Knowledge Graphs Can be Learned with Just Intersection Features

Duy Le [1] Shaochen (Henry) Zhong [2] Zirui Liu [2] Shuai Xu [1] Vipin Chaudhary [1] Kaixiong Zhou [3]
Zhaozhuo Xu [4]

Abstract

Knowledge Graphs (KGs) are potent frameworks for knowledge representation and reasoning. Nevertheless, KGs are inherently incomplete, leaving numerous uncharted relationships and facts awaiting discovery. Deep learning methodologies have proven effective in enhancing KG completion by framing it as a link prediction task, where the goal is to discern the validity of a triple comprising a head, relation, and tail. The significance of structural information in assessing the validity of a triple within a KG is well-established. However, quantifying this structural information poses a challenge. We need to pinpoint the metric that encapsulates the structural information of a triple and smoothly incorporate this metric into the link prediction learning process. In this study, we recognize the critical importance of the intersection among the k-hop neighborhoods of the head, relation, and tail when determining the validity of a triple. To address this, we introduce a novel randomized algorithm designed to efficiently generate intersection features for candidate triples. Our experimental results demonstrate that a straightforward fully-connected network leveraging these intersection features can surpass the performance of established KG embedding models and even outperform graph neural network baselines. Additionally, we highlight the substantial training time efficiency gains achieved by our network trained on intersection features.

[1]Department of Computer and Data Sciences, Case Western Reserve University [2]Department of Computer Science, Rice University [3]Department of Electrical and Computer Engineering, North Carolina State University [4]Department of Computer Science, Stevens Institute of Technology. Correspondence to: Zhaozhuo Xu <zxu79@stevens.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

1. Introduction

Knowledge Graphs (KGs) stand as powerful structures for knowledge representation and reasoning, facilitating the organization of vast amounts of information into entities and relations (Wang et al., 2017; Rossi et al., 2021). Despite their undeniable utility, KGs inherently harbor incompleteness, leaving a plethora of undiscovered relations and unverified facts concealed within their intricate networks (Shi & Weninger, 2018; Destandau & Fekete, 2021). In response to this challenge, deep learning have emerged as invaluable tools for enhancing knowledge completion (Bordes et al., 2013; Wang et al., 2014; Lin et al., 2015; Yang et al., 2015; Trouillon et al., 2016; Zhang et al., 2017; Li et al., 2018; Sun et al., 2019; Nathani et al., 2019; Li et al., 2022a). These methodologies cast the task of uncovering latent facts within KGs as a link prediction challenge, with the aim of determining the validity of a triple, composed of a head, relation, and tail.

One fundamental aspect that has gained prominence in assessing the validity of triples within KGs is the role of structural features (Wang et al., 2018; Xie et al., 2020; Shen et al., 2020; 2022; Zheng et al., 2023). In particular, structural features of a triple could be attributes derived from the surrounding subgraph of its head, relation, and tail, encompassing information such as their common neighbors or subgraph structure. For a triple, understanding its head, relation, and tail's surrounding subgraph connections in the KG can provide crucial insights into its reliability. However, we encounter three challenges when it comes to modeling the structural features of a triple within a KG. Firstly, we must pinpoint which specific structural features is crucial for validating a triple. Secondly, once we identify an effective method to quantify this information as features, how can we efficiently compute it for all candidate triples in the KG? Lastly, we must determine how to harness this triple's computed structural features in the context of link prediction. Conquering these challenges is imperative for advancing the frontiers of KG exploration.

This study recognizes the pivotal significance of the intersection among the k-hop neighborhoods of the head, relation, and tail when evaluating the validity of a triple. If we conceptualize both entities and relations as vertices within a KG, we can discern that triples exhibiting substantial overlap — a.k.a *intersections* — in their respective k-hop neighborhoods tend to signify facts. This observation is supported by statistical evidence demonstrating that valid triples consistently exhibit greater intersections than their non-valid counterparts. However, translating this insight into computational practice presents significant challenges. KGs often comprise millions, or even more, triples, rendering the computation of k-hop neighborhoods' intersection for each triple and various values of k prohibitively expensive. This trade-off between effectiveness and computational efficiency hinders progress in modeling the structural information inherent in triples.

To tackle this crucial aspect of KG, we present an innovative randomized algorithm aimed at efficiently generating intersection features for candidate triples. Our approach dissects the intersection size of k-hop neighborhoods associated with each triple's vertices into a product of 3-way Jaccard similarity and 3-way union cardinality. When considering three k-hop neighborhoods within a triple, the 3-way Jaccard similarity quantifies the percentage of their shared elements, while the 3-way union cardinality measures the total elements across three neighborhoods. Subsequently, we introduce two randomized estimations for 3-way Jaccard similarity and 3-way union cardinality, independently. These estimations are then multiplied to derive our estimation of the intersection size. Furthermore, we've devised transformations to calculate the intersection size of 1 to k-hop neighborhoods through an inductive process. Ultimately, these multi-hop intersection size estimations are consolidated into intersection features and used to train a straightforward, fully connected neural network for link prediction in the KG.

We formally summarize our contributions as follows.

- We recognize the significance of the intersection between the k-hop neighborhoods of the head, relation, and tail as crucial for discerning a valid triple. Moreover, we integrate the intersection's cardinality into our deep learning methodologies as a vital input feature.
- We present a novel randomized algorithm designed to estimate the cardinality of the intersection of the k-hop neighborhoods for entities and the relation within a triple. Using this algorithm, we efficiently extend the computation of intersection features to encompass vast KGs comprising millions of triples.
- We employ the intersection features and develop a deep neural network for predicting links within knowledge graphs. Through a comprehensive series of experiments, our results demonstrate that our simple yet effective fully-connected network surpasses KG embedding methods and even outperforms graph convolutional network baselines in terms of predictive accuracy and training efficiency.

2. Related Works

2.1. Representation Learning for Knowledge Graphs

Representation Learning for KGs has gained significant attention in recent years. It mainly focuses on embedding entities and relations from a KG into dense vectors. Building upon the pioneering work of TransE (Bordes et al., 2013), which learns vector representations of entities while treating relations as translation functions within a continuous vector space, there are subsequent advancements, such as TransH (Wang et al., 2014) and TransR (Lin et al., 2015).

However, these methods frequently exhibit lower effectiveness compared to GNN-based models when applied to large and heterogeneous knowledge graphs featuring diverse set of entities and relations. GNN-based models such as KBGAT (Nathani et al., 2019) or CompGCN (Vashishth et al., 2020) exhibit high expressiveness and possess the capability to learn intricate, non-linear relationships among entities within a knowledge graph. Furthermore, they are good at capturing rich representations of an entity's connections through the aggregation of information from neighboring entities and relations.

While GNN-based models (Nathani et al., 2019) (Zhou et al., 2020b; 2021; 2020a; 2023) (Vashishth et al., 2020) have demonstrated their effectiveness in knowledge graph representation learning, they demand a substantial allocation of computational resources and entail lengthy training periods. Furthermore, because of their intricate architecture, GNN-based models typically necessitate extensive datasets and numerous epochs to capture the knowledge graph representation effectively. This limitation hinders the feasibility of numerous potential applications that emphasize the efficient training times and scalability of the model.

2.2. Randomized Algorithms for Scalable Graph Learning

Learning on large graphs presents notable challenges related to scalability and efficiency (Duan et al., 2022; Chen et al., 2022). This complexity arises from the utilization of graph-based sparse matrix operations during the aggregation phase, characterized by numerous random memory accesses and limited data reuse. These characteristics pose an obstacle in acceleration using community hardware such as CPUs and GPUs, as discussed in (Liu et al., 2023b). Prior research has expanded the application of randomized algorithms within the field of graph learning, aiming to enhance their scalability (Liu et al., 2021). These algorithms trade off computational precision in favor of reduced memory usage and processing time. Specifically, ELPH (Chamberlain et al., 2022) proposes to encode the subgraph relationship into pairwise node features with locality-sensitive hashing. RSC (Liu et al., 2023a) accelerates the graph-based sparse operations with randomized matrix multiplication, where it sub-samples the graph adjacency matrix into the subspace where the computation can be done much faster. Another direct way to reduce the time complexity is to remove unimportant edges. Specifically, the effective resistance serves as an edge importance score, providing a theoretical guarantee for eliminating unimportant edges. However, it is very expensive to estimate in practice as it requires calculating the inverse of the giant graph Laplacian matrix. DSpar (Liu et al., 2023b) approximates the effective resistance (Spielman & Srivastava, 2008) using only the node degree information. Then it sparsifies the graph once before training with degree-based random sampling. Sketch-GNN proposes to sketch the graph adjacency matrix. It learns and updates these sketches in an online fashion using learnable LSH. In this way, it achieves sub-linear memory and time complexity with respect to the input graph size (Ding et al., 2022). However, we encounter challenges when endeavoring to expand the scope of triple learning within KGs. A triple, comprising three elements, possesses a notably intricate structure. Modeling a 3-way relation effectively with randomized algorithms remains to be explored in this work.

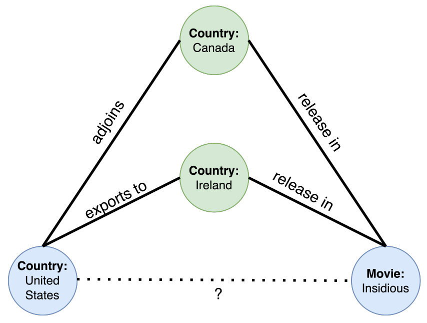
3. Motivation

While there are many ways to solve the knowledge graph learning tasks, many methods would rely on a large feature space with complex learning models, resulting in various complications or even limitations as mentioned in Section 2.

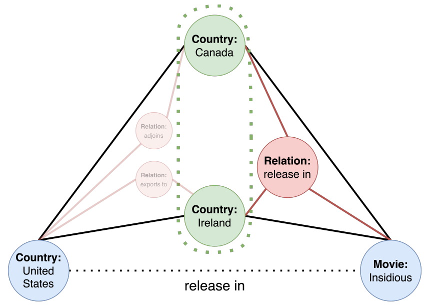
In our empirical study, we observe the following interesting pattern during the completion of r_p for a triple (h, r_p, t) (given head and tail entities as h and t): Suppose we transform a relation $r_{candidate}$ into a vertex (as shown in Figure 1a $\rightarrow$ Figure 1b) and observe that there is significant overlap among the neighbors of h, $r_{candidate}$, and t vertices, then it is advisable to select $r_p = r_{candidate}$.

To illustrate this pattern, we take a snapshot sampled in dataset FB15K237 (Toutanova & Chen, 2015) shown in Figure 1a, where we would like to determine the relation r_p for triple (h = *U.S.*, r_p, t = *Insidious*). We first conceptualize relations (edges) in the snapshot as vertices within the KG, so that Figure 1a is therefore transformed to Figure 1b. Then, we store the neighborhood information of h, t, as well as the three newly formed relation vertices — adjoins, exports to, and release in — in Figure 1b as sets, which are:

- Neighbors of adjoins: $\mathcal{N}$(*adjoins*) = {*U.S.*, *Canada*}.
- Neighbors of exports to: $\mathcal{N}$(*exports to*) = {*U.S.*, *Ireland*}.
- Neighbors of release in: $\mathcal{N}$(*release in*) = {*Insidious*, *Canada*, *Ireland*}.
- Neighbors of *U.S.*: $\mathcal{N}$(*U.S.*) = {*Canada*, *Ireland*, adjoins, exports to}.



(a) Original KG



(b) Conceptualization of relation as vertex

Figure 1. Simplified process of solving (h = *U.S.*, r_p = ?, t = *Insidious*). To quantify the intersection features of the asked triples, we conceptualize the representation of both entities and relations as vertices within this KG. Thus, Figure 1a is transformed into Figure 1b, where the relation vertex with the largest neighborhood intersection with neighbors of h and neighbors of t is the most likely candidate for r_p. In this case, this would be release in, as the neighbors of release in vertex in Figure 1b have an intersection of {*Canada*, *Ireland*} (cardinality = 2) with the neighbors of *U.S.* and the neighbors of *Insidious*. Which is larger than the neighborhood intersection among adjoins, *U.S.*, *Insidious* ({*Canada*}, cardinality = 1) or among exports to, *U.S.*, *Insidious* ({*Ireland*}, cardinality = 1). Please refer to Section 3 for a more detailed walk-through.

- Neighbors of *Insidious*: $\mathcal{N}$(*Insidious*) = {*Canada*, *Ireland*, release in}.

After collecting such sets, we calculate the intersections among h neighbor set, t neighbor set, and a particular relation vertex's neighbor set as the following:

- $\mathcal{N}$(*U.S.*) $\cap$ $\mathcal{N}$(*Insidious*) $\cap$ $\mathcal{N}$(*adjoins*) = {*Canada*}.
- $\mathcal{N}$(*U.S.*) $\cap$ $\mathcal{N}$(*Insidious*) $\cap$ $\mathcal{N}$(*exports to*) = {*Ireland*}.
- $\mathcal{N}$(*U.S.*) $\cap$ $\mathcal{N}$(*Insidious*) $\cap$ $\mathcal{N}$(*release in*) = {*Canada*, *Ireland*}.

With such information, we let the relation with the highest intersection cardinality to be the most likely candidate for r_p[1]. In this case, such relation is release in with a cardinality

[1]We like to note that selecting the relation with the highest intersection cardinality is a simplified version of our solution for the ease of demonstration. In practice, such intersection features are fed into a fully-connected network to determine the most likely

Table 1. The average 1-hop intersection cardinality of valid and invalid triples

Dataset	Average Cardinality of 1-hop Triple Intersection		
	Valid	w/ Invalid Heads	w/ Invalid Tails
YAGO3-10	**1.20**	0.60	0.06
NELL-995	**2.35**	0.30	0.24
WN18RR	**0.26**	0.04	0.003
FB15K237	**8.75**	1.65	0.65
FB15K	**8.46**	2.09	1.30

of 2, so we shall have (*U.S.*, release in, *Insidious*); which happen to be a valid prediction.

Here, we denote the number of vertices (as entities and relations) that lie in the intersection among $\mathcal{N}(h)$, $\mathcal{N}(t)$, and a certain $\mathcal{N}(r_{candidate})$ as ***intersection features*** or *cardinality of intersection* for $(h, r_{candidate}, t)$. In this work, we argue that **knowledge graphs can be learned by *only* utilizing such kinds of intersection features**.

Note that the above example is not a cherry-picked one, but rather a global phenomenon that exists across multiple knowledge graph datasets. Here, we demonstrate the ubiquitousness of such phenomenon via two different channels: 1) we count the cardinality of intersections in regard to valid and invalid triples, where valid triples tend to have a lot more intersections among the neighborhood of its h, t, and r-transformed vertices; and 2) we showcase learning upon the exact measurements of intersection features may yield dominating performance on KG link prediction task against various popular alternative approaches.

Specifically for our first setup, we randomly sample 10,000 valid triples from the corresponding training set of the five showcased datasets in Table 1. Additionally, we generate a set of 20,000 invalid triples by randomly substituting head and tail entities, ensuring that these generated invalid triples do not overlap with any valid triples in the training set. Our statistical analysis reveals a notable trend: valid triples exhibit a much higher cardinality of neighborhood intersection among their head entities, tail entities, and transformed relation vertices than their invalid counterparts.

Our second phenomenon observing experiment is designed to assess the effectiveness of learning on the exact measurement of intersection features (see Section 4.4) in link prediction tasks. Given the high computational expense associated with calculating the exact cardinality of intersections of triples, we sample a subset $\mathcal{S}$ from the NELL-995 train set (Xiong et al., 2017) by randomly selecting ten valid triples as well as their 1-hop neighborhood entities. We then separate such subset $\mathcal{S}$ into its own train and test set to ensure no leakage. Table 2 showcased a landslide dominance of our methods across four different metrics against multiple SOTA methods. For example, regarding the Mean Reciprocal Rank (MRR) metric, our method, learned with exact intersection features, excels with a score of 0.160. In contrast, several other methods yield only mediocre results, with one even as low as 0.004.

However, one foreseeable and severe challenge is, again, the high computing cost of measuring such intersection features in an exact manner. To mitigate this burden, we proposed a novel randomized algorithm to provide an efficient but faithful estimation of such intersection features, where learning on the estimated features may yield a competitive beyond-SOTA result as learning on the exact features (Table 2).

Table 2. Experimental results of our method learning upon **exactly measured** v.s. **estimated** intersection features v.s. **baselines** in a KG link prediction task on sampled subset of NELL-995 ('$\approx x$' is 'Estimated x', 'EM x' is 'Exact-Measured x').

Method	Sampled NELL			
	MRR	H@1	H@3	H@10
Intersection Features (EM Jaccard, EM Union)	0.142	0.059	0.102	0.370
Intersection Features (EM Jaccard, $\approx$ Union)	0.073	0.012	0.056	0.269
Intersection Features ($\approx$ Jaccard, EM Union)	0.146	0.034	**0.185**	**0.395**
Intersection Features ($\approx$ Jaccard, $\approx$ Union)	**0.160**	**0.102**	0.145	0.318
TransE	0.041	0.000	0.043	0.099
TransH	0.043	0.000	0.049	0.102
TransR	0.048	0.000	0.049	0.142
DistMult	0.004	0.000	0.000	0.003
ComplEx	0.078	0.049	0.090	0.117
RotatE	0.005	0.000	0.009	0.015

4. Method

This section formally introduces our approach to generating K-hop intersection features for KG learning. We start by defining and decomposing the triple intersection into triple Jaccard and triple cardinality. Next, we introduce the algorithms to estimate triple Jaccard and cardinality, respectively. Finally, we introduce our K-hop intersection features and the neural network trained on them.

Notation: We denote a KG as $\mathcal{G}$, which contains a set of relations $\mathcal{P}$ and a set of entities $\mathcal{V}$. Two entities $h, t \in \mathcal{V}$ and one relation $r \in \mathcal{P}$ form a triple (h, r, t) that represents a fact. We view all entities in $\mathcal{V}$ and relations in $\mathcal{P}$ as vertices. For every entity $v \in \mathcal{V}$, if there is a triple (v, r, t), we say vertex v is connected to vertex r and t with edges, respectively. For every relation $r \in \mathcal{P}$, if there is a triple (h, r, t), we say vertex r is connected to vertex h and t with edges, respectively. For every entity $v \in \mathcal{V}$, we denote its k-hop neighborhoods as $\mathcal{N}_k(v)$. $\mathcal{N}_k(v)$ consists of all vertices that can be reached from v by following $\leq k$ edges. We note that

r_p. Please refer to Section 4 for full details.

vertices in $\mathcal{N}_k(v)$ can be both entities and relations. Also, for every relation $r \in \mathcal{P}$, we denote its k-hop neighborhoods as $\mathcal{N}_k(r)$. We denote $|A|$ as the cardinality of a set A.

4.1. Decomposition of Triple Intersection

Following observations in Section 3, we would like to estimate the size of an intersection of the k-hop neighborhoods of a triple (h, r, t). Formally, we define the triple intersection cardinality as follows.

Definition 4.1. Given a triple (h, r, t) in a knowledge graph $\mathcal{G}$, we define its k-hop triple intersection cardinality as $|\mathcal{N}_k(h) \cap \mathcal{N}_k(r) \cap \mathcal{N}_k(t)|$.

In practice, it is challenging to explicitly compute the exact intersection cardinality for all triples in KG. A strawman solution to overcome this computation bottleneck is subsampling. However, this Monte-Carlo style strategy is sample inefficient. It requires a significant amount of samples to perform an even rough estimate.

As a result, we propose an algorithm to estimate the k-hop triple intersection (see Definition 4.1) by the decomposition as below. We show that

$$\begin{aligned}&|\mathcal{N}_k(h) \cap \mathcal{N}_k(r) \cap \mathcal{N}_k(t)| \\ =&\mathcal{J}(\mathcal{N}_k(h), \mathcal{N}_k(r), \mathcal{N}_k(t)) \cdot \mathcal{C}(\mathcal{N}_k(h), \mathcal{N}_k(r), \mathcal{N}_k(t)),\end{aligned} \tag{1}$$

where $\mathcal{J}(\mathcal{N}_k(h), \mathcal{N}_k(r), \mathcal{N}_k(t))$ is the 3-way Jaccard similarity of k-hop neighborhoods of (h, r, t) and $\mathcal{C}(\mathcal{N}_k(h), \mathcal{N}_k(r), \mathcal{N}_k(t))$ is the cardinality of the union $\mathcal{N}_k(h) \cup \mathcal{N}_k(r) \cup \mathcal{N}_k(t)$. With Eq. (1), we propose to estimate 3-way Jaccard similarity $\mathcal{J}(\mathcal{N}_k(h), \mathcal{N}_k(r), \mathcal{N}_k(t))$ and 3-way union cardinality $|\mathcal{N}_k(h) \cup \mathcal{N}_k(r) \cup \mathcal{N}_k(t)|$ independently. Next, we multiply the two estimates together for k-hop triple intersection estimation.

4.2. 3-Way Jaccard Estimation

We propose an estimator with MinHash (Broder, 1997) functions to estimate the 3-way Jaccard similarity $\mathcal{J}(\mathcal{N}_k(h), \mathcal{N}_k(r), \mathcal{N}_k(t))$. To start with, we formally, introduce the MinHash function.

Definition 4.2. Given a set X, we denote their MinHash value as $\min(\Pi(X))$, where $\Pi : X \to [m]$ is a permutation function that maps every element in X into a value in $[m]$ uniformly at random, m denote the permutation range and min function takes the minimum value as output. Moreover, given two sets A and B, we have

$$\Pr[\min(\Pi(A)) = \min(\Pi(B))] = \frac{|A \cap B|}{|A \cup B|} = \mathcal{J}(A, B).$$

MinHash function is an effective randomized estimator for pairwise Jaccard similarity. However, it requires significant effort to extend it for 3-way Jaccard similarity. Shrivastava and Li used MinHash with a bucketing scheme that estimates the 3-way Jaccard similarity (Shrivastava & Li, 2013). However, this bucketing scheme restricts the permutation range m to a small value and requires computing the $m \cdot \min(\Pi(A))$. Since $\min(\Pi(A))$ is up to m, larger m results in overflow issues in practice. On the other hand, a smaller permutation range m results in an increase in collisions, which affects the quality of estimation.

In this work, we propose a new approach for estimating 3-way Jaccard similarity. We start with defining a pair of asymmetric transforms.

Definition 4.3. Given three sets A, B, and C, we define the function ϕ and ψ as

$$\begin{aligned}\phi(A, B) &= \textsc{Concat}(\mathsf{Bit}(\min(\Pi(A)), \mathsf{Bit}(\min(\Pi(B))) \\ \psi(C) &= \textsc{Concat}(\mathsf{Bit}(\min(\Pi(C)), \mathsf{Bit}(\min(\Pi(C))),\end{aligned}$$

where $\mathsf{Bit}(x)$ denotes the bit-wise representation of integer x, $\textsc{Concat}(b_1, b_2)$ denotes the concatenation of two bit arrays b_1 and b_2.

In Definition 4.2, function Π maps every element in set A into an integer in $[m]$. Let m be a power of 2, $\min(\Pi(A))$ can be represented as a $\log_2 m$ bit array. As a result, $\phi(A, B)$ and $\psi(C)$ produce a $2\log_2 m$ bit array through concatenation. Given the asymmetric transforms in Definition 4.3, we have,

Theorem 4.4. *Let $\min(\Pi(x))$ denote a MinHash function (see Definition 4.2). Let ϕ, ψ denote the asymmetric transform in Definition 4.3, Given three set A, B and C, we show that*

$$\Pr[\phi(A, B) = \psi(C)] = \frac{|A \cap B \cap C|}{|A \cup B \cup C|} = \mathcal{J}(A, B, C).$$

Proof. We start showing that $\phi(A, B) = \psi(C)$ holds if and only if $\min(\Pi(A)) = \min(\Pi(B)) = \min(\Pi(C))$. Given that Π is an independent permutation function, without loss of generality, we show that

$$\begin{aligned}&\Pr[\min(\Pi(A)) = \min(\Pi(B)) = \min(\Pi(C))] \\ =&\frac{|A \cap B \cap C|}{|A \cup B \cup C|} = \mathcal{J}(A, B, C).\end{aligned}$$

As a result, $\Pr[\phi(A, B) = \psi(C)] = \mathcal{J}(A, B, C)$. □

Next, we propose an estimator for $\mathcal{J}(A, B, C)$ as below,

Definition 4.5. Let Φ denote a set that contains n pairs of (ϕ_i, ψ_i) (see Definition 4.3). Each pair $(\phi_i, \psi_i) \in \Phi$ is initialized independently with a permutation function Π_i. Given three sets A B and C, we define an estimator as

$$\hat{\mathcal{J}}(A, B, C) = \frac{1}{n} \cdot \sum_{i=1}^{n} \mathbf{1}\{\phi_i(A, B) = \psi_i(C)\},$$

where $\mathbf{1}$ is an indicator function.

Following Theorem 4.4, we show that $\hat{\mathcal{J}}(A, B, C)$ is an asymptotic unbiased estimator of $\mathcal{J}(A, B, C)$.

$$\mathbb{E}_{n\to\infty}[\hat{\mathcal{J}}(A, B, C)] = \mathcal{J}(A, B, C). \tag{2}$$

We can compute the MinHash value of each vertex h's k-hop neighborhoods as

$$\pi_h^k = \min_{v\in\mathcal{N}_h}(\pi_v^{k-1}). \tag{3}$$

Next, given a triple, we can apply the asymmetric transform (see Definition 4.3) to estimate its 3-way Jaccard similarity. In this way, we do not scan every vertex's k-hop neighborhoods for a triple's 3-way Jaccard similarity. Therefore, we significantly improve the computation efficiency with an acceptable sacrifice in estimation precision.

4.3. 3-Way Union Cardinality Estimation

We introduce HyperLogLog (Flajolet et al., 2007; Heule et al., 2013) data structure to estimate 3-way union cardinality $|\mathcal{N}_k(h) \cup \mathcal{N}_k(r) \cup \mathcal{N}_k(t)|$. The HyperLogLog data structure maintains a sketch of a set X as follows.

Definition 4.6. Let $p \in [31]$ denote a parameter. Let $Y = \vec{0}$ denote a 2^p-dimensional vector. Let X denote a set. Let $h : X \to [2^{32}]$ denote a universal hash function. Let g denote a function that transforms a bit array into an integer. Let $\sigma : \mathbb{N}_+ \to \mathbb{N}_+$ denote a function that takes the number of leading zeros in the bit representation of an integer. For every $x \in X$, we update Y following the rules below.

- $\mathsf{idx} \leftarrow g(\mathsf{Bit}(h(x))[: p])$
- $z \leftarrow g(\mathsf{Bit}(h(x))[p :]) + 1$
- $Y[\mathsf{idx}] \leftarrow \max(Y[\mathsf{idx}], \sigma(z))$

where $\mathsf{Bit}(h(x))[: p]$ takes the first p bits of the bit array and $\mathsf{Bit}(h(x))[p :]$ takes the rest. Finally, we output Y as the HyperLogLog sketch of X.

Given three sets, we are able to estimate their union cardinality based on merging HyperLogLog. We demonstrate it in the following statement.

Lemma 4.7. *Given three sets A, B and C, we first compute their HyperLogLog sketch following Definition 4.6 with parameter p. We denote the HyperLogLog sketch of A, B and C as Y_A, Y_B and Y_C, respectively. Next, we compute*

$$Y = \max(Y_A, Y_B, Y_C),$$

where $\max$ *is an element-wise max operation on the three vectors. Next, we have an estimate of $|A \cup B \cup C|$ below*

$$\hat{\mathcal{C}}(A, B, C) = \alpha 2^{2p}(\sum_{i=0}^{2^{2p}} 2^{-Y_i})^{-1}.$$

In Lemma 4.7, parameter α is determined following (Heule et al., 2013). We note that the estimator $\hat{\mathcal{C}}$ is asymptotically almost unbiased (Flajolet et al., 2007). Moreover, from Lemma 4.7, we know that the HyperLogLog sketch of every set can be combined for estimating their union cardinality. As a result, for each vertex h, we can sketch its k-hop neighborhoods set with HyperLogLog as

$$Y_h^k = \max_{v\in\mathcal{N}_h}(Y_v^{k-1}) \tag{4}$$

Next, given a triple, we use Lemma 4.7 to estimate its union cardinality without scanning each vertex's k-hop neighborhoods.

4.4. k-Hop Intersection Features

We continue to formally introduce 2 types of intersection features of triple (h, r, t) that are used for link prediction tasks as follows:

1. $\hat{\mathcal{I}}_{k_h,k_r,k_t}$: the estimated cardinality of common neighborhoods that are exactly k_h-hop from vertex h, k_r-hop from vertex r, and k_t-hop from vertex t.
2. $\hat{\mathcal{T}}_{k_v}$: the estimated cardinality of nodes that are exactly k_v hop from vertex v where v is in $S = \{h, r, t\}$, and $> k_v$ hop from vertices among $S \setminus \{v\}$.

Following Definition 4.5 and Lemma 4.7, we propose an estimator of k_h, k_r, and k_t-hop triple intersection cardinality, i.e., the number of distinct nodes at distance $\le k_h$ from h, $\le k_r$ from r, and $\le k_t$ from t, as follow:

$$\begin{aligned}\hat{E}_{k_h,k_l r k_t} =& \hat{\mathcal{J}}(\mathcal{N}_{k_h}(h), \mathcal{N}_{k_r}(r), \mathcal{N}_{k_t}(t))\cdot \\ & \hat{\mathcal{C}}(\mathcal{N}_{k_h}(h), \mathcal{N}_{k_r}(r), \mathcal{N}_{k_t}(t)).\end{aligned} \tag{5}$$

Next, we start computing $\hat{\mathcal{I}}_{k_h,k_r,k_t}$. To start with, we explicitly show that $\hat{\mathcal{I}}_{1,1,1} = \hat{E}_{1,1,1}$, because the 1-hop neighborhood is the lowest level of neighborhoods. Next, given k-hop intersection cardinality estimation, we can compute $\hat{\mathcal{I}}_{k_h,k_r,k_t}$ via induction as follow,

$$\begin{aligned}\hat{\mathcal{I}}_{k_h,k_r,k_t} =& \hat{E}_{k_h,k_r,k_t} \\ & - \sum_{\substack{x\le k_h \ y\le k_r \ z\le k_t \\ (x,y,z)\neq(k_h,k_r,k_t)}} \hat{\mathcal{I}}_{x,y,z}.\end{aligned} \tag{6}$$

Next, we define $\hat{\mathcal{T}}_{k_v}$ as follow,

$$\begin{aligned}\hat{\mathcal{T}}_{k_h} &= |\mathcal{N}_{k_h}| - \hat{\mathcal{T}}_{k_h-1} - \sum_{i=1}^{k_h}\sum_{j=1}^{k}\sum_{z=1}^{k}\hat{\mathcal{I}}_{i,j,z},\\ \hat{\mathcal{T}}_{k_r} &= |\mathcal{N}_{k_r}| - \hat{\mathcal{T}}_{k_r-1} - \sum_{i=1}^{k}\sum_{j=1}^{k_r}\sum_{z=1}^{k}\hat{\mathcal{I}}_{i,j,z}, \qquad (7)\\ \hat{\mathcal{T}}_{k_t} &= |\mathcal{N}_{k_t}| - \hat{\mathcal{T}}_{k_t-1} - \sum_{i=1}^{k}\sum_{j=1}^{k}\sum_{z=1}^{k_t}\hat{\mathcal{I}}_{i,j,z}.\end{aligned}$$

where k is the parameter defining the maximum hop of intersection features. There would be k^3 of $\hat{\mathcal{I}}_{k_h,k_r,k_t}$ features and $3k$ of $\hat{\mathcal{T}}_{k_v}$ where vertex $v \in S = \{h, r, t\}$, in total of $k^3 + 3k$ intersection features per each triple (h, r, t).

4.5. Learning on Intersection Features

We use the intersection features extracted in Section 4.4 to train a deep neural network (DNN) for link prediction in KG exploration. Given a triple, we regard its $k^3 + 3k$ intersection features as a vector. Next, we apply a single fully connected layer to the representation features. Subsequently, these features are normalized and go through the final activation function, e.g. Sigmoid σ, to yield the triple's predicted validity. We adhere to the adopted practice and follow TransE's (Bordes et al., 2013) convention by utilizing a margin-based loss as the architecture's primary loss function. This approach penalizes invalid triples, pushing for higher predicted validity, while minimizing the predicted values of valid triples. We note that the introduced DNN is simple yet effective. We do not introduce expensive operations such as graph convolution in the proposed DNN, resulting in a training efficiency improvement. Moreover, we incorporate multi-hop intersection features in training DNN, which leads to potential improvements over KG embedding approaches such as TransE (Bordes et al., 2013).

5. Experiment

In this section, we evaluate the performance of Intersection Features method and compare it against recent advanced link prediction methods on various LP benchmarks. Firstly, we introduce the benchmark datasets used in the experiments. Then, we provide an overview of baseline models and experimental settings. Next, We present our evaluation results. We also present efficiency profiling and ablation study in the supplementary materials.

5.1. Datasets

We first introduce knowledge graph completion datasets used in the experiments. All datasets are publicly available and widely used. Their statistics are shown in Table 4 in the supplementary materials. **NELL-995** (Xiong et al., 2017) contains triples derived from the NELL system to benchmark link prediction for multi-hop entity pairs. **WN18RR** (Dettmers et al., 2018) is a link prediction dataset derived from WordNet, a large knowledge graph of semantic relations between words. **YAGO3-10** (Mahdisoltani et al., 2015) is the largest knowledge graph completion dataset used in our experiments with more than one million triples taken from Wikipedia. We also include **FB15K237** (Toutanova & Chen, 2015) and **FB15K** (Bordes et al., 2013) are small subsets of knowledge base relation triples in Freebase, a heterogeneous and well-known knowledge graph. We present the experiment results of FB15K237 and FB15K in the supplementary material.

5.2. Experimental Settings

We evaluated Intersection Features on five introduced knowledge graph completion benchmarks and compared it against the performance of recent link prediction models. We used sixteen well-known or recent models as baselines, including TransE (Bordes et al., 2013), TransH (Wang et al., 2014), TransR (Lin et al., 2015), DistMult (Yang et al., 2015), ComplEx (Trouillon et al., 2016) with N3 regularizer proposed in (Lacroix et al., 2018), and RotatE (Sun et al., 2019), KBGAT (Nathani et al., 2019), CompGCN (Vashishth et al., 2020), SE-GNN (Li et al., 2022a), KRACL (Tan et al., 2023), AdaProp (Zhang et al., 2022b), MEIM (Tran & Takashu, 2022), HousE (Li et al., 2022b), NCRL (Cheng et al., 2023), and A*Net (Zhu et al., 2023). The performance metrics for each baseline are extracted either directly from their original paper, obtained as the best reproduced result reported in (Yang et al., 2021) (Tran & Takashu, 2022) (Das et al., 2018) (Zhang et al., 2022a) (Dettmers et al., 2018) (Li et al., 2022a), or directly reproduced by us with the best reported hyperparameters.

Evaluations We train the models on the train graph. Next, given the train graph and a corrupted triplet from the test graph $(h, r, ?)$, we would like to predict an $?$ from entities within the set of KG's entities. Note that train and test graphs are disjoint in terms of triples, but entities from test graphs can also be present in their respective train graphs. We follow widely adopted evaluation metrics on link prediction task (Bordes et al., 2013). For each positive triple (h, r, t) in the test set, where h & t represent head and tail entities and r is the relation, we corrupt it by replacing the head and tail entity with every other entity in the dataset to obtain invalid triples (h', r, t) and (h, r, t'), respectively. Each method attempts to rank each triple on how likely it is a valid one in the first place. We measure filtered *MRR*, mean reciprocal rank, and filtered *Hit@k*, how many positive triples are correctly ranked in the top **k**, given corrupted but positive triples are not considered. The higher the metrics' values are, the better the method performs.

Table 3. Experimental results of link prediction on large-scale, heterogeneous knowledge graph datasets

Method	Source	YAGO3-10				NELL-995				WN18RR			
		MRR	H@1	H@3	H@10	MRR	H@1	H@3	H@10	MRR	H@1	H@3	H@10
Intersection Features (Ours)		**0.617**	**0.590**	**0.665**	0.674	**0.789**	**0.776**	**0.791**	0.813	**0.781**	**0.724**	**0.891**	**0.899**
	Self-Replicated	0.275	0.035	0.470	0.640	0.326	0.225	0.399	0.488	0.193	0.005	0.358	0.470
	(Yang et al., 2021)	0.510	0.413	0.574	0.681	–	–	–	–	0.222	0.014	0.399	0.528
TransE	(Tran & Takashu, 2022)	0.501	0.406	–	0.674	–	–	–	–	0.222	0.031	–	0.524
(Bordes et al., 2013)	(Zhang et al., 2022a)	–	–	–	–	0.456	0.514	0.678	0.751	0.359	0.289	0.464	0.534
	(Li et al., 2022a)	–	–	–	–	–	–	–	–	0.223	0.014	0.401	0.529
	(Han et al., 2023)	–	–	–	–	–	–	–	–	0.466	0.423	–	0.556
TransH (Wang et al., 2014)	Self-Replicated	0.352	0.233	0.412	0.576	0.333	0.239	0.399	0.483	0.204	0.009	0.378	0.471
TransR (Lin et al., 2015)	Self-Replicated	0.354	0.234	0.421	0.581	0.322	0.218	0.399	0.487	0.202	0.007	0.379	0.471
	(Tran & Takashu, 2022)	0.501	0.413	–	0.661	–	–	–	–	0.465	0.432	–	0.532
DistMult	(Yang et al., 2021)	0.566	0.491	0.608	0.704	–	–	–	–	0.455	0.410	0.467	0.544
(Yang et al., 2015)	(Das et al., 2018)	–	–	–	–	0.680	0.610	0.733	0.795	0.433	0.410	0.441	0.475
	(Li et al., 2022a)	–	–	–	–	–	–	–	–	0.439	0.394	0.452	0.533
	Self-Replicated	0.578	0.501	0.624	0.714	0.401	0.345	0.448	0.495	0.471	0.430	0.489	0.553
ComplEx-N3	(Yang et al., 2021)	0.577	0.502	0.621	0.709	–	–	–	–	0.489	0.443	0.502	0.580
(Lacroix et al., 2018)	(Tran & Takashu, 2022)	0.569	0.498	0.609	0.701	–	–	–	–	0.480	0.435	0.495	0.572
	(Lacroix et al., 2018)	0.360	–	–	0.550	–	–	–	–	0.440	–	–	0.510
ComplEx	Self-Replicated	0.410	0.323	0.449	0.575	0.237	0.197	0.252	0.313	0.471	0.430	0.489	0.553
(Trouillon et al., 2016)	(Dettmers et al., 2018)	0.360	0.260	0.400	0.550	–	–	–	–	0.440	0.410	0.460	0.510
	(Das et al., 2018)	–	–	–	–	0.694	0.612	0.761	**0.827**	0.440	0.410	0.460	0.510
	Self-Replicated	0.454	0.364	0.505	0.626	0.356	0.298	0.395	0.444	0.394	0.383	0.397	0.416
RotatE	(Sun et al., 2019)	–	–	–	–	–	–	–	–	0.476	0.428	0.492	0.571
(Sun et al., 2019)	(Yang et al., 2021)	0.495	0.402	0.550	0.670	–	–	–	–	0.476	0.428	0.492	0.571
	(Zhang & Yao, 2022)	–	–	–	–	0.513	0.413	–	0.637	0.448	0.413	–	0.513
KBGAT	Self-Replicated	0.528	0.434	0.580	0.703	0.550	0.461	0.618	0.697	0.428	0.344	0.486	0.570
(Nathani et al., 2019)	(Nathani et al., 2019)	–	–	–	–	0.530	0.447	0.564	0.695	0.440	0.361	0.483	0.581
CompGCN	Self-Replicated	0.267	0.179	0.304	0.439	0.412	0.355	0.444	0.503	0.187	0.053	0.242	0.496
(Vashishth et al., 2020)	(Vashishth et al., 2020)	–	–	–	–	–	–	–	–	0.479	0.443	0.494	0.546
SE-GNN	Self-Replicated	0.238	0.164	0.254	0.384	0.214	0.164	0.242	0.300	0.485	0.440	0.503	0.571
(Li et al., 2022a)	(Li et al., 2022a)	–	–	–	–	–	–	–	–	0.484	0.446	0.509	0.572
KRACL	Self-Replicated	0.305	0.219	0.335	0.474	0.445	0.380	0.488	0.559	0.373	0.297	0.421	0.501
(Tan et al., 2023)	(Tan et al., 2023)	–	–	–	–	0.563	0.495	0.602	0.672	0.527	0.482	0.547	0.613
AdaProp	(Zhang et al., 2022b)	0.573	0.510	–	0.685	0.554	0.493	–	0.655	0.562	0.331	–	0.585
MEIM	(Tran & Takashu, 2022)	0.585	0.514	0.625	**0.716**	–	–	–	–	0.499	0.458	0.518	0.577
HousE	(Li et al., 2022b)	0.571	0.491	0.620	0.714	–	–	–	–	0.511	0.465	0.528	0.602
NCRL	(Cheng et al., 2023)	0.38	0.274	–	0.536	–	–	–	–	0.67	0.563	–	0.850
A*Net	(Zhu et al., 2023)	–	–	–	–	–	–	–	–	0.549	0.495	0.573	0.659

Implementation We employ OpenKE (Han et al., 2018) as the primary framework for training embedding-based models, and we also integrated publicly available source code for GNN-based models into OpenKE. We ensured that the best hyperparameter configurations that are publicly available for the models were utilized during the training process. Particularly for our method, the intersection features are within $[1, 2, 3]$-hop of each node, learning rate λ is chosen among $[1, 0.1, 0.05, 0.01, 0.005, 0.001]$, optimizer is chosen from $\{$Adam, Adagrad, Adadelta, SGD$\}$. Our model is trained at most 4,000 epochs among all datasets. There are 128 MinHash functions used to estimate 3-way Jaccard similarity of a triple. We note that the more MinHash functions we have, the better in estimation. However, due to the robustness of DNNs, we do not require significantly better estimation. Similarly, we set HyperLogLog's size, i.e. p, to be 8. All model training and evaluations were conducted on a single NVIDIA A100 GPU with 80G memory.

5.3. Effectiveness of Intersection Features

We present the results on large-scale datasets in Table 3. Our Intersection Features method consistently outperforms both the embedding-based and GNN-based baselines on large-scale, heterogeneous KGs, exhibiting a considerable lead over the second-best result in each dataset's metrics. For instance, in terms of the *MRR* metric for WN18RR, our method showcases a significant difference of 0.111 when compared to the runner-up.

Results on datasets with more homogeneous and smaller-scale KGs are displayed in Table 5 in the supplementary materials. Our method continues to exhibit better performance compared to embedding-based models across the majority of the benchmark datasets. Furthermore, we outperform GNN-based models in some metrics. In five specific metrics, our results are on par with the best-performing GNN-based methods, with only marginal differences.

6. Conclusion

In conclusion, this study has underscored the significance of considering the intersection of the k-hop neighborhoods of the head, relation, and tail when evaluating the validity of triples within a Knowledge Graph (KG). To tackle this challenge, we introduced a novel randomized algorithm aimed at efficiently generating intersection features for candidate triples. Our experimental findings have not only demonstrated the effectiveness of a simple fully-connected network utilizing these intersection features but have also showcased its superior performance compared to established KG embedding models and graph neural network baselines. Furthermore, our approach has achieved substantial gains in training time efficiency, reinforcing the value of incorporating structural features into the learning process for KGs. These results emphasize the potential of our method in advancing KG exploration and reasoning, offering new avenues for addressing the inherent incompleteness of KGs and uncovering hidden relationships and facts.

Acknowledgement

This research was supported in part by NSF awards 2112606 and 2117439. This work made use of the High Performance Computing Resource in the Core Facility for Advanced Research Computing at Case Western Reserve University. We give our special thanks to the CWRU HPC team for their timely and professional help and maintenance. Additionally, Zhaozhuo Xu is supported by the startup fund of Stevens Institute of Technology.

Impact Statement

We introduce a machine learning method for KG applications. We focus on reducing the computation cost of KG modeling to reduce the carbon release of deploying KG-based services.

References

Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., and Yakhnenko, O. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26, 2013.

Broder, A. Z. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*, pp. 21–29. IEEE, 1997.

Chamberlain, B. P., Shirobokov, S., Rossi, E., Frasca, F., Markovich, T., Hammerla, N. Y., Bronstein, M. M., and Hansmire, M. Graph neural networks for link prediction with subgraph sketching. In *The Eleventh International Conference on Learning Representations*, 2022.

Chen, H., Li, X., Zhou, K., Hu, X., Yeh, C.-C. M., Zheng, Y., and Yang, H. Tinykg: Memory-efficient training framework for knowledge graph neural recommender systems. In *Proceedings of the 16th ACM Conference on Recommender Systems*, pp. 257–267, 2022.

Cheng, K., Ahmed, N., and Sun, Y. Neural compositional rule learning for knowledge graph reasoning. In *International Conference on Learning Representations*, 2023.

Das, R., Dhuliawala, S., Zaheer, M., Vilnis, L., Durugkar, I., Krishnamurthy, A., Smola, A., and McCallum, A. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. In *International Conference on Learning Representations*, 2018.

Destandau, M. and Fekete, J.-D. The missing path: Analysing incompleteness in knowledge graphs. *Information Visualization*, 20(1):66–82, 2021.

Dettmers, T., Pasquale, M., Pontus, S., and Riedel, S. Convolutional 2d knowledge graph embeddings. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Ding, M., Rabbani, T., An, B., Wang, E., and Huang, F. Sketch-gnn: Scalable graph neural networks with sublinear training complexity. *Advances in Neural Information Processing Systems*, 35:2930–2943, 2022.

Duan, K., Liu, Z., Wang, P., Zheng, W., Zhou, K., Chen, T., Hu, X., and Wang, Z. A comprehensive study on large-scale graph training: Benchmarking and rethinking. In *Advances in Neural Information Processing Systems*, 2022.

Flajolet, P., Fusy, É., Gandouet, O., and Meunier, F. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. *Discrete mathematics & theoretical computer science*, 2007.

Han, C., He, Q., Yu, C., Du, X., Tong, H., and Ji, H. Logical entity representation in knowledge-graphs for differentiable rule learning. In *International Conference on Learning Representations*, 2023.

Han, X., Cao, S., Xin, L., Lin, Y., Liu, Z., Sun, M., and Li, J. Openke: An open toolkit for knowledge embedding. In *EMNLP*, 2018.

Heule, S., Nunkesser, M., and Hall, A. Hyperloglog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm. In *Proceedings of the 16th International Conference on Extending Database Technology*, pp. 683–692, 2013.

Lacroix, T., Usunier, N., and Obozinski, G. Canonical tensor decomposition for knowledge base completion. In *International Conference on Machine Learning*, 2018.

Li, M., Wang, Y., Zhang, D., Jia, Y., and Cheng, X. Link prediction in knowledge graphs: A hierarchy-constrained approach. *IEEE Transactions on Big Data*, 8(3):630–643, 2018.

Li, R., Cao, Y., Zhu, Q., Bi, G., Fang, F., Liu, Y., and Li, Q. How does knowledge graph embedding extrapolate to unseen data: a semantic evidence view. In *Thirty-Fifth AAAI Conference on Artificial Intelligence*, 2022a.

Li, R., Zhao, J., Li, C., He, D., Wang, Y., Liu, Y., Sun, H., Wang, S., Deng, W., Shen, Y., Xie, X., and Zhang, Q. House: Knowledge graph embedding with householder parameterization. In *International Conference on Machine Learning*, 2022b.

Lin, Y., Liu, Z., Sun, M., Liu, Y., and Zhu, X. Learning entity and relation embeddings for knowledge graph completion. *AAAI Conference on Artificial Intelligence*, 29, 2015.

Liu, Z., Zhou, K., Yang, F., Li, L., Chen, R., and Hu, X. Exact: Scalable graph neural networks training via extreme activation compression. In *International Conference on Learning Representations*, 2021.

Liu, Z., Shengyuan, C., Zhou, K., Zha, D., Huang, X., and Hu, X. Rsc: Accelerate graph neural networks training via randomized sparse computations. In *International Conference on Machine Learning*, pp. 21951–21968. PMLR, 2023a.

Liu, Z., Zhou, K., Jiang, Z., Li, L., Chen, R., Choi, S.-H., and Hu, X. Dspar: An embarrassingly simple strategy for efficient gnn training and inference via degree-based sparsification. *Transactions on Machine Learning Research*, 2023b.

Mahdisoltani, F., Biega, J., and Suchanek, F. M. Yago3: A knowledge base from multilingual wikipedias. In *Proceedings of the Conference on Innovative Data Systems Research*, 2015.

Nathani, D., Chauhan, J., Sharma, C., and Kaul, M. Learning attention-based embeddings for relation prediction in knowledge graphs. In *The 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2019.

Rossi, A., Barbosa, D., Firmani, D., Matinata, A., and Merialdo, P. Knowledge graph embedding for link prediction: A comparative analysis. *ACM Transactions on Knowledge Discovery from Data*, 15(2):1–49, 2021.

Shen, T., Zhang, F., and Cheng, J. A comprehensive overview of knowledge graph completion. *Knowledge-Based Systems*, pp. 109597, 2022.

Shen, Y., Ding, N., Zheng, H.-T., Li, Y., and Yang, M. Modeling relation paths for knowledge graph completion. *IEEE Transactions on Knowledge and Data Engineering*, 33(11):3607–3617, 2020.

Shi, B. and Weninger, T. Open-world knowledge graph completion. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Shrivastava, A. and Li, P. Beyond pairwise: Provably fast algorithms for approximate k-way similarity search. *Advances in neural information processing systems*, 26, 2013.

Spielman, D. A. and Srivastava, N. Graph sparsification by effective resistances. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pp. 563–568, 2008.

Sun, Z., Deng, Z.-H., Nie, J.-Y., and Tang, J. Rotate: Knowledge graph embedding by relational rotation in complex space. *The Seventh International Conference on Learning Representations*, 2019.

Tan, Z., Chen, Z., Feng, S., Zhang, Q., Zheng, Q., Li, J., and Luo, M. Kracl: Contrastive learning with graph context modeling for sparse knowledge graph completion. In *The Web Conference*, 2023.

Teru, K. K., Denis, E. G., and William, H. L. Inductive relation prediction by subgraph reasoning. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

Toutanova, K. and Chen, D. Observed versus latent features for knowledge base and text inference. In *The 3rd Workshop on Continuous Vector Space Models and their Compositionality*. Association for Computational Linguistics, 2015.

Tran, H.-N. and Takashu, A. Meim: Multi-partition embedding interaction beyond block term format for efficient and expressive link prediction. In *International Joint Conference on Artificial Intelligence*, 2022.

Trouillon, T., Welbl, J., Riedel, S., Gaussier, E., and Bouchard, G. Complex embeddings for simple link prediction. *International Conference on Machine Learning*, 2016.

Vashishth, S., Sanyal, S., Nitin, V., and Talukdar, P. Composition-based multi-relational graph convolutional networks. In *The Eighth International Conference on Learning Representations*, 2020.

Wang, Q., Mao, Z., Wang, B., and Guo, L. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743, 2017.

Wang, Z., Zhang, J., Feng, J., and Chen, Z. Knowledge graph embedding by translating on hyperplanes. *AAAI Conference on Artificial Intelligence*, 28, 2014.

Wang, Z., Lv, Q., Lan, X., and Zhang, Y. Cross-lingual knowledge graph alignment via graph convolutional networks. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 349–357, 2018.

Wang, Z., Xu, Z., Wu, X., Shrivastava, A., and Ng, T. E. Dragonn: Distributed randomized approximate gradients of neural networks. In *International Conference on Machine Learning*, pp. 23274–23291. PMLR, 2022.

Wang, Z., Jia, Z., Zheng, S., Zhang, Z., Fu, X., Ng, T. E., and Wang, Y. Gemini: Fast failure recovery in distributed training with in-memory checkpoints. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 364–381, 2023a.

Wang, Z., Lin, H., Zhu, Y., and Ng, T. E. Hi-speed dnn training with espresso: Unleashing the full potential of gradient compression with near-optimal usage strategies. In *Proceedings of the Eighteenth European Conference on Computer Systems*, pp. 867–882, 2023b.

Wang, Z., Wu, X., Xu, Z., and Ng, T. Cupcake: A compression scheduler for scalable communication-efficient distributed training. *Proceedings of Machine Learning and Systems*, 5, 2023c.

Wang, Z., Xu, Z., Shrivastava, A., and Ng, T. Zen: Near-optimal sparse tensor synchronization for distributed dnn training. *arXiv preprint arXiv:2309.13254*, 2023d.

Xie, Z., Zhou, G., Liu, J., and Huang, X. Reinceptione: relation-aware inception network with joint local-global structural information for knowledge graph embedding. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pp. 5929–5939, 2020.

Xiong, W., Hoang, T., and Wang, W. Y. Deeppath: A reinforcement learning method for knowledge graph reasoning. In *Conference on Empirical Methods in Natural Language Processing*, 2017.

Yang, B., Yih, W.-t., He, X., Gao, J., and Deng, L. Embedding entities and relations for learning and inference in knowledge bases. *International Conference on Learning Representations*, 2015.

Yang, J., Shi, Y., Tong, X., Wang, R., Chen, T., and Ying, X. Improving knowledge graph embedding using affine transformations of entities corresponding to each relation. In *EMNLP*, 2021.

Zhang, D., Li, M., Jia, Y., Wang, Y., and Cheng, X. Efficient parallel translating embedding for knowledge graphs. In *Proceedings of the International Conference on Web Intelligence*, pp. 460–468, 2017.

Zhang, D., Yuan, Z., Liu, H., Lin, X., and Xiong, H. Learning to walk with dual agents for knowledge graph reasoning. In *Thirty-Sixth AAAI Conference on Artificial Intelligence*, 2022a.

Zhang, Y. and Yao, Q. Knowledge graph reasoning with relational digraph. In *The Web Conference*, 2022.

Zhang, Y., Zhou, Z., Yao, Q., Chu, X., and Han, B. Adaprop: Learning adaptive propagation for graph neural network based knowledge graph reasoning. In *Conference on Knowledge Discovery and Data Mining*, 2022b.

Zheng, S., Wang, W., Qu, J., Yin, H., Chen, W., and Zhao, L. Mmkgr: Multi-hop multi-modal knowledge graph reasoning. In *2023 IEEE 39th International Conference on Data Engineering*, pp. 96–109. IEEE, 2023.

Zhou, K., Huang, X., Li, Y., Zha, D., Chen, R., and Hu, X. Towards deeper graph neural networks with differentiable group normalization. In *Advances in Neural Information Processing Systems*, 2020a.

Zhou, K., Song, Q., Huang, X., Zha, D., Zou, N., and Hu, X. Multi-channel graph neural networks. In *International Joint Conferences on Artificial Intelligence*, 2020b.

Zhou, K., Huang, X., Zha, D., Chen, R., Li, L., Choi, S.-H., and Hu, X. Dirichlet energy constrained learning for deep graph neural networks. *Advances in Neural Information Processing Systems*, 34:21834–21846, 2021.

Zhou, K., Choi, S.-H., Liu, Z., Liu, N., Yang, F., Chen, R., Li, L., and Hu, X. Adaptive label smoothing to regularize large-scale graph training. In *Proceedings of the 2023 SIAM International Conference on Data Mining (SDM)*, pp. 55–63. SIAM, 2023.

Zhu, Z., Yuan, X., Galkin, M., Xhonneux, S., Zhang, M., Gazeau, M., and Tang, J. A*net: A scalable path-based reasoning approach for knowledge graphs. In *Advances in Neural Information Processing Systems*, 2023.

A. More Experiments

A.1. More Settings

We provide the statistics of the five datasets in Table 4.

Table 4. The statistics of knowledge graph datasets.

	YAGO3-10	**NELL-995**	**WN18RR**	**FB15K237**	**FB15K**
# Train	1,079,040	149,678	86,835	272,115	483,142
# Valid	5,000	543	3,034	17,535	50,000
# Test	5,000	3,992	3,134	20,466	59,071
# Entities	123,182	75,492	40,943	14,541	14,951
# Relations	37	200	11	237	1345

For hyperparameters used in our approach, we refer the readers to our implementation `https://github.com/Escanord/Intersection_Features` for details.

A.2. Experiments on Smaller Scale More Homogeneous Knowledge Graph Datasets

We present the results on datasets with more homogeneous and smaller-scale KGs in Table 5 in the supplementary materials. Our approach consistently demonstrates superior performance when compared to embedding-based models across the majority of benchmark datasets. Additionally, our results align with the top-performing GNN-based methods in five specific metrics, showing only marginal differences.

A.3. Efficiency of Intersection Features

We report the total training time of all models on each dataset that is reproduced by us in Table 6. Remarkably, our method, leveraging straightforward yet highly effective intersection features, attains the training time on par with embedding-based methods renowned for their simplicity and speed. In fact, it outperforms all GNN baselines, and our method's efficiency remains closely comparable to the fastest training method, i.e. TransE. Notably, the training time for our method on all 5 datasets is consistently less than 3 hour, while pre-computing all MinHash & HyperLogLog values to estimate intersection features of all links takes less than 0.25 seconds. Note that there is a possibility that both our implementation and baselines can be accelerated. In the future, we would like to further accelerate the learning process with novel distributed learning techniques (Wang et al., 2023a;c; 2022; 2023b;d).

A.4. Ablation Study on k

In this section, we perform a parameter study on the proposed approach. In Table 7, we vary the k in the k-hop intersection features in Section 4.4 and test the DNN model's performance using the NELL-995 dataset. We observe that intersection features built upon just 1-hop neighborhoods may not be effective enough to achieve the best performance in link prediction. However, the intersection features with k being 2 and 3 are close to each other. The results across all metrics indicate that our approach demonstrates a degree of robustness to different k parameters. Consequently, $k = 2$ may be preferred, particularly when resources are limited, or when we want to achieve more efficient training and inference times.

A.5. Ablation Study on Hashing Hyperparameters

In this section, we perform a parameter study to examine the effectiveness and robustness of our proposed hashing method's hyperparameters, i.e. MinHash and HyperLogLog. In Table 8 and Table 9, we vary the number of MinHash functions and HyperLogLog's p respectively. Per each variation, we test the DNN model's performance using the NELL-995 and WN18RR datasets. The results across all metrics indicate that our approach demonstrates a degree of robustness to different hash parameters. Thus, we recommend that 128 MinHash functions and $p = 8$ are enough for modeling KG with our proposed method.

A.6. A Study on Semi-Inductive Knowledge Graph Completion Setting

In this section, we provide a simple yet intuitive answer to explain the effectiveness of our proposed method. To start with, we introduce a semi-inductive setting for the KG completion task. We borrow the design of (Teru et al., 2020), where we

Table 5. Experimental results of link prediction on smaller scale, more homogeneous knowledge graph datasets

Method	Source	FB15K237				FB15K			
		MRR	H@1	H@3	H@10	MRR	H@1	H@3	H@10
Intersection Features (Ours)		**0.625**	**0.617**	**0.620**	0.644	0.699	0.689	0.705	0.715
	Self-Replicated	0.253	0.132	0.310	0.487	0.521	0.390	0.610	0.744
TransE	(Bordes et al., 2013)	–	–	–	–	–	–	–	0.471
(Bordes et al., 2013)	(Zhang et al., 2022a)	0.361	0.248	0.401	0.450	–	–	–	–
	(Li et al., 2022a)	0.330	0.231	0.369	0.528	–	–	–	–
TransH	Self-Replicated	0.284	0.183	0.326	0.483	0.525	0.387	0.626	0.753
(Wang et al., 2014)	(Wang et al., 2014)	–	–	–	–	–	–	–	0.644
TransR	Self-Replicated	0.253	0.132	0.310	0.487	0.521	0.390	0.610	0.744
(Lin et al., 2015)	(Lin et al., 2015)	–	–	–	–	–	–	–	0.702
DistMult	(Yang et al., 2015)	–	–	–	–	0.360	–	–	0.585
(Yang et al., 2015)	(Tran & Takashu, 2022)	–	–	–	–	0.313	0.224	–	0.490
	(Yang et al., 2021)	–	–	–	–	0.370	0.275	0.417	0.568
ComplEx-N3	Self-Replicated	0.354	0.261	0.391	0.542	0.835	**0.793**	0.863	0.907
(Lacroix et al., 2018)	(Lacroix et al., 2018)	–	–	–	–	**0.860**	–	–	0.910
	(Yang et al., 2021)	0.365	0.270	0.403	0.558	–	–	–	–
ComplEx	Self-Replicated	0.280	0.193	0.310	0.458	0.618	0.495	0.708	0.818
(Trouillon et al., 2016)	(Trouillon et al., 2016)	–	–	–	–	0.692	0.599	0.759	0.840
	(Das et al., 2018)	0.394	0.303	0.434	0.572	–	–	–	–
RotatE	Self-Replicated	0.322	0.221	0.363	0.523	0.724	0.652	0.769	0.848
(Sun et al., 2019)	(Sun et al., 2019)	0.338	0.241	0.375	0.533	0.797	0.746	0.830	0.884
KBGAT	Self-Replicated	0.526	0.449	0.565	**0.665**	0.800	0.699	**0.892**	**0.935**
(Nathani et al., 2019)	(Nathani et al., 2019)	0.518	0.460	0.540	0.626	–	–	–	–
CompGCN	Self-Replicated	0.394	0.129	0.213	0.482	0.694	0.524	0.687	0.844
(Vashishth et al., 2020)	(Vashishth et al., 2020)	0.355	0.264	0.390	0.535	–	–	–	–
SE-GNN	Self-Replicated	0.365	0.271	0.401	0.554	0.573	0.465	0.640	0.761
(Li et al., 2022a)	(Li et al., 2022a)	0.365	0.271	0.399	0.549	–	–	–	–
KRACL	Self-Replicated	0.358	0.264	0.394	0.545	0.602	0.498	0.666	0.784
(Tan et al., 2023)	(Tan et al., 2023)	0.360	0.266	0.395	0.548	–	–	–	–
AdaProp	(Zhang et al., 2022b)	0.417	0.331	–	0.585	–	–	–	–
MEIM	(Tran & Takashu, 2022)	0.369	0.274	0.406	0.557	–	–	–	–
HousE	(Li et al., 2022b)	0.361	0.266	0.399	0.551	0.811	0.759	0.847	0.898
NCRL	(Cheng et al., 2023)	0.30	0.209	–	0.473	–	–	–	–
A*Net	(Zhu et al., 2023)	0.411	0.321	0.453	0.586	–	–	–	–

split a dataset into train/val/test set, where there are three graphs disjoint in terms of triples. Methods learn on train graph, conduct hyperparameter tuning on the validation set, and evaluate on the test set. We would like to note that all entity nodes within the test set are also included in the train set (though test relation edges are invisible during training). During the evaluation of the test set, we remove one triple from the full test set, and only feed the test set, without the removed triple, a triplet as the input. We then conduct the standard (s, r, ?) corruption query.

We report the performance of our method under the introduced semi-inductive setting vs. the typical transductive setting of the KG completion task in Table 10. When applying our method in the semi-inductive setting, we utilize the test graph as input, and our method tends to perform better when the (# test triples) / (# test relations) ratio in the Table 4 is large; as a larger such ratio suggests there will be more intersection features to utilize. For example, our method still performs pretty well on YAGO3-10, NELL-995, and WN18RR with large (# test triples) / (# test relations) ratios, but not so well on FB15K237 and FB15K where such ratios are relatively smaller.

Table 6. The training time of methods (hours : minutes)

Method		YAGO3-10	NELL-995	WN18RR	FB15K237	FB15K
Intersection Features		02:31	00:54	00:32	02:17	01:26
Knowledge Graph Embedding	**TransE**	04:31	00:13	01:15	01:59	02:03
	TransH	01:27	00:17	00:11	03:52	00:40
	TransR	17:38	03:49	02:01	07:11	12:11
	DistMult	05:20	01:56	00:43	02:48	03:24
	ComplEx	03:54	01:42	00:43	01:13	04:22
	RotatE	37:23	04:17	02:26	05:36	10:17
GNN Baselines	**KBGAT**	09:56	01:07	00:39	01:58	03:34
	CompGCN	06:54	02:29	02:42	00:55	02:49
	SE-GNN	54:11	16:17	17:19	12:14	46:40
	KRACL	136:23	6:23	3:03	24:10	108:18

Table 7. Intersection Features performance with different k parameter shown in Section 4.4.

	NELL-995			
	MRR	**H@1**	**H@3**	**H@10**
3-hop Intersection Features	0.837	0.835	0.837	0.840
2-hop Intersection Features	0.834	0.827	0.842	0.846
1-hop Intersection Features	0.789	0.776	0.791	0.813

Table 8. Intersection Features performance with different numbers of MinHash functions shown in Section 4.2.

# functions	**NELL-995**				**WN18RR**			
	MRR	H@1	H@3	H@10	MRR	H@1	H@3	H@10
32	0.767	0.756	0.766	0.767	0.518	0.516	0.517	0.523
64	0.783	0.769	0.795	0.797	0.343	0.341	0.344	0.347
128	0.789	0.776	0.791	0.813	0.781	0.724	0.891	0.899
256	0.644	0.641	0.643	0.650	0.580	0.579	0.581	0.582

Table 9. Intersection Features performance with different p parameter shown in Section 4.3.

p	**NELL-995**				**WN18RR**			
	MRR	H@1	H@3	H@10	MRR	H@1	H@3	H@10
4	0.677	0.668	0.672	0.676	0.500	0.492	0.500	0.520
6	0.719	0.718	0.717	0.719	0.542	0.533	0.535	0.568
8	0.789	0.776	0.791	0.813	0.781	0.724	0.891	0.899
10	0.680	0.660	0.684	0.696	0.599	0.595	0.600	0.610

Subsequently, when we apply our method under the typical transductive setting, where we can utilize the full train graph as input and only query a triple from the test graph, the (# test triples) / (# test relations) no longer matter, but (# train triples) / (# test relations) do. However, in such case, even the smallest (# train triples) / (# test relations) (502.75 for FB15K) is much larger than the largest (# test triples) / (# test relations) (332.67 for NELL-995), so our method's performance is significantly boosted. It can be observed that under the typical transductive setting, our method achieves SOTA on FK15K237.

We further note our method is competitive but not good enough on FB15K, this is likely because FB15K's test set includes many inverse relations of its train set, where some methods specifically model upon this observation to score on such inverse relations. Our proposed method does not leverage this leakage and, therefore, might lack the last few points to be competitive on FB15K.

Table 10. Intersection Features performance under transductive setting vs. semi-inductive setting.

Setting	**Dataset**	**MRR**	**H@1**	**H@3**	**H@10**
Transductive	WN18RR	0.781	0.724	0.891	0.899
	NELL-995	0.789	0.776	0.791	0.813
	YAGO3-10	0.617	0.590	0.665	0.674
	FB15K237	0.625	0.617	0.620	0.644
	FB15K	0.699	0.689	0.705	0.715
Semi-Inductive	WN18RR	0.886	0.870	0.874	0.966
	NELL-995	0.738	0.490	0.984	0.985
	YAGO3-10	0.646	0.482	0.960	0.969
	FB15K237	0.332	0.155	0.514	0.726
	FB15K	0.804	0.803	0.804	0.893