuning TWIG with either a random 5% (for the 5%-shot setting) or a random 25% (for the 25%-shot setting) of all hyperparameter combinations on the hold-out KG before evaluating on that KG. All combinations of 4 training KGs and one hold-out KG are shown.

The results indicate that TWIG is able to simulate ComplEx not only on unseen hyperparameters, but also on entirely unseen KGs. Particularly notably, TWIG achieves an $R2$ of between 0.54 (at the lowest) and 0.73 (at the best) in the 0-shot setting where it is not trained on any information about the hold-out KG. This improves to 0.77 (at the worst) to 0.96 (at the best) in the 5%-shot setting and 0.86 to 0.99 in the 25%-shot setting, where 5% or 25% of the hold-out KG is used to finetune TWIG before evaluation.

We highlight that these results have two immediate, major implications:

- that TWIG is able to use a general knowledge of KG structure and hyperparameter effects to simulate KGEs on datasets and hyperparameter combinations it has never seen in training, and

Training KGs				Testing KG		
				0-shot	**5%-shot**	**25%-shot**
CodExSmall	DBpedia50	Kinships	OpenEA		UMLS	
0.95	0.81	0.98	0.83	0.64	0.91	0.97
CodExSmall	DBpedia50	Kinships	UMLS		OpenEA	
0.95	0.85	0.94	0.96	0.54	0.77	0.97
CodExSmall	DBpedia50	OpenEA	UMLS		Kinships	
0.97	0.83	0.83	0.86	0.65	0.90	0.99
CodExSmall	UMLS	Kinships	OpenEA		DBpedia50	
0.95	0.93	0.95	0.88	0.73	0.81	0.86
UMLS	DBpedia50	Kinships	OpenEA		CoDExSmall	
0.98	0.88	0.98	0.89	0.73	0.96	0.98

Table 4
R^2 values achieved by TWIG in the on unseen hyperparameters, as well as in the 0-shot, 5%-shot, and 25%-shot settings on all combinations of 4 training KGs and 1 unseen KG. $R2$ is calculated between ground truth and predicted MRR values for each hyperparameter combination on each KG.

- that TWIG is highly receptive to finetuning, and that finetuning even on small amounts of data can lead to substantial increases in its ability to simulate KGEs.

We further highlight that the high performance of the 5%-shot and 25%-shot evaluation setting is not unexpected – since TWIG is finetuned on only one dataset, rather than having to simulate KGEs on all datasets, it has a much higher ability to learn the details of that KG in particular. When trained on multiple KGs, TWIG's performance drops somewhat due to the need to simulate KGEs in so much more diverse of an environment. In fact, a weaker version of this effect is seen in the slight per-KG increase in performance on unseen hyperparameters observed when TWIG is trained 4 KGs only (as in Table 4), versus when it is trained on all 5 (as in Table 3).

Finally, it is important to highlight that zero-shot and few-show prediction both work *regardless of the domain of the unseen KG*. For example, when predicting hyperparameter preference for Kinships (the only KG containing family-tree data) and UMLS (the only KG containing biological data), R2 values of 0.65 and 0.64 are obtained respectively. In both cases, no family tree data, nor biological data, was present in the KGs TWIG was trained on. This suggests that structural impact on hyperparameters may be domain-agnostic, allowing it to generalise across KGs even from different knowledge domains and with distinct information content.

4. Conclusion

In this paper, we extend TWIG, a novel system that can predict KGE model performance and hyperparameter preference for unseen KGs based on graph structure. Our results indicate that TWIG can accurately predict and represent the overall performance of the KGE model ComplEx on both unseen hyperparameters and on unseen KGs. This last ability, its ability to predict overall KGE performance for various hyperparameter sets on unseen KGs, is very significant for several reasons.

First, it suggests that hyperparameter preference is a function of KG structure. Since TWIG is able to make this prediction across KGs from different domains, it further highlights that structural are fundamental to KGE-based KG learning. It is therefore possible that the information content of the KG, paradoxically, may be less important than KG structure for determining how well the KG can be learned by KGE models.

Second, the substantively increased performance on formerly unseen KGs in the few-shot setting suggests that TWIG could possibly serve as a replacement for large-scale hyperparameter searches. Since TWIG can predict hyperparameter preference with high accuracy when trained on only 5% or 25% of a hyperparameter grid, and since this effect persists across various KGs from different domains, there is strong initial evidence that it could be used in lieu of a traditional hyperparameter search.

Finally, the ability of TWIG to model hyperparameter preference and KGE model performance across multiple KGs (seen and unseen) using their structural characteristics suggests that some elements of KG structure are common among different KGs. For example, Bonner et al. [5] have shown that, in the presence of strongly skewed distributions of node degrees, KGE learning can be heavily biased. It is very possible that a variety of effect such as this underpin TWIG's ability to predict KGE performance. Determining the exact nature of such structural relations, and to what extent TWIG uses them in its predictions, is left for future work.

We must finally acknowledge some limitations of this study. Most notably, only one KGE model (ComplEx) has been tested here. While our results clearly show that TWIG can generalise across KGs, it remains unknown if it can generalise across different KGE models. We leave testing other literature-standard models, such as DistMult and TransE, to future directions.

Further, all KGs examined in this work are relatively small, especially compared to the standard benchmark KGs FB15k-237 and WN18RR. It is, as such, unclear how TWIG reacts to KGs of different size but otherwise similar structure, or if TWIG can effectively generalise across KGs of much more variable size. Evaluating TWIG in these settings is also left as a future direction.

Acknowledgements

This research was conducted with the financial support of Science Foundation Ireland D-REAL CRT under Grant Agreement No. 18/CRT6225 at the ADAPT SFI Research Centre at Trinity College Dublin, together with sponsorship of Sonas Innovation Ireland. The ADAPT SFI Centre for Digital Content Technology is funded by Science Foundation Ireland through the SFI Research Centres Programme and is co-funded under the European Regional Development Fund (ERDF) through Grant # 13/RC/2106_P2.

References

[1] A. Hogan, E. Blomqvist, M. Cochez, C. D'amato, G. D. Melo, C. Gutierrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier, A.-C. N. Ngomo, A. Polleres, S. M. Rashid, A. Rula, L. Schmelzeisen, J. Sequeda, S. Staab, A. Zimmermann, Knowledge graphs, ACM Comput. Surv. 54 (2021). URL: https://doi.org/10.1145/3447772. doi:10.1145/3447772.

[2] Q. Wang, Z. Mao, B. Wang, L. Guo, Knowledge graph embedding: A survey of approaches and applications, IEEE Transactions on Knowledge and Data Engineering 29 (2017) 2724–2743. doi:10.1109/TKDE.2017.2754499.

[3] M. Nickel, K. Murphy, V. Tresp, E. Gabrilovich, A review of relational machine learning for knowledge graphs, Proceedings of the IEEE 104 (2016) 11–33. doi:10.1109/JPROC.2015.2483592.

[4] A. Rossi, D. Barbosa, D. Firmani, A. Matinata, P. Merialdo, Knowledge graph embedding for link prediction: A comparative analysis, ACM Transactions on Knowledge Discovery from Data 15 (2021) 1–49. doi:10.1145/3424672.

[5] S. Bonner, U. Kirik, O. Engkvist, J. Tang, I. P. Barrett, Implications of topological imbalance for representation learning on biomedical knowledge graphs, Briefings in bioinformatics 23 (2022) bbac279.

[6] S. K. Mohamed, V. Nováček, A. Nounu, Discovering protein drug targets using knowledge graph embeddings, Bioinformatics 36 (2020) 603–610.

[7] R. Zhang, D. Hristovski, D. Schutte, A. Kastrin, M. Fiszman, H. Kilicoglu, Drug repurposing for covid-19 via knowledge graph completion, Journal of biomedical informatics 115 (2021) 103696.

[8] S. K. Mohamed, A. Nounu, V. Nováček, Drug target discovery using knowledge graph embeddings, in: Proceedings of the 34th ACM/SIGAPP symposium on applied computing, 2019, pp. 11–18.

[9] R. Celebi, H. Uyar, E. Yasar, O. Gumus, O. Dikenelli, M. Dumontier, Evaluation of knowledge graph embedding approaches for drug-drug interaction prediction in realistic settings, BMC bioinformatics 20 (2019) 1–14.

[10] M. Ali, M. Berrendorf, C. T. Hoyt, L. Vermue, M. Galkin, S. Sharifzadeh, A. Fischer, V. Tresp, J. Lehmann, Bringing light into the dark: A large-scale evaluation of knowledge graph embedding models under a unified framework, IEEE Transactions on Pattern Analysis and Machine Intelligence 44 (2021) 8825–8845.
[11] D. Ruffinelli, S. Broscheit, R. Gemulla, You can teach an old dog new tricks! on training knowledge graph embeddings, in: ICLR, 2020.
[12] R. Kadlec, O. Bajgar, J. Kleindienst, Knowledge base completion: Baselines strike back, arXiv preprint arXiv:1705.10744 (2017).
[13] P. Jain, S. Rathi, S. Chakrabarti, et al., Knowledge base completion: Baseline strikes back (again), arXiv preprint arXiv:2005.00804 (2020).
[14] B. Kotnis, V. Nastase, Analysis of the impact of negative sampling on link prediction in knowledge graphs, arXiv preprint arXiv:1708.06816 (2017).
[15] S. K. Mohamed, V. Novácek, P.-Y. Vandenbussche, E. Muñoz, Loss functions in knowledge graph embedding models, in: DL4KG@ESWC, 2019.
[16] H. Zhang, T. Zheng, J. Gao, C. Miao, L. Su, Y. Li, K. Ren, Data poisoning attack against knowledge graph embedding, arXiv preprint arXiv:1904.12052 (2019).
[17] P. Bhardwaj, J. Kelleher, L. Costabello, D. O'Sullivan, Adversarial attacks on knowledge graph embeddings via instance attribution methods, arXiv preprint arXiv:2111.03120 (2021).
[18] J. Sardina, J. D. Kelleher, D. O'Sullivan, Twig: Towards pre-hoc hyperparameter optimisation and cross-graph generalisation via simulated kge models, in: 2024 IEEE 18th International Conference on Semantic Computing (ICSC), 2024, pp. 122–129. doi:10.1109/ICSC59802.2024.00025.
[19] A. T. McCray, A. Burgun, O. Bodenreider, Aggregating umls semantic types for reducing conceptual complexity, Studies in health technology and informatics 84 (2001) 216.
[20] P. Chandak, K. Huang, M. Zitnik, Building a knowledge graph to enable precision medicine, Scientific Data 10 (2023) 67.
[21] M. Dumontier, A. Callahan, J. Cruz-Toledo, P. Ansell, V. Emonet, F. Belleau, A. Droit, Bio2rdf release 3: a larger connected network of linked data for the life sciences, in: Proceedings of the 2014 international conference on posters & demonstrations track, volume 1272, Citeseer, 2014, pp. 401–404.
[22] K. Toutanova, D. Chen, Observed versus latent features for knowledge base and text inference, in: A. Allauzen, E. Grefenstette, K. M. Hermann, H. Larochelle, S. W.-t. Yih (Eds.), Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality, Association for Computational Linguistics, Beijing, China, 2015, pp. 57–66. URL: https://aclanthology.org/W15-4007. doi:10.18653/v1/W15-4007.
[23] F. Mahdisoltani, J. Biega, F. M. Suchanek, A knowledge base from multilingual wikipedias–yago3, Technical report, Telecom ParisTech (2014).
[24] J. Sardina, D. O'Sullivan, Structural characteristics of knowledge graphs determine the quality of knowledge graph embeddings across model and hyperparameter choices (2020).
[25] T. Lacroix, N. Usunier, G. Obozinski, Canonical tensor decomposition for knowledge base completion, in: International Conference on Machine Learning, PMLR, 2018, pp. 2863–2872.
[26] T. Safavi, D. Koutra, Codex: A comprehensive knowledge graph completion benchmark, arXiv preprint arXiv:2009.07810 (2020).
[27] B. Shi, T. Weninger, Open-world knowledge graph completion, in: Proceedings of the AAAI conference on artificial intelligence, volume 32, 2018.
[28] C. Kemp, J. B. Tenenbaum, T. L. Griffiths, T. Yamada, N. Ueda, Learning systems of concepts with an infinite relational model, in: AAAI, volume 3, 2006, p. 5.
[29] Z. Sun, Q. Zhang, W. Hu, C. Wang, M. Chen, F. Akrami, C. Li, A benchmarking study of embedding-based entity alignment for knowledge graphs, arXiv preprint arXiv:2003.07743 (2020).
[30] M. Ali, M. Berrendorf, C. T. Hoyt, L. Vermue, S. Sharifzadeh, V. Tresp, J. Lehmann, PyKEEN 1.0: A Python Library for Training and Evaluating Knowledge Graph Embeddings, Journal of Machine Learning Research 22 (2021) 1–6. URL: http://jmlr.org/papers/v22/20-825.html.