DeCon: Path-based Conflict Detection for Knowledge Graph

Yihan Wang
University of Science and Technology of China
Hefei, China
yihanw@mail.ustc.edu.cn

Qi Song
University of Science and Technology of China
Hefei, China
qisong09@ustc.edu.cn

Ling Zheng
University of Science and Technology of China
Hefei, China
lingzheng@mail.ustc.edu.cn

Xiang-Yang Li
University of Science and Technology of China
Hefei, China
xiangyangli@ustc.edu.cn

ABSTRACT

Knowledge graphs (KGs), which utilize graph structures to represent real-world facts and associations, are pivotal in information retrieval and artificial intelligence. However, the necessity of constant updates often leads to errors or conflicts. Some conflicts, such as expression ambiguities, are challenging to detect at triple granularity and require semantic analysis together with other triples. In this paper, we define *path-based conflict* in KGs, extending error detection from triple granularity to path granularity. We demonstrate that path-based conflicts are confined to ring structures in KGs. We prove that indicating whether a triple brings such conflicts only requires dealing with any one of its related ring structures, which significantly simplifies our problem. We propose RingE, a novel ring embedding method, to detect conflicts, and DeCon, a conflict detection framework suitable for both static and incremental scenes. This framework divides the path-granularity conflict detection task in KGs into ring discovery and conflict detection. We also design a pruning strategy for ring discovery to improve efficiency. Using real-world graphs, we experimentally verify that our algorithms are effective and feasible for large graphs. Our case study also verifies that the algorithms can detect real conflicts.

1 INTRODUCTION

Knowledge graphs (KGs) are typically used to store and represent real-world information [22, 23], which have found prevalent use in data-driven and cross-domain applications such as those related to data management [43, 53], information retrieval [6, 27, 59], and artificial intelligence. Notably, even today, when large language models (LLMs) show strong power in the above applications, KGs continue to demonstrate their vital role, particularly in dealing with problems for domain-specific knowledge and LLM hallucination [40, 42, 57]. Litslink predicts that, in the next $3-5$ years, more than 30% of organizations will apply graphs to drive greater contextualization in decision-making [36].

EDBT'25, 25-28 March, 2025, Barcelona, Spain

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

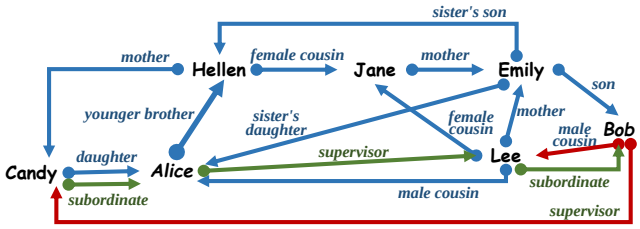


Figure 1: Path-based KG error with two relation types. We label different types with different colors: blue indicates kinship, green indicates working relationship, and red is error.

Critical to decision-making is the quality of the KGs. If the KGs contain inconsistent or inaccurate triples, decisions made based on such KGs are unreliable. Real-world KGs are often constructed by either using heuristic algorithms [10, 39] or crowdsourcing [46]. The heuristic methods may generate incorrect triples or miss some triples due to the lack of correctness guarantee [51], while the crowdsourcing methods require a lot of manual work to generate correct triples. Thus incorrect triples exist in real-world graphs. For example, in Nell [10], the relation between Colin Hanks and Tom Hanks is labeled as "Parent of a person" with Colin as the head entity and Tom as the tail entity. While in Wikipedia [2], Tom is, in fact, the parent (father) of Colin. Such data quality issues are even critical in domain-specific applications. Errors in malfunction detection KGs are considered a big challenge in troubleshooting. Any uncorrected errors or conflicts within the KGs can significantly hinder rather than help engineers [26, 37].

Real-world KGs are dynamic as they are continuously being updated and revised. In this paper, we thus consider the two most common forms of errors in KGs: existing conflicts within the KG and conflicts brought by newly added triples [51]. Take PC malfunction detection KG as an example, which includes software, hardware, alarm information, fault types, fault causes, and processing methods. One PC can contain up to $3,000$ types of materials and up to $40,000$ collocation relations. This KG will become rather complex due to the interplay and updating of its diverse elements. Thus, the KG for all series of PCs is rather large, with all kinds of materials and their variants. Due to the impracticality of manual error detection in large and continuously updated graphs, the automated method is essential to handling a large amount of data in KGs efficiently.

Automated error detection in KGs has been widely studied, which includes either learning-based methods or rule-based methods [11, 31, 50, 55]. Both methods focus on discussing the correctness of a

single triple. They pay more attention to entire triples or entities and hardly pay any attention to the relation between entity pairs. At the same time, the semantics of each independent triple cannot be directly considered correct or error because it needs to rely on semantic connections with other triples in the specific KG. These researches mostly ignore errors caused by the semantic connections of relations that are associated with the relation path. Such errors usually appear in the form of semantic conflicts and involve a series of interrelated triples (i.e., relations). Consider the real-world example below.

Example 1: Fig. 1 depicts relations and entities within a family where two potential conflicts exist. This example focuses on the conflict involving Emily, Lee, and Bob. *(Lee, mother, Emily)*, *(Emily, son, Bob)*, and *(Bob, male cousin, Lee)* are all correct individually. However, when evaluating them together, there exists a conflict: both Lee and Bob have Emily as their mother, indicating that they are brothers, not cousins. In terms of semantic understanding, there are indeed differences in the expressions of cousin and brother, but this may also be a misunderstanding that often occurs in the real world. Especially when people communicate with each other, they usually ignore the difference between cousin and brother, which can cause misunderstandings among others. □

As shown in Example. 1, such conflicts cannot be detected when considering them only at triple granularity. However, these conflicts can be detected if considering path granularity with interrelated triples. Moreover, newly added triples are considered correct individually, especially those relations that have never appeared in the KG. However, these triples can bring conflict when semantically related to the existing triples. If these conflicts are not detected, they may lead to incorrect decisions in downstream tasks. For example, consider the triple *(alarm A, processing method, open B)* in a malfunction detection KG. This triple may be correct when the detection is isolated. However, at path granularity, another triple *(open B, generate fault, alarm A)* suggests that "open B" also causes "alarm A". This conflict where "open B" is both the reason and solution of "alarm A" highlights the necessity of evaluating triples at path granularity. Detecting such conflicts is a critical step in ensuring the quality of the decisions. Consequently, there is an urgent need to design an error detection method for relations in the paths. The paths between two entities are usually complex, and learning the rules from them will be rather challenging. Thus, we consider the learning-based method to detect conflict at path granularity. However, some challenges still need to be resolved.

Immense number of paths. A KG is typically seen as a directed graph, allows for varying numbers of paths between node (i.e. entity) pairs, ranging from 0 (no connection) to potentially infinite with loops. For a graph with $|V|$ entities, there are $|V|^2 - |V|$ potential node pairs (in directed graphs, (v_1, v_2) and (v_2, v_1) are different pairs, $v_1, v_2 \in V$). If the average number of the paths per pair is n, the total number of paths is $n \cdot (|V|^2 - |V|)$. Traversing all paths to detect conflicts is impractical, so we need to identify representative paths that cover as many conflicts as possible in the KG.

Long path lengths. As mentioned earlier, path lengths range from $[0, +\infty)$. Paths with loops lead to repetition, so segments before the first loop are crucial. Even without loops, the number of paths remains large, with a theoretical upper bound of $(|V| - 1)!$. Therefore, it is important to limit the maximum length of paths to cover as many conflicts as possible.

Varying path lengths. Current error detection methods for KGs only require inputting the element of each triple, with a fixed input length of 3 at triple granularity. However, for path granularity, the length of inputs varies due to the differing lengths of the paths. Thus, the error detection method at path granularity must handle different input length. Additionally, embedding methods at path granularity must adapt to the diversity of paths between the same node pair. This means the embedding need to reflect the relationship between a node pair and remains constant regardless of the path taken. For instance, the kinship between two individuals (i.e., entities) is consistent, even if the description (i.e., path) involves different entities and relations. To sum up, the varying input length and path diversity between the same node pair make detection and embedding at path granularity significantly distinct from those at triple granularity in KGs.

In this paper, we focus on a novel *path-based conflict detection* in KGs, extending the error detection problem from triple to path granularity. Our work is designed to detect conflicts within the relations of paths in KGs. In order to solve this problem, we propose a unique ring embedding method, RingE, for *detecting* conflicts within rings, along with a conflict detection framework DeCon.

Contributions. This paper studies path-based conflict detection method in KGs. We summarize the main contributions of this work as follows:

- We define *path-based conflict* in KGs, an extension of error detection from the granularity of triples to that of paths. We theoretically prove that such conflicts are confined to ring structures within KGs. We theoretically demonstrate that, in order to detect potential conflicts brought by a specific triple, we only need to analyze one of its related ring structures.
- We propose a novel ring embedding method RingE, which uniquely combines head and tail entities with relations in the connected path and enables the detection of conflicts within KGs.
- We propose DeCon, the first framework for conflict detection within rings in KGs. This framework is divided into *ring discovery* and *conflict detection*, accommodating the requirements of both static and incremental scenes. DeCon implements a pruning strategy for ring discovery, significantly reducing the time complexity from $O(2^{|V|} \cdot (|V| + |E|))$ to $O(\frac{\zeta}{2\alpha} \cdot (|V| + |E|))$ with $|V|$ entities and $|E|$ triples, which ζ, α are both constant.
- We evaluate our work using three real-world datasets. The experimental results show that RingE effectively detects conflicts within ring structures with an accuracy of more than 90% and a recall of 100%, and DeCon has good performance where several magnitudes reduce the time consumption while improving the efficiency of detecting conflicts by up to 85%. We also show that DeCon can detect real conflicts in the case study.

These results yield effective methods for conflict detection in large graphs. We provide detailed proofs and additional experiment results in [1].

2 RELATED WORK

2.1 Error Detection in Knowledge Graph

Error detection is an essential part of quality management in KGs. The main goal is to detect errors or outdated information [12], which can affect the reliability and usefulness of the KG [54]. Traditionally, detecting errors in KGs is done during their construction. But recently, more focus has been on detecting errors during the learning of knowledge representation. There are two main types of methods for this. First, there are embedding-based methods, like TransE [7] and its variations [29, 35, 47], ComplEx [45], and DistMult [52]. The second type is path-based methods [11, 15, 31, 50, 55, 58], which use the ideas of resource allocation [34] and confidence [8]. Most methods use TransE as the basic embedding method and add a confidence framework in the implementation. Several specific methods have been studied. For example, CKRL [50] uses the structure of the KG itself to find errors. SCEF [58] looks at both the KG structure and external text to fix errors. KGTtm [31] combines internal information with overall KG. PGE [11], for product KGs, integrates text and structure and adds triple confidence to facilitate error detection. While GAGED [55] creatively uses an unsupervised approach.

However, these methods mostly focus on simple errors at triple granularity in the KG and don't address complex relations formed by linking multiple triples. Therefore, errors and conflicts caused by relation derivations cannot be effectively detected. In addition, some researchers extend methods used in other types of entity resolution and conflict resolution methods to KGs [4, 5, 13, 16, 18, 20, 21, 28, 38]. They use specific rules (like GFDs [21] and NGFDs [18]) and develop parallel algorithms [17, 19] to detect inconsistencies. These methods are more about mining and applying rules to detect errors in attributes, while our problem focuses on the semantics.

2.2 Knowledge Representation Learning.

Representation learning involves the process of converting semantic information from images, texts, speech, and other mediums into low-dimensional, dense vectors, commonly referred to as embeddings. Knowledge representation learning (KRL) specifically deals with representing entities and relationships in KGs as such dense vectors. Traditional KRL methods, including the TransE series [7, 29, 35, 47], ComplEx [45], DistMult [52], SimplE [32], and RotatE [44], vary in aspects such as representation space, scoring functions, coding models etc [30]. In recent years, the diverse range of downstream applications has necessitated knowledge representations that encapsulate entity semantics, thereby enhancing the effectiveness of KRL. This requirement has led to the development of various text-enhanced KRL methods [3, 48, 49, 60]. For example, DKRL [49] employs entity description-enhanced representation learning, utilizing two distinct encoders to encode the semantics of entity descriptions. However, most error detection methods primarily rely on traditional KRL approaches. These approaches involve converting entities directly into vectors, learned from the triples in KGs. Thus, some researchers have incorporated descriptions of entities and relations in this process [11, 58]. However, we should recognize that conflict detection needs semantics. Therefore, we propose enhanced knowledge representation by integrating entity descriptions into the framework to improve its effectiveness.

3 PROBLEM FORMULATION

We start with the notions of knowledge graphs.

Graphs. We consider a directed graph $G = (V, E, L_E, \mathcal{L}_E)$, where V is a finite set of nodes, which represents the set of entities, and $E \subseteq V \times V$ is a set of edges, i.e., relationships. Each edge $e \in E$ carries two labels $L(e)$ and $\mathcal{L}(e)$. In practice, the labels of the edges represent relations (predicates) and types, respectively. We use r to correspond to $L(e)$ for each edge $e \in E$.

Relation Path and Path Set. Given two connected entities randomly selected from the node-set V, there can be multiple paths between them. Each path consists of an ordered sequence of several relations and entities that link one entity to another. The start entity of the path is referred to as the head entity h and the end entity as the tail entity t. Thus, the *Relation Path* is denoted as $p_{h\to t} = (r_1, v_1, r_2, v_2, \ldots, v_{m-1}, r_m)$, where v_i and v_{i+1} is connected via $r_{i+1} \in L_E$ for $i \in \{0, m-1\}$. Denote h as v_0 and t as v_m, these entities and the intervening relations are represented as a *Path Triple* $(h, p_{h\to t}, t) = (h, (r_1, v_1, r_2, v_2, \ldots, v_{m-1}, r_m), t)$. Notably, we do not allow any loop in the path as illustrated in Example 2. Thus, for each pair of two entities, we define the *Path Set* of such paths as $P_{h\to t} = \{p^1_{h\to t}, p^2_{h\to t}, \ldots, p^l_{h\to t}\}$, while l is the number of loop-free paths from h to t.

Example 2: In Fig. 1, the path set from "Alice" to "Bob", $P_{Alice\to Bob}$, includes the two relation paths: *(younger brother, Hellen, female cousin, Jane, mother, Emily, son)* and *(supervisor, Lee, subordinate)*. Both paths connect Alice to Bob without any entity repetition. However, the path *(supervisor, Lee, male cousin, Alice, supervisor, Lee, subordinate)* also links Alice to Bob but is excluded from $P_{Alice\to Bob}$ due to the repeated entity "Lee". □

Semantic of the Relation. We use $\mathcal{L}(e)$ of each edge to represent its type, i.e., the relation type. We consider an edge labeling function $\mathbb{L}_E : E \to \mathcal{L}_E$, which maps each edge $e \in E$ to a label $\mathcal{L}(e)$ within a predefined set of labels $\mathcal{L}_E$. Relation type categorizes the various kinds of relations between entities in knowledge graphs. For instance, kinship, friendship, and working relationships are all examples of relation types that describe specific connections between two individuals (i.e., entities).

Example 3: Fig. 1 presents two distinct relation types: kinship in blue, and working relationship in green. We can easily find a path pair including both of them, such as *{(supervisor, Lee, mother, Emily, son), (supervisor, Candy, daughter)}* connecting Alice and Bob. However, since kinship and working relationships are unrelated categories within this KG, it is illogical to use information from kinship relations to infer or deduce aspects of working relationships, and vice versa. Thus, assessing the correctness of relations in this mixed path pair is not feasible. □

We observe that only relations with the same type can be jointly assessed for semantic correctness. Therefore, we evaluate the correctness of path pairs with identical relation types. This means, in DeCon, the relations within each relation path have the same type, and all relations in one detection must also have same type.

Conflict. We consider the conflict as the data quality issue we want to solve in this paper. There exists a path triple $(h, p^a_{h\to t}, t)$ from h to t and anther reversed one $(t, p^b_{t\to h}, h)$ from t to h, both

constructed using the same type of relations. Conflict refers to a significant semantic inconsistency between these two path triples. Specifically, within the path sets $P_{h\to t}$ and $P_{t\to h}$, we can get a pair of paths denoted as $(p^a_{h\to t}, p^b_{t\to h})$ with manifest inconsistencies in their relations and the semantics conveyed by the involved entities.

Semantic Consistency. The path from h to t is $p^a_{h\to t} = (r^a_1, v^a_1, r^a_2, v^a_2, \ldots, v^a_{m-1}, r^a_m)$, and the path from t to h is $p^b_{t\to h} = (r^b_1, v^b_1, r^b_2, v^b_2, \ldots, v^b_{n-1}, r^b_n)$. Following the idea introduced in TransE [9], we define

$$\vec{A} = \vec{r^a_1} + \vec{r^a_2} + \cdots + \vec{r^a_m} = \vec{t} - \vec{h},$$
$$\vec{B} = \vec{r^b_1} + \vec{r^b_2} + \cdots + \vec{r^b_n} = \vec{h} - \vec{t}. \quad (1)$$

Semantic consistency means $||A + B|| \leq \epsilon$, where ϵ is a predefined small real value.

Semantic Inconsistency. Based on the above definition, semantic inconsistency satisfies $||A + B|| > \epsilon$.

Example 4: Consider the path pair *{(female cousin, Jane, mother), (sister's son)}* between Hellen and Emily in Fig. 1. From the first path, it's evident that Hellen and Jane are cousins, suggesting their mothers are different. Since Jane's mother is Emily, Hellen is likely Emily's sister's son, indicating consistency in this path pair as the relations and semantics align logically. However, the path pair between Lee and Bob, *{(mother, Emily, son), (male cousin)}*, shows semantically inconsistent. If we accept that both Bob and Lee are Emily's children, they cannot be cousins but must be brothers, which meets the condition of semantic inconsistency. Likewise, the path pair between Candy and Bob, *{(subordinate, Alice, supervisor, Lee, subordinate), (supervisor)}*, is also semantically inconsistent. □

Based on the previously defined concepts of relation path, path set and conflict, we can derive the following *Insight* and *Theorem*.

Insight 1: *Any pair of relation paths, denoted as $(p^a_{h\to t}, p^b_{t\to h})$, which connects the head entity h and the tail entity t, there exists at least a ring structurally.* □

Proof sketch: A pair of paths $(p^a_{h\to t}, p^b_{t\to b})$, connecting h and t are from a directed graph. Therefore, the relation path from h to t is represented as $\mathcal{V} = (v_1, v_2, \ldots, v_g)$, while the path from t to h is denoted as $\mathcal{V}' = (v'_1, v'_2, \ldots, v'_q)$ using only the entities involved.

- If $\forall f \in (v_1, v_2, \ldots, v_g), f \notin (v'_1, v'_2, \ldots, v'_q)$ (Fig. 2(a) as an example), we randomly select a node k in the set of nodes $N = \mathcal{V} \cap \mathcal{V}'$, and there exists paths respectively from k to t in $\mathcal{V}$, from t to h in $\mathcal{V}'$, and from h to k in $\mathcal{V}$. The ring structure is confirmed.
- If $\exists f \in (v_1, v_2, \ldots, v_g), f \in (v'_1, v'_2, \ldots, v'_q)$ (Fig. 2(b) and 2(c) are examples). Let the set of nodes satisfying these conditions be represented as $F = (f_1, \ldots, f_j)$, preserving the order in the path.
  - For the segment containing h, there exists paths from h to f_1 in $\mathcal{V}$, and f_1 to h in $\mathcal{V}'$. Thus, a ring structure is established.
  - For the segment involving t, there exists paths from f_j to t in $\mathcal{V}$, and t to f_j in $\mathcal{V}'$. Thus, a ring structure is also established.
  - For other segments, randomly chosen $f_s \in (f_2, \ldots, f_{j-1})$. There are paths connecting f_s to f_{s+1} in $\mathcal{V}$, and f_{s+1} to f_s in $\mathcal{V}'$. Consequently, ring structures are also established.

To sum up, there must be at least one ring in $(p^a_{h\to t}, p^b_{t\to h})$. □

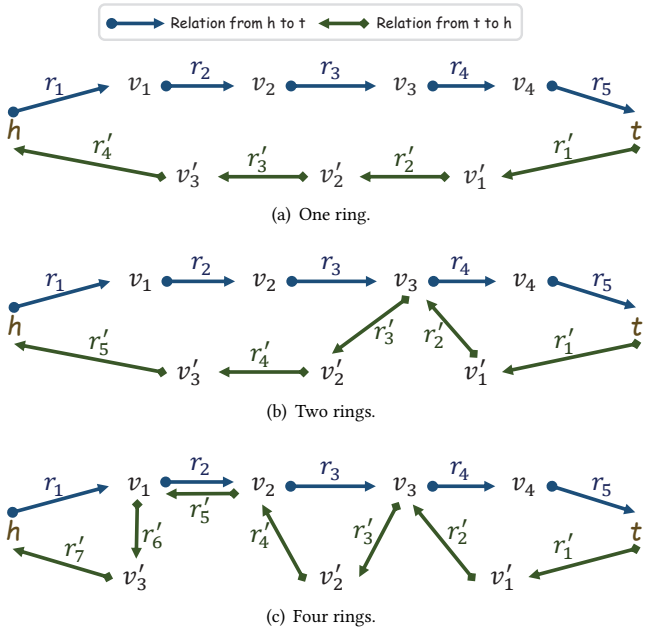


Figure 2: Examples of ring structures. Blue arrows indicate the path from head to tail, while green arrows indicate the path from tail to head.

Example 5: The path pair *{(younger brother, Hellen, female cousin, Jane, mother, Emily, son), (male cousin, Lee, male cousin)}* connecting Alice and Bob in Fig. 1 is the same as Fig. 2(a), features a single ring structure. Meanwhile, another path pair between Alice and Bob in Fig. 1, *{(younger brother, Hellen, female cousin, Jane, mother, Emily, son), (male cousin, Lee, mother, Emily, sister's daughter)}*, is the same as Fig. 2(b), containing two rings within the path pair structure. □

Theorem 1: *If there are conflicts between the head entity h and the tail entity t, then the conflicts must indeed exist within the rings of the path pair $(p^a_{h\to t}, p^b_{t\to h})$.* □

Proof: According to the above proof for Insight 1, a pair of paths $(p^a_{h\to t}, p^b_{t\to h})$ between h and t must be able to be divided into $(j+1)$ segments. Consequently, this pair of paths can be deconstructed into sets of sub-paths, each of which is demarcated by a common intermediary entity. These sub-paths are represented as follows: $< h \to f_1, f_1 \to h >, < f_1 \to f_2, f_2 \to f_1 >, < f_2 \to f_3, f_3 \to f_2 >, \ldots, < f_{j-1} \to f_j, f_j \to f_{j-1} >, < f_j \to t, t \to f_j >$.

Hence, these sub-paths can be represented as

$$\begin{cases} < f_{s-1} \to f_s, f_s \to f_{s-1} > & s \in [2, j] \\ < h \to f_1, f_1 \to h > & s = 1 \\ < f_j \to t, t \to f_j > & s = j. \end{cases}$$

We define a conflict detection function $De()$, which takes a path pair as input, and outputs conflict detection results (0 for no conflict, 1 for conflict). If there are conflicts in $(p^a_{h\to t}, p^b_{t\to h})$, then

$$De(p^a_{h\to t}, p^b_{t\to h}) = Step[\sum_{s=1}^{j-1} De(< f_{s-1} \to f_s, f_s \to f_{s-1} >) +$$

$$De(< h \to f_1, f_1 \to h >) + De(< f_j \to t, t \to f_j >)]$$

Where $Step[]$ is the step function in the activation function. In this definition, if it is known that there are conflicts in the given pair of paths $(p^a_{h\to t}, p^b_{t\to h})$, $De(p^a_{h\to t}, p^b_{t\to h})$ returns 1, indicating the existence of conflicts. Otherwise, it returns 0 to signify the absence of conflicts. To sum up, if it is known that there is a conflict in the pair of paths $(p^a_{h\to t}, p^b_{t\to h})$, there must be conflicts exist in one or more of the sub-paths mentioned above. Conversely, if there is no conflict detected within any sub-paths, then the pair of paths $(p^a_{h\to t}, p^b_{t\to h})$ is considered to be conflict-free.

Therefore, conflicts are confined within the ring structure. □

Based on the preceding proof, it is established that conflicts within a KG exist within the ring structures in the KG. Therefore, conflict detection can be performed by discovering and analyzing ring structures. In light of this, we propose the following definition for conflict detection within a KG.

Problem statement. Given a KG, the objective is to discover all the ring structures within the KG and assess whether each of them exhibits semantic consistency or inconsistency between the entities and relations involved. When discovering a ring $(v_1, r_1, v_2, r_2, \ldots, r_{w-1}, v_w, r_w)$, where r_w is the relation from v_w to v_1, assuming that there exists a vector space for the representation of entities and relations, where entities are denoted by $\{\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_w}\}$ and relations by $\{\mathbf{r_1}, \mathbf{r_2}, \ldots, \mathbf{r_w}\}$, based on the idea $h+r \simeq t$ of TransE [7], we can describe the ring structure as follows:

$$\begin{aligned}\overrightarrow{\mathbb{R}} = &\quad (\mathbf{v_1} + \mathbf{r_1} - \mathbf{v_2}) + (\mathbf{v_2} + \mathbf{r_2} - \mathbf{v_3}) + \cdots + (\mathbf{v_w} + \mathbf{r_w} - \mathbf{v_1}) \\ = &\quad \mathbf{r_1} + \mathbf{r_2} + \cdots + \mathbf{r_w}\end{aligned} \tag{2}$$

The problem of conflict detection can be represented as:

$$||\overrightarrow{\mathbb{R}}|| = ||\mathbf{r_1} + \mathbf{r_2} + \cdots + \mathbf{r_w}|| \overset{?}{\leq} \epsilon \tag{3}$$

This problem is, nevertheless, nontrivial.

4 DeCon DESIGN

Conflict detection within the KG can be transformed into conflict detection in the rings within the KG. This requires initially discovering all rings before conflict detection within a KG. In order to detect conflict effectively, it is essential to establish a vector space and develop an operator for semantic computation on these rings. Thus, we need to address three hierarchical questions. 1) Can we reduce the complexity of conflict detection on the rings? 2) Which embedding method can represent rings for conflict detection? 3) How to apply the conflict detection methods to real systems? We next answer these questions in detail in this section.

Initially, it is important to evaluate the solvability of the conflict detection problem. If the complexity of conflict detection for a single triple is too high, let alone extending it to the entire KG. To address this, we first explore strategies to effectively reduce complexity, which is essential for the practicality and scalability of our methods. Then, we propose a ring embedding method, designed specifically for conflict detection within KGs. Considering the expensive cost of ring discovery in large graphs, we explore a pruning strategy to further reduce the complexity, enabling efficient and effective application of our methods to large-scale graphs. Finally, to support real-world applications, we present a ring discovery algorithm and a real system framework for conflict detection in both static and incremental scenes.

4.1 Complexity Reduction with Ring

When considering whether a specific edge (i.e., triple) is associated with a conflict, it is necessary to analyze all rings that include this edge (i.e., triple) as Theorem 1 shows. However, in cases where the length of the ring is not predefined, the number of such rings could potentially be infinite. Importantly, nodes and edges in a KG carry unique semantics that distinguishes them from corresponding nodes and edges in simple graph structures, and *relations of the same type within a KG share semantic relevance and can be logically derived, while relations of different types lack significant connections for meaningful discussion* (As we state in Sec. 3). Based on the observation, we only focus on rings composed of relations of the same type. Even if we limit the length of the ring and make the relations within the ring all the same, we also need to do conflict detection for all the rings containing this specific edge (i.e., triple). Nonetheless, we still have to traverse every ring. Given the potentially vast number of rings, discovering and traversing all rings is a highly time-consuming task.

Based on this understanding, we explore strategies for reducing the time complexity of conflict detection. To achieve this goal, we give the following property and lemma, which may hold the key to reducing time complexity.

Property. Since the semantic association between two entities is fixed, and the semantic association of each type is unique, the paths between two entities formed by relations of the same type need to convey the same semantic. In other words, different paths with the same type of relations which connect the same pair of entities, should ideally express consistent semantics. This can also extend to the embeddings of these paths, where these embeddings should also be consistent.

Lemma 2: *To determine whether a specific edge (i.e., triple) in a KG is implicated in conflicts, assessing only one of all rings that include this edge (i.e., triple) is enough.* □

Proof: Given an edge (h, r_{ht}, t) in a KG, which can be considered as a path $p_{h\to t}$ from h to t. We can find any two distinct paths from t to h in the KG, represented as $p_{t\to h}, p'_{t\to h}$. The proof for the Lemma starts from the relation below, which means we only use the relation in each triple to represent the edge in the KG. That's to say, $p_{h\to t} = (r_{h\to t})$,

$$p_{t\to h} = (r_1, r_2, \ldots, r_{i+1}), p'_{t\to h} = (r'_1, r'_2, \ldots, r'_{j+1})$$

Perform vector operations under ideal conditions, if there is a conflict in the ring $\overrightarrow{\mathbb{R}}_{h\to t\to h}$ formed by $p_{h\to t}$ and $p_{t\to h}$, as the shown in Equ. 3, then

$$\overrightarrow{\mathbb{R}}_{h\to t\to h} = R_{h\to t} + p_{t\to h} = r_1 + r_2 + \cdots + r_{i+1} + r_{h\to t} > \epsilon,$$

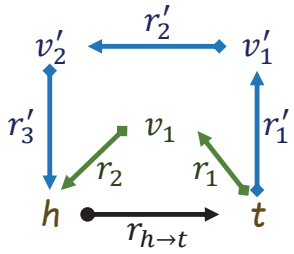


Figure 3: An example of Lemma 2. For the relation $r_{h\to t}$ from h to t, adding v_1 can get one ring (represented as green arrows with $r_{h\to t}$), while adding v'_1 and v'_2 can get the other ring (represented as red arrows with $r_{h\to t}$).

According to the above *Property*, there is $r_1 + r_2 + \cdots + r_{i+1} = r'_1 + r'_2 + \cdots + r'_{j+1}$, and then for the ring $\overrightarrow{\mathbb{R}}'_{h\to t\to h}$ formed by $p_{h\to t}$ and $p'_{t\to h}$, we can get

$$\overrightarrow{\mathbb{R}}'_{h\to t\to h} = R_{h\to t} + p'_{t\to h} = r'_1 + r'_2 + \cdots + r'_{j+1} + r_{h\to t} =$$

$$r_1 + r_2 + \cdots + r_{i+1} + r_{h\to t} = p_{h\to t} + p_{t\to h} = \overrightarrow{\mathbb{R}}_{h\to t\to h} > \epsilon.$$

That is to say, the ring $\overrightarrow{\mathbb{R}}'_{h\to t\to h}$ also has conflict.

In the same way, if there is no conflict in the ring $\overrightarrow{\mathbb{R}}_{h\to t\to h}$,

$$\overrightarrow{\mathbb{R}}_{h\to t\to h} = p_{h\to t} + p_{t\to h} = r_1 + r_2 + \cdots + r_{i+1} + r_{h\to t} \le \epsilon,$$

According to the above *Property*, there is $r_1 + r_2 + \cdots + r_{i+1} = r'_1 + r'_2 + \cdots + r'_{j+1}$, we also can get

$$\overrightarrow{\mathbb{R}}'_{h\to t\to h} = p_{h\to t} + p'_{t\to h} = r'_1 + r'_2 + \cdots + r'_{j+1} + r_{h\to t} =$$

$$r_1 + r_2 + \cdots + r_{i+1} + r_{h\to t} = p_{h\to t} + p_{t\to h} = \overrightarrow{\mathbb{R}}_{h\to t\to h} \le \epsilon.$$

That's to say, the ring $\overrightarrow{\mathbb{R}}'_{h\to t\to h}$ also has no conflict.

For extension, we also prove from the perspective of triple and using the idea of TransE, which means $h + r \simeq t$. The conclusion is also consistent with the above (see [1] for detailed proof, A.2). □

Based on Lemma 2, to evaluate a triple, we only need to do conflict detection on any one of the related rings. This can significantly reduce the time and cost by detecting only one ring instead of traversing all rings.

Example 6: In Fig. 1, the triple *(Alice, younger brother, Hellen)* is part of three distinct paths that contribute to forming ring structures, each marked with different line shapes in the figure. These paths are *(mother, Candy, daughter)*, *(female cousin, Jane, mother, Emily, sister's daughter)*, and *(female cousin, Jane, mother, Emily, son, Bob, male cousin, Bee, male cousin)*. For conflict detection, it's unnecessary to analyze all these paths. Instead, selecting just one, such as the shortest, is typically sufficient. □

4.2 RingE: Embedding the Ring

Different from the error detection at *triple* granularity, the conflict detection at *path* granularity requires an embedding method with semantics. In our work, we focus on the rings in the KG, so we need

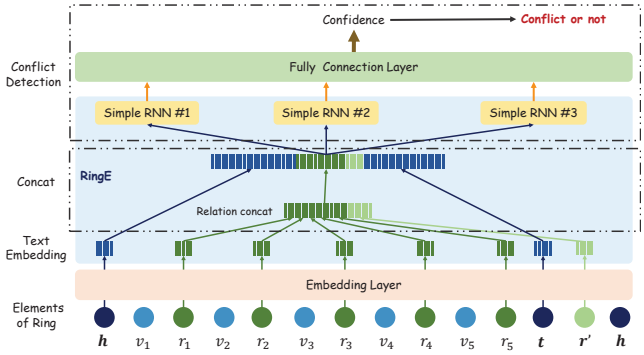


Figure 4: RingE and Conflict Detection Models. The blue embedding vectors are entities, and the green embedding vectors are relations. The three orange arrows form the ring embedding. RingE is gathering together with a blue background.

to get an embedding method to present the semantics for the rings. When randomly choosing two entities from the ring, We make one of the entities as the head entity and the other as the tail entity to evaluate whether there is conflict among them. Our goal is to make the difference of the inverse of the path embedding from head to tail entity and the path embedding from tail to the head entity as less as possible, as mentioned in Equ. 1 and 3 because they are semantically opposite. A critical limitation arises when considering the embeddings of paths. In an ideal scenario, the difference in the embedding of a path from one entity (denoted as the head entity) to another (denoted as the tail entity) and the reverse path from the tail to the head entity should be 0 (or ϵ), reflecting that they are semantically opposite. However, traditional KRL methods often fail to capture this nuance, leading to inadequate representation for conflict detection at path granularity.

So, in our design, we propose a ring embedding method RingE as illustrated in Fig. 4. RingE processes the initialized embeddings of entities and their relation sequence to produce an output that represents the embedding of the ring. Since TransE can get the semantics in triples and perform vector operations, TransE can be an effective method for path semantic initialization. We use the TransE model to initialize the embeddings of entities and relations within a ring structure, which then become the inputs for generating the ring embedding. Following the proofs in Sec. 3, we find that the relations in a path are more valuable than entities. Therefore, as shown in Fig. 4, RingE focuses on the relation sequence, excluding the entities from the input sequence, which helps simplify the representation. The objective function can be converted as follows.

$$\arg\min ||\overrightarrow{\mathbb{R}}|| = \arg\min ||\mathbf{r_1} + \mathbf{r_2} + \cdots + \mathbf{r_w}|| \tag{4}$$

Then, we have to integrate these initialized embeddings, each with its unique number of units, to generate three different outputs of the input sequence. To address the complexity of both individual entities and sequences of relations, we adopt the Simple RNN architecture as our baseline method. We employ three distinct Simple RNNs to complete the integration. These three outputs collectively form the embedding of the ring, which satisfies different concerns within the ring. The ring embedding, composed of the three RNN

modules' outputs, is then integrated using a full connection layer. This layer synthesizes the three diverse outputs and calculates a confidence score, which indicates the likelihood of conflict within the ring. Lower scores suggest a higher likelihood of conflict.

To enhance the conflict detection capabilities of RingE, we propose replacing the Simple RNN with more advanced sequence models like GRU and LSTM, which are better equipped to handle complex sequences. Additionally, we recommend replacing the TransE model with more advanced text embedding techniques, such as Bert and GPT. These models offer more robust mechanisms for capturing and representing the nuances. As depicted in Fig. 4, our method, RingE, is designed to be flexible, allowing for integrating other advanced models. By upgrading the text embedding and ring embedding modules to more powerful alternatives, the effectiveness of the method can be significantly enhanced. This adaptability ensures that RingE remains current with evolving model architectures.

Example 7: In Fig. 1, the embedding of the path *(Alice, younger brother, Hellen, female cousin, Jane)* ideally exhibits a complementary relation to the embedding of the reverse path *(Jane, mother, Emily, sister's daughter, Alice)*. Similarly, the path embedding for *(Emily, son, Bob)* should be complementary to that of *(Bob, male cousin, Lee, mother, Emily)*, so as the others. □

4.3 DeCon: from Theory to Practice

RingE relies on the input of a ring, thus the problem that follows is how to find the ring. Given the potentially infinite number of rings of varying lengths within a KG, selecting appropriate rings as inputs for the algorithm becomes crucial. To address this challenge, we propose a framework specifically designed to adapt our methods for practical use in both *Static* and *Incremental scenes*.

As illustrated in Fig. 5, our framework comprises two principal modules: *Ring Discovery* and *Conflict Detection*. As detailed in Alg. 1 and Alg. 2, the ring discovery module finds potential conflict-involved rings within the KG. Due to different scene settings, this module is applied differently in static and incremental scenes. We notice that in static scenes, the time complexity is $O(2^{|V|} \cdot (|V|+|E|))$ if we find all rings for each triple, so we implement a ***pruning strategy*** to reduce the complexity in static scenes. This strategy effectively reduces the time complexity to $O(\frac{\zeta}{2\alpha} \cdot (|V|+|E|))$. Here, ζ represents the upper bound of the number of triples appearing in the rings, and α represents the average number of rings that a triple exists in. The conflict detection module, as described in Alg. 3, employs RingE to analyze whether a conflict exists in the

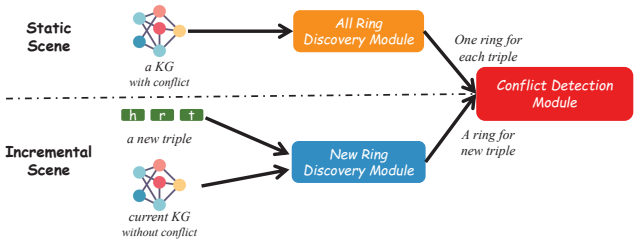


Figure 5: Ring discovery under static and incremental scene.

Algorithm 1: Ring Discovery(Incremental)

Input: T_{KG} :a set of triples from one KG without conflict,
$tri_{new} =< h, t >$:a triple to be added in the above KG,
τ: lower bound of the number of relations in a ring,
ζ: upper bound of the number of relations in a ring.
Output: γ: a ring contain t_{new}.

```
Initialize a stack S with h and t.
while len(stack) ≤ ζ do
    if len(stack) ≥ τ − 1 then
        Find the neighbor entities NE for t.
        if h in NE then
            if len(stack) ≥ τ then
                Break.
        else if h not in NE then
            Do one step DFS and find a new node n'.
    if n' not in S then
        Add a new node to S.
    else if n' is in S then
        if n' is h && len(stack) ≥ τ then
            Find the corresponding triples γ for all the nodes
              in S.
            Break.
return γ.
```

discovered ring. These modules form a framework that detects conflicts within KGs.

4.3.1 ***Ring Discovery.*** When executing the ring discovery algorithm in a directed graph, determining whether there exist rings of relations with the same type $\mathcal{L}(e)$ carries a time complexity of $O(|V| + \frac{|E|}{|\mathcal{L}_E|})$, where $|V|$ is the number of nodes, $|E|$ is the number of edges in the graph, and $|\mathcal{L}_E| << |E|$ is the number of relation types which is a small constant. However, when dealing with dense graphs where it is necessary to check for edges between each pair of vertices, the time complexity can approach $O(|V|^2)$. Then, finding all rings for the above triples containing in rings can lead to a near-exponential number of rings, potentially close to $2^{|V|}$ in the worst case where every node in the KG is potentially connected to every other node. In the average cases, time complexity can be much lower than the worst case, but may still be exponential because the number of rings may grow rapidly. The task of discovering all rings in a KG thus has a time complexity of $O((|V| + \frac{|E|}{|\mathcal{L}_E|}) \cdot 2^{|V|})$, and $O(|V|^2 \cdot 2^{|V|})$ in the dense graph. Thus, even if we categorize the relations and ensure the relations in a ring of the same type, the time complexity is also at an exponential level. Moreover, if the number of relations in the ring is not specified, according to the Lemma 2, even the seemingly simple task of discovering a corresponding ring for each edge can approach a time complexity of $O(|V|^2)$. Additionally, since our conflict detection method relies on the embedding method, the embedding method has small errors itself, so the small errors will continue to accumulate with the length of the path, which may cause misjudgments by the conflict

detection method as the length continues to grow. Hence, limiting the number of relations in a ring becomes imperative as it affects the usability of the method. To address this, we introduce two parameters: τ as the lower bound and ζ as the upper bound of the number of relations in a ring.

Incremental scene. We will first show our design for incremental scenes, which is much simpler. In this scene, we consider a conflict-free KG with a new triple, which will be added. This new triple, not previously existing in the KG, is defined by its head entity h and tail entity t, as outlined in Alg. 1. The algorithm is initialized with h as the starting node and t as the secondary node in a deep first search (DFS) algorithm. According to Lemma 2, we only need to find one ring for this new triple to evaluate whether it will introduce conflicts to the KG or not. So, the goal for the incremental scene is only to discover one ring containing the new triple, and we choose the shortest ring to minimize time consumption. Consequently, if the goal is only to discover the shortest ring, the time complexity is $O(\zeta n)$, in which $n \in [0, |V|]$ is the number of neighbor nodes (i.e., triples) within one hop with t as the head node. $n = |V|$ in a dense graph, all other nodes form a triple with t as the head node, and the time complexity rises up to $O(\zeta|V|)$. That's to say, by Lemma 2, we reduce the time complexity from $O(|V|^2)$ to $O(\zeta|V|)$.

Static scene. As for the static scene, where the goal is to conduct ring discovery for each triple in the KG, the approach differs from the incremental scene. As the incremental scene demonstrates, the worst-case time complexity for ring discovery is $O(\zeta|V|)$. Extending this process to every triple in the KG, the time complexity could rise up to $O(\zeta(|V| + |E|))$, and $O(\zeta|V|^2)$ in the dense graph. We introduce a pruning strategy based on the Lemma 2 to optimize this process. This strategy requires only one ring for each triple to conduct conflict detection effectively. However, a single ring often encompasses multiple triples. Therefore, as Alg. 2 demonstrates, we label these triples (Line 13) to avoid redundant ring discoveries (Lines 3 ~4). With this pruning strategy, the time complexity can be reduced to $O(\frac{\zeta}{2\alpha} \cdot (|V| + |E|))$, where $\alpha \in [1, |V|]$ represents the average number of rings that a triple exists in. Consequently, when $\alpha = |V|$, the time complexity is $O(\frac{\zeta}{2\alpha} \cdot (1 + \frac{|E|}{|V|}))$, but when $\alpha = 1$, the time complexity is $O(\frac{\zeta}{2} \cdot (|V| + |E|))$. Thus, in dense graphs, it is $O(\frac{\zeta}{2}|V|^2)$, and when $\alpha = |V|$, it approximates $O(\frac{\zeta}{2}|V|)$.

Algorithm 2: Ring Discovery(Static)

Input: KG :a knowledge graph with a set of triples $Tri = \{tri_1, tri_2, \dots\}$ with conflict, τ: lower bound of the number of relations in a ring, ζ: upper bound of the number of relations in a ring.
Output: $\Gamma = \{\gamma_1, \gamma_2, \dots\}$: a set of rings that may contain conflicts.

1 Initialize a boolean dict *visited* (the keys are Tri, and the values are *False* at initial), a stack S, and a result list Γ.
2 **for** *each* tri_i *in* T **do**
3 **if** $visited[tri_i]$ *is True* **then**
4 Break.
5 Add the head entity h_i and tail entity t_i to S.
6 **while** $len(stack) \leq \zeta$ **do**
7 Do one step DFS and find a new node n'.
8 **if** n' *not in* S **then**
9 Add a new node to S.
10 **else if** n' *is in* S **then**
11 **if** n' *is* h *and* $len(stack) \geq \tau$ **then**
12 Find the corresponding triples γ for all the nodes in S.
13 Modify the values to $True$ in *visited* for all the triples in γ.
14 Add γ to Γ.
15 Make S and γ empty.
16 Break.
17 **return** Γ.

4.3.2 ***Conflict Detection.*** The task for this module is conflict detection. We employ the embedding and detection model detailed in Sec. 4.2 to detect conflicts within a ring. As shown in Alg. 3, the input is a ring, and the output is whether the ring has conflicts. To analyze a set of rings, Alg. 3 can be repeated as many times as the number of rings.

Algorithm 3: Conflict Detection

Input: γ: a ring.
Output: χ: the result of conflict or not.

1 Put γ in the conflict detection model in Sec. 4.2.
2 Get the result χ of γ.
3 **return** χ.

It is worth mentioning that in the incremental scene, new triples are constantly added. However, since the new triples have the same expression content and form as the existing triples. Consequently, we assume that a data distribution remains consistent despite the ongoing addition of triples, eliminating the need for fine-tuning the RingE model.

5 EVALUATION

Using three real-world graphs, we will next experimentally verify the effectiveness and efficiency of our methods.

5.1 Experiment Setting

Datasets. In this paper, we evaluate DeCon over three real-world datasets: WN18RR [14], NELL-995 [10] and YAGO 3-10 [39]. WN18RR is the subset of WordNet [41], and is widely used in the error detection at triple granularity. The details about the datasets are summarized in Tab. 1. Since there are no labels about conflict in any of the above datasets, inspired by the negative triples construction method in CKRL [50], we get conflict detection datasets with labels about conflict. Specifically, given a ring $(v_1, r_1, v_2, r_2, \dots, v_w, r_w)$ with the same type of relations, in which r_w is the relation from v_w to v_1, we randomly replace one relation or more in the ring with the same type of relation. According to Euq. 3, for a conflict-free ring, $\mathbf{r_1} + \mathbf{r_2} + \cdots + \mathbf{r_w} \leq \epsilon$. So, after replacement, there is a high probability that the result of $\mathbf{r_1} + \mathbf{r_2} + \cdots + \mathbf{r'_{false}} + \cdots + \mathbf{r_w}$ may greater than ϵ.

Table 1: Dataset statistics

Dataset	#Entities	#Relations	#Relation types	#Rings (length from 3 to 5)			#Positive	#Negative
				#Rings (length 3)	#Rings (length 4)	#Rings (length 5)	Rings	Rings
WN18RR	40943	11	1	8862 / 17724	28608 / 57216	28040 / 56080	65510	65510
NELL-995	75492	200	10	2362* / 4724*	12192* / 24384*	70328* / 140656*	84882*	84882*
YAGO 3-10	123182	37	3	40754* / 81508*	33909* / 67818*	36197* / 72394*	110860*	110860*

* The number of rings for YAGO 3-10 is only the number we used in our experiments, in fact, the number is very large.
/ The number before slash indicates the rings without conflict, which comes from the ring discovery. The number after the slash indicates the rings with conflict, which is the output of our conflict construction method.

Thus, we can get a ring with conflict $(v_1, r_1, \ldots, r'_{false}, \ldots, v_w, r_w)$. In other words, for a conflict-free ring, when one or more relations are replaced, the semantics conveyed by the ring will cause errors. As Sec. 4.1 said, the rings with the same type of relations are useful, so in our design, we use the same type of relations to replace them.

Unfortunately, the relations in these three datasets do not have any labels about type, so we need to classify and label the relations first. We use unsupervised clustering methods to classify relations and use the results as classification criteria. However, we have observed that the classification results are not satisfactory. So, we manually classify the relations in NELL-995 and YAGO 3-10 and divide them into several types according to the type of the tail entities that may be connected to the relation. The sub-graphs of some relation types contain too many rings (we did not finish ring discovery for 48 hours), so we only used some of the relation types in our experiments. After classifying and doing ring discovery on these datasets, we summarize the datasets in Tab. 1.

Algorithms for Comparison. Since we are the first work to study conflict detection at path granularity, we compare our methods with the following baseline and state-of-the-art (SOTA) algorithms on error detection in KGs at triple granularity.

(1) TransE [7]: The most classic algorithm for structure-based KG embedding, which uses the idea of $h + r \simeq t$.
(2) DistMult [52]: Another classic knowledge represent learning algorithm for structure-based KG, which concentrates on mining logical rules and capturing relational semantics.
(3) KGTtm [31]: The classic error detection algorithm at triple granularity, which uses both internal semantic information and KG global reasoning information to quantify the semantic correctness of triples.
(4) CAGED [56]: The SOTA error detection algorithm at triple granularity, which is an unsupervised learning mechanism using contrastive learning. CAGED augments a KG into different hyper-views by using each triple as a node to detect errors.

5.2 Implementation

Our goal is to detect the conflict in a ring within the KG. According to the above, only rings with limited length can get accurate results. In practice, we chose 3 as the lower bound and 5 as the upper bound via experiments in Fig. 7(a). That is, the length of our rings in the experiment is $[3, 5]$. And the length of a ring is the number of triples in the ring, which equals the number of relations in this ring. However, it is not the length of the path list, which contains the entities in the path. The length of the path list varies from 6 to 10 when it contains both entities and relations.

We implemented DeCon-simple, DeCon-GRU and DeCon-LSTM in Python using Keras [25], which has already implemented the RNN methods and full connection layer for users. The dimension of the entity and relation embeddings for the input of the model is 100, and we use TransE as the initialization embedding method. Among the models, the batch size is fixed to 50, and we use three RNN modules with 50, 80 and 100 units. The stop conditions for model training are not only determined by the max number of epochs, which is 200, but also are affected by the early stopping method [24] based on the performance of the valid datasets. We use squared hinge loss [33], which performs better in our two-category task. Adam optimizer is commonly used as the parameter optimization. To reduce overfitting, we use the dropout method. Specifically, our inputs of the embedding model contain a relation list, which indicates the ring, so we use the mask and global average pooling to deal with the dynamic length of the relation list.

We apply the same settings to all baseline methods to make the comparison fair. We ran all our experiments on a Linux machine powered by an NVIDIA GeForce RTX 4090 GPU, a 32-core Intel 3.0 GHz CPU with 128 GB of memory. We ran each experiment 10 times and reported the averaged results.

5.3 Experimental results

We next present our findings.

5.3.1 ***Validity of Conflict Detection.*** We evaluate our conflict detection model with the above baseline methods. In order to comprehensively evaluate our proposed model framework, we compare DeCon with simple RNN, GRU, and LSTM, which are all implemented based on Keras. Worth mentioning that the baseline methods are all designed for error detection at triple granularity, we modified r in the input to a relation list R, and using "GlobalAveragePooling1D" in Keras to compress the shape of R to 1.

The evaluation results are presented in Tab. 2. Our model framework can effectively detect conflicts. Our methods perform better than all baselines, and all three methods achieve the best accuracy, higher than 90%, while achieving the highest recall. Our method with LSTM achieves the highest F1-score among all, representing a significant improvement compared to other methods. Among the three network structures, we find out that LSTM shows better

Table 2: Comparison of average Accuracy (Acc) performance among different methods.

Types	Method	WN18RR			NELL-995			YAGO3-10		
		Acc	Recall	F1-score	Acc	Recall	F1-score	Acc	Recall	F1-score
Traditional Methods	TransE [7]	0.883	0.810	0.845	0.900	0.995	0.945	0.881	0.802	0.840
	DistMult [52]	0.500	1.0	0.667	0.500	1.0	0.667	0.500	1.0	0.667
Triple Error Detection Methods	KGTtm [31]	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.500
	CAGED [56]	0.503	0.500	0.501	0.499	0.500	0.499	0.499	0.496	0.497
Our Methods	DeCon-simple	0.661	0.322	0.417	0.706	0.603	0.650	0.800	0.800	0.733
	DeCon-GRU	0.733	0.468	0.567	0.900	0.800	0.800	0.903	0.806	0.811
	DeCon-LSTM	**0.780**	**0.560**	**0.652**	**0.950**	**1.0**	**0.974**	**0.925**	**1.0**	**0.961**

The bolds represent the best performances among our methods.
The underlines represent the best performances among all baselines.

results than GRU, especially over WN18RR and NELL-995. LSTM shows a greater performance improvement compared to GRU and simple RNN, which can be improved by more than 30% (resp. 20%) for WN18RR (resp. NELL-995). Compared with the previous error detection method, DeCon shows better detection capabilities for complex conflicts, and the accuracy exceeds 90% in most cases, with the highest recall of nearly 100%. We also notice that for the cases where all the relations in the rings have the same type, our methods can achieve close to 100% accuracy while achieving a recall of 100%.

The result shows that DistMult cannot fit the demand of error detection at path granularity, tending to predict all test data to be negative which makes the recall as 1. In order to follow the same conflict construction strategy as error detection at triple granularity, which sets the ratio of positive and negative triples to $1:1$, we also let the ratio of positive and negative triples in test data be $1:1$. Thus, the accuracy result in 0.5 and recall result in 1, for all the positive triples also predicted to be negative. We notice that the most classic method, TransE, shows a better performance. That is because our theoretical basis and text embedding method are both based on TransE. Also TransE has good operational characteristics that can effectively capture path semantics. The performance of TransE is close to our detection model when following our methods because we replace the simple RNN module in the framework with TransE directly in the experiment. TransE performs best among all the baselines. Unfortunately, other error detection methods at triple granularity, which are designed for triples, show low accuracy close to 0.5. That's because they tend to predict all test data to be positive, which makes the recall 0.5.

To demonstrate the capabilities of DeCon, we run all variable-parameter experiments using DeCon-LSTM, as shown below. The results on other versions are the same, and the results are in B.1.

Varying model parameters. We also explored different model parameter selections to study the effects of the parameter selection on the performance of RingE. We selected 3 additional sets of parameters to make comparisons, respectively set as $50-50-50$, $30-60-90$, and $100-100-100$, which have the same order of magnitude as the embedding dimension of TransE. As shown in Fig. 11, our selections of parameters can meet the best among them. That's because the different numbers of units among different RNN modules can make the output different, and the irregular number of units selected can make the difference more obvious.

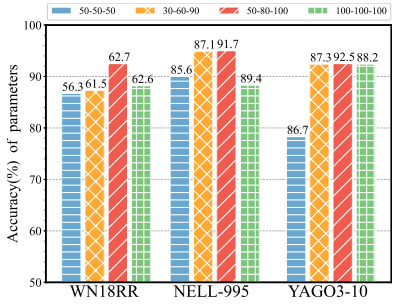
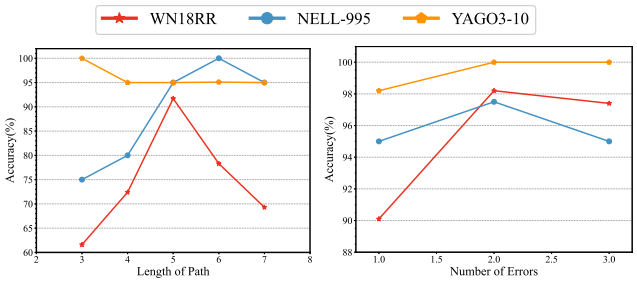


Figure 6: Effective results with the different number of units.

Varying path length. We evaluated the effect of the path length based on accuracy, as shown in Fig. 7(a). The path length indicates the number of relations in the path. We experimented with lengths from 3 to 7, respectively, and fixed the number of conflict triples for each path, which is 1 in the experiment. The result shows that, within a reasonable length range, the longer the path is, the more evidence there is and the higher the accuracy of the detection. When the path length exceeds this range, the accuracy begins to decrease due to the small errors cumulative effect mentioned in Sec. 4.3.1. As Tab. 1 shows, the reasonable path length range is $[3,5]$ that we use in our experiments. The decreasing point in most cases is 6. In the other cases, even though the results of 6 are increased compared to 5, the model accuracy on three datasets begins to decrease at 7. Thus, we set the length range as $[3,5]$ to fit all cases. But for length 3, we notice that the accuracy is much smaller than that of lengths 4 and 5. That's because of the datasets. As Tab. 1 shows, the number of paths of length 3 is much smaller compared with those of lengths 4 and 5, so the models based on length 3 may not fit well enough.



(a) Varying Length (b) Varying #Error

Figure 7: DeCon: Effectiveness. (a) shows the effect of the length of the path while the number of error triples is fixed. (b) shows the effect of the number of error triples in the path, while the length is from 3 to 7.

In order to minimize the impact of manually injecting conflicts to construct the datasets, the datasets we used to test different path lengths were extracted from the overall experiment according to the length of the path. But this does not prevent us from drawing the above conclusions.

Varying error rate. We next evaluated the effect of the number of errors in the path and the results are shown in Fig. 7(b). We still used all paths with lengths 3 to 5, but we increased the number of errors for them. Based on the original setting, which includes only a single error, we modified 2 triples and 3 triples to construct the triple conflict and conducted the experiment. It is worth noting that, since the length of the path is 3, the number of error triples is at most 3 under our experimental setting. The results show that the more error triples exist in the path, the more obvious the conflict will be and the easier it will be detected. However, since the length of the path we used is 3 to 5 when the conflict is 3, a *pseudo-replacement* is performed for the path with length 3. After replacement, there is a probability that the path is conflict-free, so it will mislead the model. Thus, when the number of conflict injections is 3, the accuracy decreases slightly.

5.3.2 ***System Efficiency.*** We then evaluated the efficiency of the system in the static scene. As we mentioned above, YAGO 3-10 has a large amount of rings that cannot be collected within 2 days. So YAGO 3-10 has enough rings for us to evaluate in this part, and we use the sub-graphs of YAGO 3-10 to evaluate the efficiency under different graph sizes. We used a sample strategy to get the sub-graphs, in which we randomly chose a set of entities in YAGO 3-10 following the sample ratios, and then we did a breadth-first search (BFS) for all of these entities to get all the 3 hops entities and relations among them. Through pre-experiments, we find that when the sampling rate is 20% with 50,000 entities and 220,000 triples, and mining rings with a length of 3 to 5, the pruning strategy consumes 3.5 hours discovering 1,640,000 rings, while the program doesn't end within 48 hours without pruning.

So in our experiments, we set the sample ratio as 1%, 5%, 10%, 15%, and 20%, the number of entities and triples in these sub-graphs are summarized in Tab. 3. We evaluate the efficiency with the lengths of $[3,4]$, $[4,5]$ and $[5,6]$ as shown in Fig. 8. It's worth stating that for lengths $[5,6]$, only 1% without pruning can get the result within

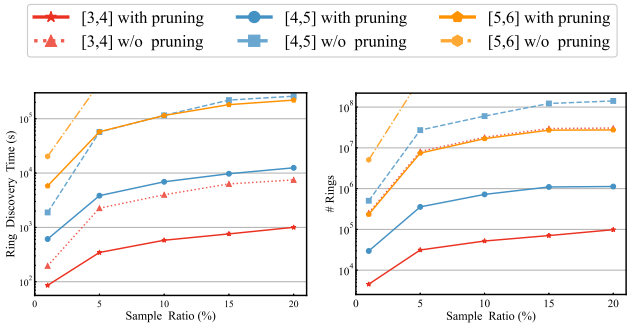
Table 3: The statistics of sample result of YAGO 3-10.

Sample Ratio(%)	# Entities	# Triples
1	13999	32917
5	28899	89309
10	39383	139399
15	47174	187602
20	53872	229133

48 hours, so we add 1% to our results. We compare DeCon with pruning and without pruning to demonstrate the effectiveness of our pruning strategy. The efficiency improvement on NELL-995 is similar to that of YAGO 3-10. Pre-experiments with the sample ratio as 1% and the length as $[3,4]$ only output 3 rings on WN18RR, and for length $[3,5]$, there are only 60,000, which doesn't have enough rings for this experiment. For results in NELL-995, please refer to B.2

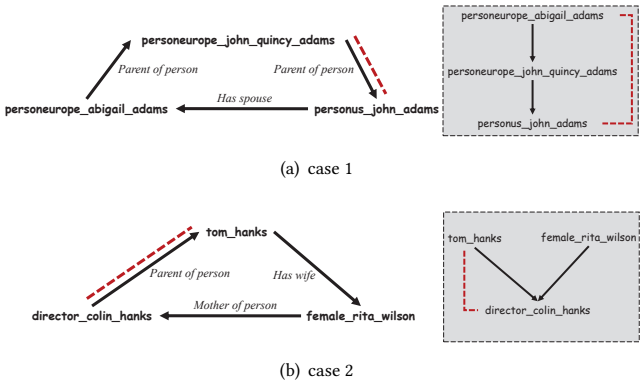
The results in Fig. 8(a) and 8(b) show our pruning strategy can greatly reduce the time consumption of ring discovery, and then, can greatly reduce the number of rings for subsequent conflict detection task. It is worth mentioning that since our pruning strategy for ring discovery is specific to our tasks and based on our derived Lemma, we only study ring discovery methods that find all rings.

In Fig. 8, we notice that there are some nodes out of figures because we cannot complete ring discovery within 48 hours, which means the actual time consumption and the number of rings may be very large. The results of ring discovery without pruning are all above that of discovering with pruning. The time consumption can be improved by up to 85% for $[3,4]$, up to 92% for $[4,5]$, close even up to 100% for $[5,6]$ of sample ratio 5%, 10%, 15% and 20%.



(a) Time consumption. (b) The number of rings.

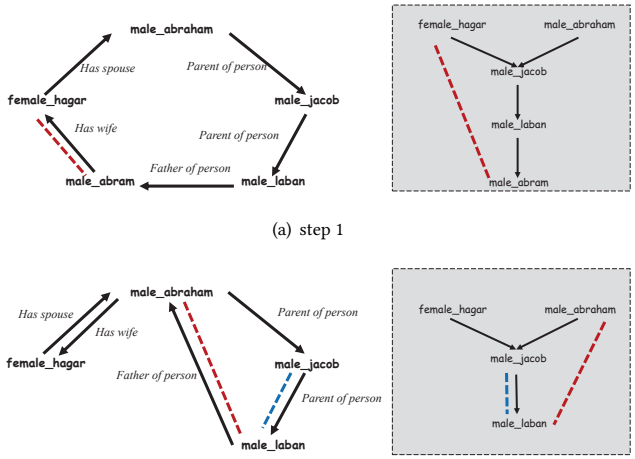
Figure 8: Efficiency Results of Ring Discovery. (a) shows the results about time consumption. (b) shows the results about the number of rings. The results out of the figures are those who cannot get the result within 48 hours. The same color with different shades is for the same length, the dark color represents with pruning, and the light color represents w/o pruning. All solids are the results with pruning.



(a) case 1

(b) case 2

Figure 9: Case 1 and 2. The left graphs represent sub-graphs in NELL-995, and kinship trees are on the right.



(a) step 1

(b) step 2 and 3

Figure 10: Case 3. Red (resp. blue) indicates step 2 (resp. 3).

The higher the sample ratio, the more time consumption reduction. And the longer the ring, the more time consumption reduction. The results can meet our analysis in Sec. 4.3. The results show that the pruning strategy based on Lemma 2 can significantly reduce time cost. Since fewer rings need to be detected, DeCon can certainly further reduce the conflict detection time in the static scene.

5.4 Case Study

The above experiments have shown that RingE performs well for conflict detection, and DeCon has a strong ability to detect conflicts at path granularity. We next conducted the case study to further demonstrate the ability of RingE and DeCon in detecting conflicts at path granularity in the real world. We gave examples of detecting conflicts on NELL-995 as Fig. 9 and 10 shown. Note that these conflicts can not be detected by any baselines.

In order to make the kinship clearer, we rewrite it as a kinship tree on the right of the figures. Before analyzing, we need to declare "parent" refers to other elders besides mother and father in NELL-995. We have observed semantic conflicts in the relation paths, leading to the path being detected as having conflict. In Fig. 9, the conflicts in the two cases share the same form and are both caused by the misdirection of the edge (denoted relation in the triple). Our method can catch the semantic conflict that a person's children cannot be his or her parents. Because the data in NELL-995 are all from the real world web, we also searched the web to confirm the results. For case 1 in Fig. 9(a), we confirm that "personeurope_abigail_adams" and "personus_john_adams" are couples, and are parents of "personeurope_john_quincy_adams". Thus, the relation between "personeurope_john_quincy_adams" and "personus_john_adams" is incorrect, which causes the conflict, and the relation must be reversed. Similarly for case 2 in Fig. 9(b), we confirm that "tom_hanks" and "female_rita_wilson" are couples, and "director_colin_hanks" is their child. So, the relation between "director_colin_hanks" and "tom_hanks" must also be reversed., as mentioned in Sec. 1.

In Fig. 10, the conflict is more complex, and we will explain it step by step. First, our method catches the semantic conflict in the ring, and we notice that "female_hagar" has two husbands: "male_abraham" and "male_abram". According to the kinship tree, there is a three-generation gap between the two people, so there has an error. Then, we confirm that "male_abraham" and "male_abram" refers to the same person who changed his name. Thus, we aligned these two entities as Fig. 10(b) shows. Afterward, our method still caught a conflict in the newly generated rings that the parent of the parent of the person cannot be the child of the person. Furthermore, we confirm that the relation between "male_abraham" and "male_laban" must also be reversed. Moreover, we notice that "male_jacob" is the grandson of "male_abraham", and "male_laban" is the maternal uncle of "male_jacob". Thus, the relation between "male_jacob" and "male_laban" must also be reversed.

The above conflicts are all difficult to detect at triple granularity while can be easily detected at path granularity.

6 CONCLUSION

In this paper, to eliminate the problem of semantic inconsistency in multi-triple concatenation, we discuss path-granularity conflicts in KG. We theoretically prove that these conflicts are associated with rings. Furthermore, we simplify the complexity of conflict detection, and propose a ring embedding method RingE to represent ring. We also design DeCon, a conflict detection framework based on RingE. This conflict detection framework on KGs is applied to both static and incremental scenes respectively. Based on theoretical proof, we propose a pruning strategy to reduce the time consumption. Extensive evaluation with three real-world datasets shows that RingE can accomplish the conflict detection task well, and the pruning strategy in DeCon can exponentially reduce the time consumption and the number of rings used for detection. Future work includes model optimization for detection tasks and employing complex language models for ring representation.

REFERENCES

[1] [n.d.]. Full version. https://yolanda-w98.github.io/edbt2025full.pdf.
[2] 2024. [n.d.]. Wikipedia. http://wikipedia.org/.
[3] Bo An, Bo Chen, Xianpei Han, and Le Sun. 2018. Accurate Text-Enhanced Knowledge Graph Representation Learning. In *NAACL*. 745–755.
[4] Arvind Arasu, Michaela Götz, and Raghav Kaushik. 2010. On active learning of record matching packages. In *ACM SIGMOD*. 783–794.
[5] Arvind Arasu, Christopher Ré, and Dan Suciu. 2009. Large-scale deduplication with constraints using dedupalog. In *IEEE ICDE*. 952–963.
[6] Akari Asai, Sewon Min, Zexuan Zhong, and Danqi Chen. 2023. Retrieval-based language models and applications. In *ACL*. 41–46.
[7] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. *NeurIPS* 26 (2013).
[8] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. *NIPS* 26 (2013).
[9] Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. 2013. Translating Embeddings for Modeling Multi-relational Data. In *NIPS*.
[10] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam Hruschka, and Tom Mitchell. 2010. Toward an architecture for never-ending language learning. In *AAAI*, Vol. 24. 1306–1313.
[11] Kewei Cheng, Xian Li, Yifan Ethan Xu, Xin Luna Dong, and Yizhou Sun. 2022. PGE: Robust Product Graph Embedding Learning for Error Detection. *VLDB* 15, 6 (2022), 1288–1296.
[12] Yurong Cheng, Lei Chen, Ye Yuan, Guoren Wang, Boyang Li, and Fusheng Jin. 2022. Strict and Flexible Rule-Based Graph Repairing. *TKDE* 34, 7 (2022), 3521–3535.
[13] Xu Chu, Ihab F Ilyas, and Paraschos Koutris. 2016. Distributed data deduplication. *VLDB* 9, 11 (2016), 864–875.
[14] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. 2018. Convolutional 2d knowledge graph embeddings. In *AAAI*, Vol. 32.
[15] Junnan Dong, Qinggang Zhang, Xiao Huang, Qiaoyu Tan, Daochen Zha, and Zhao Zihao. 2023. Active ensemble learning for knowledge graph error detection. In *ACM WSDM*. 877–885.
[16] Wenfei Fan, Floris Geerts, Nan Tang, and Wenyuan Yu. 2014. Conflict resolution with data currency and consistency. *JDIQ* 5, 1-2 (2014), 1–37.
[17] Wenfei Fan, Ruochun Jin, Muyang Liu, Ping Lu, Chao Tian, and Jingren Zhou. 2020. Capturing associations in graphs. *VLDB* 13, 12 (2020), 1863–1876.
[18] Wenfei Fan, Xueli Liu, Ping Lu, and Chao Tian. 2018. Catching numeric inconsistencies in graphs. In *ACM TODS*. 381–393.
[19] Wenfei Fan, Ping Lu, Chao Tian, and Jingren Zhou. 2019. Deducing certain fixes to graphs. *VLDB* 12, 7 (2019), 752–765.
[20] Wenfei Fan, Chao Tian, Yanghao Wang, and Qiang Yin. 2021. Parallel discrepancy detection and incremental detection. *VLDB* 14, 8 (2021), 1351–1364.
[21] Wenfei Fan, Yinghui Wu, and Jingbo Xu. 2016. Functional dependencies for graphs. In *ACM SIGMOD*. 1843–1857.
[22] Dieter Fensel, Umutcan Şimşek, Kevin Angele, Elwin Huaman, Elias Kärle, Oleksandra Panasiuk, Ioan Toma, Jürgen Umbrich, Alexander Wahler, Dieter Fensel, et al. 2020. Introduction: what is a knowledge graph? *Knowledge graphs: Methodology, tools and selected use cases* (2020), 1–10.
[23] Google. 2012. Introducing the Knowledge Graph: things, not strings. https://blog.google/products/search/introducing-knowledge-graph-things-not/. Access: 2024-10.
[24] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. Speech recognition with deep recurrent neural networks. In *IEEE ICASSP*. Ieee, 6645–6649.
[25] Antonio Gulli and Sujit Pal. 2017. *Deep learning with Keras*. Packt Publishing Ltd.
[26] Huihui Han, Jian Wang, Xiaowen Wang, and Sen Chen. 2022. Construction and Evolution of Fault Diagnosis Knowledge Graph in Industrial Process. *TIM* 71 (2022), 1–12.
[27] Yacheng He, Qianghuai Jia, Lin Yuan, Ruopeng Li, Yixin Ou, and Ningyu Zhang. 2023. A Concept Knowledge Graph for User Next Intent Prediction at Alipay. *WWW* (2023).
[28] Alireza Heidari, Joshua McGrath, Ihab F Ilyas, and Theodoros Rekatsinas. 2019. Holodetect: Few-shot learning for error detection. In *ACM SIGMOD*. 829–846.
[29] Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Knowledge graph embedding via dynamic mapping matrix. In *ACL/IJCNLP*. 687–696.
[30] Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and S Yu Philip. 2021. A survey on knowledge graphs: Representation, acquisition, and applications. *TNNLS* 33, 2 (2021), 494–514.
[31] Shengbin Jia, Yang Xiang, Xiaojun Chen, and Kun Wang. 2019. Triple trustworthiness measurement for knowledge graph. In *ACM WWW*. 2865–2871.
[32] Seyed Mehran Kazemi and David Poole. 2018. Simple embedding for link prediction in knowledge graphs. *NIPS* 31 (2018).
[33] Ching-Pei Lee and Chih-Jen Lin. 2013. A study on L2-loss (squared hinge-loss) multiclass SVM. *Neural computation* 25, 5 (2013), 1302–1323.
[34] Yankai Lin, Zhiyuan Liu, Huanbo Luan, Maosong Sun, Siwei Rao, and Song Liu. 2015. Modeling Relation Paths for Representation Learning of Knowledge Bases. In *EMNLP*. 705–714.
[35] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. 2015. Learning entity and relation embeddings for knowledge graph completion. In *AAAI*, Vol. 29.
[36] Listlink. [n.d.]. Exploring Key Data Science Trends for 2024: Insights for the Year Ahead. https://litslink.com/blog/data-science-trends#data-science-trend-2-the-rise-of-graph-technology-and-analytical. 2024.
[37] Haiying Liu, Ruizhe Ma, Daiyi Li, Li Yan, and Zongmin Ma. 2021. Machinery fault diagnosis based on deep learning for time series analysis and knowledge graphs. *Journal of Signal Processing Systems* 93 (2021), 1433–1455.
[38] Mohammad Mahdavi, Ziawasch Abedjan, Raul Castro Fernandez, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2019. Raha: A configuration-free error detection system. In *ACM SIGMOD*. 865–882.
[39] Farzaneh Mahdisoltani, Joanna Biega, and Fabian M Suchanek. 2013. Yago3: A knowledge base from multilingual wikipedias. In *CIDR*.
[40] Lars-Peter Meyer, Claus Stadler, Johannes Frey, Norman Radtke, Kurt Junghanns, Roy Meissner, Gordian Dziwis, Kirill Bulert, and Michael Martin. 2023. Llm-assisted knowledge graph engineering: Experiments with chatgpt. *arXiv* (2023).
[41] George A Miller. 1995. WordNet: a lexical database for English. *CACM* 38, 11 (1995), 39–41.
[42] Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. 2023. Unifying Large Language Models and Knowledge Graphs: A Roadmap. *arXiv* (2023).
[43] Yuhan Sun, Jia Yu, and Mohamed Sarwat. 2019. Demonstrating spindra: A geographic knowledge graph management system. In *IEEE ICDE*. 2044–2047.
[44] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. 2019. Rotate: Knowledge graph embedding by relational rotation in complex space. *arXiv* (2019).
[45] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2016. Complex embeddings for simple link prediction. In *ACM ICML*. PMLR, 2071–2080.
[46] Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Commun. ACM* 57, 10 (Sept. 2014), 78–85.
[47] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge graph embedding by translating on hyperplanes. In *AAAI*, Vol. 28.
[48] Han Xiao, Minlie Huang, Lian Meng, and Xiaoyan Zhu. 2017. SSP: semantic space projection for knowledge graph embedding with text descriptions. In *AAAI*, Vol. 31.
[49] Ruobing Xie, Zhiyuan Liu, Jia Jia, Huanbo Luan, and Maosong Sun. 2016. Representation learning of knowledge graphs with entity descriptions. In *AAAI*, Vol. 30.
[50] Ruobing Xie, Zhiyuan Liu, Fen Lin, and Leyu Lin. 2018. Does William Shakespeare REALLY Write Hamlet? Knowledge Representation Learning with Confidence. In *AAAI/IAAI/EAAI*. AAAI Press, Article 607, 8 pages.
[51] Bingcong Xue and Lei Zou. 2022. Knowledge graph quality management: a comprehensive survey. *IEEE TKDE* 35, 5 (2022), 4969–4988.
[52] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2014. Embedding entities and relations for learning and inference in knowledge bases. *arXiv* (2014).
[53] Sheng Yu, Zheng Yuan, Jun Xia, Shengxuan Luo, Huaiyuan Ying, Sihang Zeng, Jingyi Ren, Hongyi Yuan, Zhengyun Zhao, Yucong Lin, et al. 2022. Bios: An algorithmically generated biomedical knowledge graph. *arXiv* (2022).
[54] Amrapali Zaveri, Dimitris Kontokostas, Sebastian Hellmann, Jürgen Umbrich, Michael Färber, Frederic Bartscherer, Carsten Menne, Achim Rettinger, Amrapali Zaveri, Dimitris Kontokostas, Sebastian Hellmann, and Jürgen Umbrich. 2018. Linked Data Quality of DBpedia, Freebase, OpenCyc, Wikidata, and YAGO. *Semant. Web* 9, 1 (2018), 77–129.
[55] Qinggang Zhang, Junnan Dong, Keyu Duan, Xiao Huang, Yezi Liu, and Linchuan Xu. 2022. Contrastive knowledge graph error detection. In *ACM CIKM*. 2590–2599.
[56] Qinggang Zhang, Junnan Dong, Keyu Duan, Xiao Huang, Yezi Liu, and Linchuan Xu. 2022. Contrastive knowledge graph error detection. In *ACM CIKM*. 2590–2599.
[57] Zhengyan Zhang, Zhiyuan Zeng, Yankai Lin, Huadong Wang, Deming Ye, Chaojun Xiao, Xu Han, Zhiyuan Liu, Peng Li, Maosong Sun, et al. 2023. Plug-and-play knowledge injection for pre-trained language models. *arXiv* (2023).
[58] Yu Zhao and Ji Liu. 2019. Scef: A support-confidence-aware embedding framework for knowledge graph refinement. *arXiv* (2019).
[59] Da Zheng, Xiang Song, Chao Ma, Zeyuan Tan, Zihao Ye, Jin Dong, Hao Xiong, Zheng Zhang, and George Karypis. 2020. Dgl-ke: Training knowledge graph embeddings at scale. In *ACM SIGIR*. 739–748.
[60] Huaping Zhong, Jianwen Zhang, Zhen Wang, Hai Wan, and Zheng Chen. 2015. Aligning knowledge and text embeddings by entity descriptions. In *EMNLP*. 267–272.

A DETIALED PROOF

A.1 Proof of Insight 1

A pair of paths $(p^u_{h\to t}, p^v_{t\to h})$, connecting head entity h and tail entity t can be conceptualized as a directed graph. In this graph, nodes correspond to entities, while directed edges represent relations. Consequently, each path can be expressed as an ordered sequence of nodes.

Therefore, the path from h to t can be represented as $E = (e_1, e_2, \ldots, e_g)$, while the path from t to h is $E' = (e'_1, e'_2, \ldots, e'_q)$. Thus, we can represent the pair of paths as follows:

$$< (h, e_1, e_2, \ldots, e_g, t), (t, e'_1, e'_2, \ldots, e'_q, h) > .$$

- If $\forall f \in (e_1, e_2, \ldots, e_g), f \notin (e'_1, e'_2, \ldots, e'_q)$, this situation can be abstracted as illustrated in Fig. 2(a). Consider any node k within the set of nodes N. It is guaranteed that there exists a path from node k to entity t. Moreover, path E' represents the path from entity t to entity h. Therefore, there must exist a path from node k to entity h. Finally, since k is a node in E, it is evident that there exists a path from entity h to node k. Consequently, there must be a path from node k to itself, forming a ring structure. In summary, the presence of a ring structure is confirmed in this case.
- If $\exists f \in (e_1, e_2, \ldots, e_g), f \in (e'_1, e'_2, \ldots, e'_q)$, which can be abstracted as depicted in Fig. 2(b) and Fig. 2(c). In this case, let the set of nodes satisfying these conditions be represented as $F = (f_1, f_2, \ldots, f_j)$, maintaining the order in the path. For E, $F = (f_1, f_2, \ldots, f_j)$, but for E', $F' = (f_j, f_{j-1}, \ldots, f_1)$. Then E and E' can be divided into $(j+1)$ segments by the nodes in F, expressed as

$$E = (e_1, \ldots, f_1, \ldots, f_2, \ldots, f_j, \ldots, e_g).$$

$$E' = (e'_1, \ldots, f_j, \ldots, f_{j-1}, \ldots, f_1, \ldots, e'_q).$$

  - For the segment containing h, there exists a path from h to f_1 within path E, and there is a path from f_1 to h within path E'. As a result, a ring structure is established around the head entity h.
  - For the segment involving t, there exists a path from $f_j \to t$ within path E, and a path from t to f_j within path E'. This forms a ring structure around the tail entity t.
  - For the remaining segments, arbitrarily chosen $f_s \in (f_2, \ldots, f_{j-1})$. There is a path connecting f_s to f_{s+1} within path E, and a path from f_{s+1} to f_s within path E'. Consequently, ring structures are established involving these intermediate nodes.

To sum up, there must be several rings together to form a pair of paths $(p^u_{h\to t}, p^v_{t\to h})$ between h and t.

A.2 Proof of Lemma 2

Given an edge $R_{A\to B}$ in a KG, find two distinct paths from B to A in the KG, represented as $p^1_{B\to A}, p^2_{B\to A}$. The proofs for the Lemma from the triple and TransE below.

Triple perspective. The proof for the Lemma starts from the triple below, which means we use the triple to represent the edge in the KG. That's to say, $R_{A\to B} = Ar_{AB}B$,

$$p^1_{B\to A} = (Br^1_1 n^1_1, n^1_1 r^1_2 n^1_2, \ldots, n^1_i r^1_{i+1} A),$$
$$p^2_{B\to A} = (Br^2_1 n^2_1, n^2_1 r^2_2 n^2_2, \ldots, n^2_j r^2_{j+1} A).$$

Ideally, vector operations are performed. if there is a conflict in the ring $\overrightarrow{\mathbb{R}}^1_{A\to B\to A}$ formed by $R_{A\to B}$ and $p^1_{B\to A}$, as the shown in Equ. 3, then

$$\overrightarrow{\mathbb{R}}^1_{A\to B\to A} = R_{A\to B} + p^1_{B\to A} =$$

$$Br^1_1 n^1_1 + n^1_1 r^1_2 n^1_2 + \cdots + n^1_i r^1_{i+1} A + Ar_{AB}B > \epsilon,$$

According to the *Property* in Sec. 4.1, there is

$$Br^1_1 n^1_1 + n^1_1 r^1_2 n^1_2 + \cdots + n^1_i r^1_{i+1} A =$$

$$Br^2_1 n^2_1 + n^2_1 r^2_2 n^2_2 + \cdots + n^2_j r^2_{j+1} A,$$

and then

$$\overrightarrow{\mathbb{R}}^2_{A\to B\to A} = R_{A\to B} + p^2_{B\to A} = Br^2_1 n^2_1 + n^2_1 r^2_2 n^2_2 + \cdots + n^2_j r^2_{j+1} A + Ar_{AB}B =$$

$$Br^1_1 n^1_1 + n^1_1 r^1_2 n^1_2 + \cdots + n^1_i r^1_{i+1} A + +Ar_{AB}B = R_{A\to B} + p^1_{B\to A} = \overrightarrow{\mathbb{R}}^1_{A\to B\to A} > \epsilon.$$

That is to say, the ring $\overrightarrow{\mathbb{R}}^2_{A\to B\to A}$ also has conflict.

In the same way, if there is no conflict in the ring $\overrightarrow{\mathbb{R}}^1_{A\to B\to A}$, then

$$\overrightarrow{\mathbb{R}}^1_{A\to B\to A} = R_{A\to B} + p^1_{B\to A} =$$

$$Br^1_1 n^1_1 + n^1_1 r^1_2 n^1_2 + \cdots + n^1_i r^1_{i+1} A + Ar_{AB}B \le \epsilon,$$

According to the *Property* in Sec. 4.1, there is

$$Br^1_1 n^1_1 + n^1_1 r^1_2 n^1_2 + \cdots + n^1_i r^1_{i+1} A =$$

$$Br^2_1 n^2_1 + n^2_1 r^2_2 n^2_2 + \cdots + n^2_j r^2_{j+1} A,$$

and then

$$\overrightarrow{\mathbb{R}}^2_{A\to B\to A} = R_{A\to B} + p^2_{B\to A} = Br^2_1 n^2_1 + n^2_1 r^2_2 n^2_2 + \cdots + n^2_j r^2_{j+1} A + Ar_{AB}B =$$

$$Br^1_1 n^1_1 + n^1_1 r^1_2 n^1_2 + \cdots + n^1_i r^1_{i+1} A + +Ar_{AB}B = R_{A\to B} + p^1_{B\to A} = \overrightarrow{\mathbb{R}}^1_{A\to B\to A} \le \epsilon.$$

That is to say, the ring $\overrightarrow{\mathbb{R}}^2_{A\to B\to A}$ also has no conflict.

TransE **with triple perspective.** The proof for the Lemma starts from the TransE below, which means we use TransE to convert the relation in the triple. In relation perspective without TransE, the paths are $R_{A\to B} = Ar_{AB}B$, $p^1_{B\to A} = (Br^1_1 n^1_1, n^1_1 r^1_2 n^1_2, \ldots, n^1_i r^1_{i+1} A)$, and $p^2_{B\to A} = (Br^2_1 n^2_1, n^2_1 r^2_2 n^2_2, \ldots, n^2_j r^2_{j+1} A)$. As the idea of TransE, if $h + r \simeq t$, the paths with relations can be rewrited as $R_{A\to B} = 2B$,

$$\overrightarrow{\mathbb{R}}^1_{A\to B\to A} = p^1_{B\to A} = (2n^1_1, 2n^1_2, \ldots, 2A),$$

$$p^2_{B\to A} = (2n^2_1, 2n^2_2, \ldots, 2A)$$

Ideally, vector operations are performed. If there is a conflict in the ring structure formed by $R_{A\to B}$ and $p^1_{B\to A}$, then

$$R_{A\to B} + p^1_{B\to A} =$$

$$2n^1_1 + 2n^1_2 + \cdots + 2A + 2B > \epsilon.$$

According to the *Property* in Sec. 4.1, there is

$$2n^1_1 + 2n^1_2 + \cdots + 2A =$$

$$2n^2_1 + 2n^2_2 + \cdots + 2A,$$

and then

$$\overrightarrow{\mathbb{R}}^2_{A\to B\to A} = R_{A\to B} + p^2_{B\to A} = 2n^2_1 + 2n^2_2 + \cdots + 2A + 2B$$

$$= 2n^1_1 + 2n^1_2 + \cdots + 2A = R_{A\to B} + p^1_{B\to A} = \overrightarrow{\mathbb{R}}^1_{A\to B\to A} > \epsilon.$$

That is to say, the ring $\overrightarrow{\mathbb{R}}^2_{A\to B\to A}$ also has conflict.

In the same way, if there is no conflict in the ring structure formed by $R_{A\to B}$ and $p^1_{B\to A}$, then

$$\overrightarrow{\mathbb{R}}^1_{A\to B\to A} = R_{A\to B} + p^1_{B\to A} =$$

$$2n^1_1 + 2n^1_2 + \cdots + 2A + 2B \le \epsilon,$$

According to the above nature, there is

$$2n^1_1 + 2n^1_2 + \cdots + 2A =$$

$$2n^2_1 + 2n^2_2 + \cdots + 2A,$$

and then

$$\overrightarrow{\mathbb{R}}^2_{A\to B\to A} = R_{A\to B} + p^2_{B\to A} = 2n^2_1 + 2n^2_2 + \cdots + 2A + 2B$$

$$= 2n^1_1 + 2n^1_2 + \cdots + 2A = R_{A\to B} + p^1_{B\to A} = \overrightarrow{\mathbb{R}}^1_{A\to B\to A} \le \epsilon.$$

That is to say, the ring $\overrightarrow{\mathbb{R}}^2_{A\to B\to A}$ also has no conflict.

TransE **with relation perspective.** The proof for the Lemma starts from the TransE below, which means we use TransE to convert the relation in the triple. In relation perspective without TransE, the paths are $R_{A\to B} = (r_{AB})$, $p^1_{B\to A} = (r^1_1, r^1_2, \ldots, r^1_{i+1})$ and $p^2_{B\to A} = (r^2_1, r^2_2, \ldots, r^2_{j+1})$. As the idea of TransE, if $h + r \simeq t$, the paths with relations can be rewrited as $R_{A\to B} = (A - B)$

$$p^1_{B\to A} = (B - n^1_1, n^1_1 - n^1_2, \ldots, n^1_i - A),$$

$$p^2_{B\to A} = (B - n^2_1, n^2_1 - n^2_2, \ldots, n^2_j - A)$$

Ideally, vector operations are performed. If there is a conflict in the ring $\overrightarrow{\mathbb{R}}^1_{A\to B\to A}$ formed by $R_{A\to B}$ and $p^1_{B\to A}$, then

$$\overrightarrow{\mathbb{R}}^1_{A\to B\to A} = R_{A\to B} + p^1_{B\to A} =$$

$$(B - n^1_1) + (n^1_1 - n^1_2) + \cdots + (n^1_i - A) + (A - B) > \epsilon,$$

According to the *Property* in Sec. 4.1, there is

$$(B - n^1_1) + (n^1_1 - n^1_2) + \cdots + (n^1_i - A) =$$

$$(B - n^2_1) + (n^2_1 - n^2_2) + \cdots + (n^2_j - A),$$

and then

$$\overrightarrow{\mathbb{R}}^2_{A\to B\to A} = R_{A\to B} + p^2_{B\to A} = (B - n^2_1) + (n^2_1 - n^2_2) + \cdots + (n^2_j - A) + (A - B) =$$

$$(B - n^1_1) + (n^1_1 - n^1_2) + \cdots + (n^1_i - A) + (A - B) = R_{A\to B} + p^1_{B\to A} = \overrightarrow{\mathbb{R}}^1_{A\to B\to A} > \epsilon.$$

That is to say, the ring $\overrightarrow{\mathbb{R}}^2_{A\to B\to A}$ also has conflict.

In the same way, if there is no conflict in the ring $\overrightarrow{\mathbb{R}}^1_{A\to B\to A}$, then

$$\overrightarrow{\mathbb{R}}^1_{A\to B\to A} = R_{A\to B} + p^1_{B\to A} =$$

$$(B - n^1_1) + (n^1_1 - n^1_2) + \cdots + (n^1_i - A) + (A - B) \le \epsilon,$$

according to the *Property* in Sec. 4.1, there is

$$Br^1_1 n^1_1 + n^1_1 r^1_2 n^1_2 + \cdots + n^1_i r^1_{i+1} A =$$

$$Br^2_1 n^2_1 + n^2_1 r^2_2 n^2_2 + \cdots + n^2_i r^2_{i+1} A,$$

and then

$$\overrightarrow{\mathbb{R}}^2_{A\to B\to A} = R_{A\to B} + p^2_{B\to A} = (B - n^2_1) + (n^2_1 - n^2_2) + \cdots + (n^2_j - A) + (A - B) =$$

$$(B - n^1_1) + (n^1_1 - n^1_2) + \cdots + (n^1_i - A) + (A - B) = R_{A\to B} + p^1_{B\to A} \le \epsilon.$$

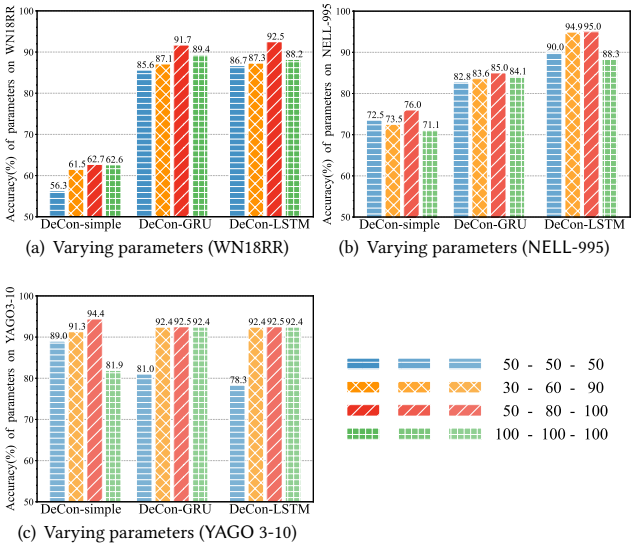
That is to say, the ring $\overrightarrow{\mathbb{R}}^2_{A\to B\to A}$ also has no conflict.

To sum up, the Lemma 2 holds under both triple and relation perspectives, and holds under both common and TransE perspectives.

B ADDITIONAL EXPERIMENT RESULTS

B.1 Variable-parameter Experiments

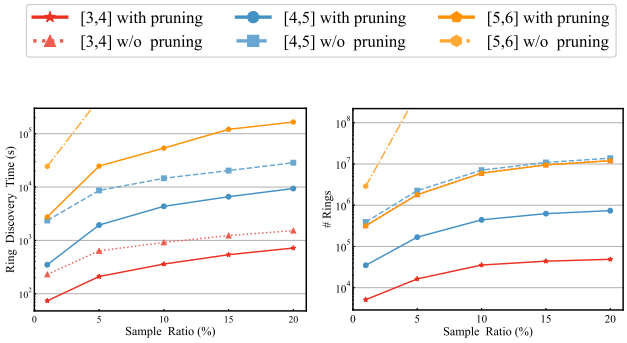
In order to make the presentation of our experiment results clearer, We organized the results by different datasets instead of by different models, as shown in Fig. 11. The results with simple RNN and GRU models show the same results as LSTM model, that our selections of parameters can meet the best among them. That's because the different numbers of units among different RNN modules can make the output different, and the irregular number of units selected can make the difference more obvious.



(a) Varying parameters (WN18RR) (b) Varying parameters (NELL-995)

(c) Varying parameters (YAGO 3-10)

Figure 11: The impact of different numbers of units.



(a) Time consumption. (b) The number of rings.

Figure 12: Results of Ring Discovery with and without pruning on NELL-995. (a) shows the time consumption with and w/o pruning on NELL-995. (b) shows the number of rings with and w/o pruning on NELL-995. The results out of the figure are those who cannot get within 48 hours. The same color with different shades is for the same path length, the dark color represents with pruning, and the light color represents w/o pruning. The solid are the results with pruning.

B.2 System Efficiency on NELL-995

We also experiment the system efficiency on NELL-995 dataset. NELL-995 also has enought rings for us to sample. We set the sample ratio as 1%, 5%, 10%, 15% and 20%, which are the same as the experiments on YAGO 3-10. And the results are in Fig. 12. Results show that the pruning strategy can improve the efficiency for ring discovery around 60% for $[3, 4]$ and around 70% for $[4, 5]$. The reduction is smaller than YAGO 3-10, because the data sizes of NELL-995 are smaller than that of YAGO 3-10.

Table 4: The statistics of sample result of NELL-995.

Sample Ratio(%)	# Entities	# Triples
1	14929	26625
5	19566	39165
10	24032	49223
15	27378	55819
20	30689	62774