Ring-based Semantic Conflict Location and Correction in Knowledge Graphs

Yihan Wang, Qi Song, Ziwen Wang, Ling Zheng, Xiang-Yang Li
University of Science and Technology of China, Hefei, China
yihanw@mail.ustc.edu.cn, qisong09@ustc.edu.cn, {wangziwen, lingzheng}@mail.ustc.edu.cn, xiangyangli@ustc.edu.cn

***Abstract*—Knowledge graphs (KGs) use graph structures to represent facts and associations in the real world. Domain-specific knowledge graphs, such as industrial KGs, are critical for industry analysis and decision-making, thus requiring a low tolerance for errors or conflicts. However, these knowledge graphs integrate heterogeneous data from multiple sources, which often leads to semantic conflicts among entities. The ensuing challenge is how to precisely locate and correct specific conflict triples in these semantic conflict triple sets. In this paper, we provide a formal definition of the conflict triple location problem on the KG and prove that the problem is NP-hard. Although intractable, we develop feasible algorithms, including exact algorithms with pruning techniques, as well as an approximate method** ApproxLoc **that achieves a near-optimality guarantee $O(log|\mathcal{R}|)$, where $|\mathcal{R}|$ is the number of so-called conflict rings, which is a ring structure that contains the conflict triple(s). After the conflict triples have been located, we further introduce** CorrectCon**, a conflict correction framework that includes embedding and prediction models, which incidentally categorizes the types of conflicts in triples. Using real-world graphs, we show that** ApproxLoc **achieves a 100% recall and over 90% precision while significantly improving the efficiency. Our case analysis also verifies that the algorithms can locate and correct real conflicts.**

***Index Terms*—Knowledge Graph, Semantic Conflict**

I. Introduction

Knowledge Graphs (KGs) are an effective method for representing the entities and the relationships among them in real world [1, 2], thus can be used in numerous real-world applications, including data management [3], information retrieval [4–6], etc. Among all the KGs, domain-specific knowledge graphs (DKG), such as industrial knowledge graphs, have attracted increasing attention in recent years as they form the cornerstone of domain-specific cognitive intelligence applications [7, 8].

Domain-specific knowledge graphs are typically employed in various complex analyses and decision-making processes, e.g., manufacturing and production tasks under the industrial scenario. One widely used application is BOM (bill of materials) management, where a material knowledge graph is constructed based on the materials and the relationships among them. One such relationship between two materials represents that they can be installed on a certain type of product. BOM knowledge graph can be used to detect potential conflicts, such as putting a 2666MHz memory on a motherboard that provides only 2133MHz memory slots. These tasks are very sensitive to data quality, i.e., even minor errors in the DKG can lead to significant economic costs and damage the company's social reputation. Moreover, any uncorrected errors within the DKG can significantly hinder rather than help engineers[9].

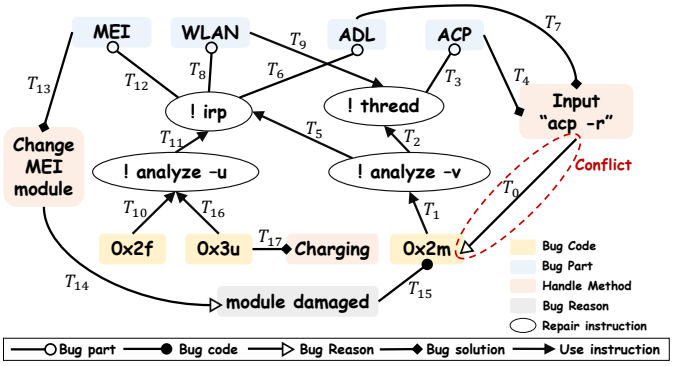


Fig. 1. Conflicts in an industrial knowledge graph.

In this paper, we mainly consider conflict, which represents the incorrect relationships among entities, as the major data quality issue for DKGs. Since the DKGs are constructed based on multiple data sources, conflicts can be easily introduced during the construction process. For example, the malfunction of PCs or cell phones is a very complex process. One type of PC may contain up to 3 thousands of types of materials, including software and hardware, and up to 40 thousand of collocation relations of the materials [10]. Given the large volume, it is very hard for domain experts to verify the correctness of the whole DKG. Thus requires an automatical way to effectively detect these conflicts.

Example 1: Fig. 1 depicts part of a DKG for PC malfunction with a potential conflict. The triple T_0 itself is correct, which means the bug reason of "0x2m" is "Input "acp -r"". However, there is a set of triples $ST_0 = \{T_1, T_2, T_3, T_4\}$, which indicates that the solution of "Input "acp -r"" is "0x2m". Thus, "Input "acp -r"" is both the reason and solution for "0x2m", which leads to a conflict. T_0 and ST_0 together form a ring structure R_0, we thus call it a conflict ring. Besides R_0, there are two other rings containing T_0: $R_1 = \{T_0, T_1, T_5, T_6, T_7\}$, and $R_2 = \{T_0, T_1, T_5, T_8, T_9, T_3, T_4\}$, which have the same conflict as R_0. Consider R_0, R_1 and R_2 together, there is a high probability that T_0 and T_1 are erroneous. □

As shown in Example. 1, detecting some semantic conflicts requires checking not only the incorrect triple itself, but the semantically related triples. The first step of correcting these conflicts is to locate them, which is a challenging task for domain experts, given the large volume and complex semantics. For instance, it is difficult to determine whether "Input "acp -r"" is the reason or solution of "0x2m" with only R_0 in Example. 1. At the same time, existing error detection [11–13]

methods primarily focus on detecting the error at the triple granularity, which mainly consider the semantic correlation between the two entities. However, complex conflicts typically involve a set of triples. Rule-based methods [14–17] have been proposed to deal with conflict detection in this case by specific patterns. Unfortunately, it's still difficult to locate the conflict triples for these methods due to the following challenges.

C1: Semantically related sub-graph. Boundaries between correct and conflict triples are hard to delineate. As shown in Example. 1, relying on a single set of triples does not suffice to distinguish between correct and conflict triples. Accurate assessments require information from semantically related triples. Collecting information from the whole KG is ineffective and not necessary. Thus we need to design a strategy to obtain a sub-graph, i.e. containing all semantically related triples, that can be used to locate conflicts.

C2: Statistically similar. A conflict triple can be contained in multiple conflict triples sets, i.e., T_0 contained in R_0, R_1 and R_2 in Example. 1. This establishes a one-to-many $(1 : n)$ relationship for the triple and all the related conflict triples sets. At the same time, these conflict triples sets may contain the same correct triple, which also forms a similar one-to-many $(1 : n)$ relationship. This makes it challenging to distinguish between conflict and correct triples statistically.

C3: Strong dependency. Conflict triples often have strong dependency with specific and correct triples. As shown in Example. 1, T_0 and T_1 have a strong dependency, i.e., both of them are contained in R_0, R_1 and R_2. This dependency complicates the distinction between correct and conflict triples, resulting in these correct triples easily being located as conflict.

In this paper, we aim to locate and correct the conflict triples while minimizing the impact of irrelevant triples. We propose various algorithms for conflict location, from exact algorithms to an approximate algorithm, while we also propose a correction method to get the correct relation.

Contribution. This paper studies feasible algorithms to locate and correct conflicts in complex knowledge graphs.

1) We formalize both the conflict location and correction problems for KGs. We focus on semantic conflicts within rings in KGs. We show that the ring-based conflict location problem is NP-hard (Sec. II).
2) Despite the hardness, we first propose an enumeration-based algorithm ExactLoc to compute the optimal conflict triple set with time complexity $O(2^{|\mathcal{T}|})$, where $|\mathcal{T}|$ is the number of triples contained in all conflict rings. A sub-graph generation method has been proposed to solve the related sub-graph issue. We also optimize this algorithm by introducing pruning strategies (Sec. III).
3) To improve the efficiency of conflict location, we propose ApproxLoc, an approximation method that achieves a near-optimality guarantee $O(\log |\mathcal{R}|)$, where $|\mathcal{R}|$ is the number of conflict rings. This approach contains two major components that can handle the statistically similar and strong dependency challenge. The *Selector* involves two selection strategies to get all the candidate conflict triples from a large number of triples, while the *Filter* removes irrelevant triples. Compared with ExactLoc, ApproxLoc significantly reduces the time complexity to $O(|\mathcal{T}|^2)$ (Sec. IV).
4) After the conflicts are located, we propose CorrectCon, a correction framework contains an embedding model and a prediction model. CorrectCon transforms the correction problem into relation prediction within the ring (Sec. V).
5) Using real-world graphs, we experimentally verify the effectiveness and efficiency of ApproxLoc and CorrectCon. The results show that ApproxLoc gets a recall of 100%, and a precision of more than 90% with up to 10^6 times faster compared with ExactLoc. We also show that ApproxLoc and CorrectCon can locate and correct conflicts over real-world knowledge graphs in the case study (Sec. VI).

These results yield effective methods for conflict location and correction in large graphs. We also provide a full version [18] which contains all the detailed proofs, the description of the algorithms and the related analyses.

Related Work. We categorize the related work as follows.

Error Location in Knowledge Graph. Previously, studies on errors in KGs mainly studied on single triples, which do location via detection. There are two main types of methods. The first category relies on embedding-based methods, such as TransE [19] and its variants [20–24], CKRL [11], KGTtm [12], GEDet [25], CAGED [26], PGE [13] and NYLON [27], use the structure of KG that is embedded while evaluating the confidence, while PGE [13] is used on product graphs, and NYLON [27] is used on hyper-relational KGs. CKRL [11], which employs the structure to identify discrepancies. KGTtm [12] merges internal KG details with broader contextual data. GEDet [25] combines graph data enhancement and generative adversarial networks. CAGED [26] uses contrastive learning for KG error detection. However, since the above methods perform at the triple granularity, they do not need location. In addition, some researchers extend methods used in other types of conflict resolution methods to KGs, such as Dis-Dedup [28], GFDs [14], Raha [29], Holodetect [30], PREEDet [31], Dis-Dedup [28], TKGC [32], SHACTOR [17]. They use cleaning rules on specific patterns. But they still have no idea about semantic conflict location.

Relation Correction. Most quality management work focuses on error detection and graph completion [33], but there is less research on error correction. There are only a few works that focus on correcting entities [34–36] or assertions [37, 38] but do not pay attention to correcting relations. Assertion correction [37, 38] is mainly used to correct grammar in sentences and is unsuitable for KGs. Current entity correction work in KG is mainly used in automatic speech recognition systems [35, 36], using context comparison and query rewriting to correct the entities without paying attention to the relations. However, we find that relation correction in KGs is also an important problem. Moreover, there are studies on label correction in machine learning using the idea of prediction. PENCIL [39], M-correction [40], PLC [41] and SELC [42] all utilize historical model outputs for label correction through

prediction. Inspired by label correction, we employ prediction and knowledge representation learning for KGs to obtain the relation prediction embedding of erroneous triples.

Knowledge Representation Learning. To apply machine learning algorithms to KGs, many knowledge representation learning (KRL) methods have emerged. Traditional KRL methods, including the TransE series [19–22], ComplEx [23], DistMult [24], SimplE [43], and RotatE [44], vary in aspects such as representation space, scoring functions, coding models etc [45]. Among these methods, TransE and RotatE represent their relations in the same vector space as entities, enabling them to represent compositional relations in KG effectively. Graph neural networks (GNNs) have also attracted attention for their ability to extract structural features from graphs [46–48]. Some researchers use GNNs to extract structural information from KG implicitly [49–51]. Pre-trained Language Models (PLMs) [52–54] effectively address the expressiveness gap for long-tail entities in structure-based embeddings. However, it's worth noting that while pre-trained language models perform admirably in this aspect, structure-based methods typically outshine them when it comes to popular entities. We use the above methods to verify the performance of CorrectCon. Experiments prove that the above work can provide a reasonable embedding method for conflict correction, making it suitable for relation prediction.

II. Problem Formulation

In this section, we first define conflicts (Sec. II-A), and then formulate conflict location and correction problem (Sec. II-B).

A. Graphs and Conflicts

We start with the notions of graphs and conflict.

Graphs. We consider a directed graph $G = (V, E, L_E)$, where V is a finite set of vertices and $E \subseteq V \times V$ is a set of edges. The edges are ordered pairs of the form (u, v), representing a directed edge from vertex u to vertex v. Each edge $e \in E$ carries a label $L(e)$.

Triples. A directed graph can represent a knowledge graph (KG). A triple in the KG, also known as a fact or statement, is a triple $T = (h, r, t)$, where $h, t \in V$ and r corresponds to $L(e)$ for each edge $e \in E$.

Paths. A path $p_{v_1 \to v_n}$ from vertex v_1 to v_n in G is a list $p_{v_1 \to v_n} = (v_1, r_1, v_2, r_2, \ldots, r_{n-1}, v_n)$, such that $(v_{i-1}, r_{i-1}, v_i), i \in [2, n]$ is an edge in G (i.e. a triple in KG). A path is simple if $v_i \neq v_j$ for $i \neq j$, i.e., each vertex appears on $p_{v_1 \to v_n}$ at most once. We consider simple paths in our work.

Rings. A ring in G is a non-empty vertex sequence $(v_1, v_2, \cdots, v_m), v_i \in V$, satisfying the following conditions:

1) *Vertices are distinct*: if $i, j \in [1, m], i \neq j$, then $v_i \neq v_j$.
2) *Adjacent vertices form an edge*: $\forall T_i = (v_i, v_{i+1}), i \in [1, m)$, which is a triple, $\exists\,(v_i, v_{i+1}) \in E$ in G.
3) *The sequence is closed*: $\exists\,(v_m, v_1) \in E$, forming a closed loop that starts from and returns to v_1.

Thus, we can also represent the ring as a triple sequence $(T_1, T_2, \cdots, T_m)$, where each T_i is a triple with head entity v_i and tail entity v_{i+1}. Notably, T_m is a triple from v_m to v_1.

Conflicts. In our work, we consider conflicts as semantic inconsistency among rings. Consider the ring $(T_1, T_2, \cdots, T_m)$, which can also be denoted as $((v_1, r_1, v_2), (v_2, r_2, v_3), \cdots, (v_m, r_m, v_1))$. If $(T_1, T_2, \cdots, T_m)$ has semantic inconsistency, we call the ring has at least a conflict. To fully understand semantic inconsistency, we must first define semantic consistency within the ring.

Semantic Consistency. Following the ideas of TransE [19], the elements in the triples can be represented in the same vector space. We define

$$\begin{aligned} \vec{W} &= (\vec{v_1} + \vec{r_1} - \vec{v_2}) + \cdots + (\vec{v_m} + \vec{r_m} - \vec{v_1}) \\ &= \vec{r_1} + \vec{r_2} + \cdots + \vec{r_m} \end{aligned} \tag{1}$$

Semantic consistency means $||W|| \leq \epsilon$ (ϵ is a small error).

Semantic Inconsistency. The same as the definition of semantic consistency, semantic inconsistency means $||W|| > \epsilon$.

Conflict Triples. For a triple $T_k = (v_k, r_k, v_q)$, if the triple is erroneous, which causes semantic inconsistency with other triples in any rings, we call it a "conflict triple".

Conflict Rings. A triple list $(T_1, T_2, \cdots, T_m)$ can also be represented as $((v_1, r_1, v_2), (v_2, r_2, v_3), \cdots, (v_m, r_m, v_1))$. A conflict ring is defined as a ring where $\vec{r_1} + \vec{r_2} + \cdots + \vec{r_m} > \epsilon$. Conflict rings have two obvious properties: (1) A conflict ring contains at least one conflict triple. (2) If a triple is a conflict triple, all rings containing it are conflict rings.

Example 2: Using the set of triples in Example. 1, R_0, R_1 and R_2 all can form a ring in the KG, respectively. R_0, R_1 and R_2 are all conflict rings that have semantic inconsistency. Fig. 1 shows T_0 is a conflict triple, which makes R_0, R_1 and R_2 to be conflict. □

Lemma 1: *If there are conflicts between two interconnected entities (i.e., vertices) h and t, the conflicts must indeed exist within the rings containing them.* □

Proof sketch: If h and t are interconnected, there are a path $p_{h \to t}$ from h to t, and a path $p_{t \to h}$ from t to h in G. There is a semantic associated with $p_{h \to t}$ and another with $p_{t \to h}$. If these two semantics conflict, it indicates that at least one conflict triple exists among the triples in $p_{h \to t}$ and $p_{t \to h}$.

However, all triples in $p_{h \to t}$ and $p_{t \to h}$ may not form just one ring. If there are j same vertices between $p_{h \to t}$ and $p_{t \to h}$, all triples in $p_{h \to t}$ and $p_{t \to h}$ can be divided into $(j+1)$ rings. These rings are represented as follows: $< h \to f_1, f_1 \to h >, < f_1 \to f_2, f_2 \to f_1 >, < f_2 \to f_3, f_3 \to f_2 >, \ldots, < f_{j-1} \to f_j, f_j \to f_{j-1} >, < f_j \to t, t \to f_j >$. If there are conflicts between two interconnected entities h and t, there is at least one conflict ring among the $(j+1)$ rings. □

Lemma. 1 proves that conflicts are expressed in the form of simple rings in KGs, so conflicts on any set of triples become conflicts on the set of triples, which can form a ring. Thus, finding conflicts can be converted to finding conflict rings.

B. Conflict Location & Correction

We tend to solve the conflict location and correction problem as illustrated in Fig. 2. For conflict location, we tend to find the conflict triples by using a set of conflict rings as inputs. Then, for conflict correction, we try to modify each conflict triple to get the correct one.

Next, we will formulate the problems in detail.

Problem Statement (Conflict Location). Given a set of conflict rings $\mathcal{R} = \{R_1, R_2, \cdots, R_n\}$, every ring is represented as $R_i = \{T_1^i, T_2^i, \cdots, T_{d_i}^i\}$ using triples. Thus, we can get the set of triples containing in all conflict rings $\mathcal{T} = \{T_1, T_2, \cdots, T_l\}, l = \sum_{i=1}^{n} d_i$. The set of all conflict triples in these conflict rings is defined as $H = \{T_k | T_k \in \mathcal{T}, \forall R_j \ni T_k, R_j \; is \; conflict \; ring\}$. The problem of conflict location is to find the conflict triples set H

$$H \subseteq \mathcal{T}, \text{s.t.} \arg\min |H|, \forall R_i \in \mathcal{R}, H \cap R_i \neq \varnothing \tag{2}$$

To refine this problem, we consider

$$\mathcal{T}^* = \{T_j | T_j \in \mathcal{T}, and \frac{|R_j^*|}{|C_j|} > 1 - \delta\}, \tag{3}$$

where $R^* = \{R_k | R_k \in \mathcal{R}, T_j \in R_k\}$, $C = \{C_i | C_i \; is \; a \; ring, T_j \in C_i\}$, and $\frac{|R_j^*|}{|C_j|}$ is the conflict cover ratio of T_j. So, the conflict triples set H needs to satisfy two conditions:

- Cover Condition: As formulated in Equ. 2, cover condition means that there is at least one triple in each conflict ring that exists in the conflict triples set H.
- Conflict Condition: Conflict condition means that each triple in the conflict triples set must be conflict as Equ. 3 shows.

Thus, we can rewrite the conflict location problem with the cover and conflict conditions as follows:

$$Find \; H \subseteq \mathcal{T}^*, \; \text{s.t.} \arg\min |H|, \forall R_i \in \mathcal{R}, H \cap R_i \neq \varnothing. \tag{4}$$

The conflict location problem is, nevertheless, nontrivial.

Theorem 2: *The problem of finding the conflict triples set (conflict location) is NP-hard.* □

Proof sketch: We show a strong result that the conflict location problem is already NP-hard when the conflict condition is not considered, by a reduction from "vertex cover". Consider a set of conflict rings $\mathcal{R} = \{R_1, R_2, \cdots, R_n\}$ and a limit length b. The vertex cover is a subset of vertices $V' \subseteq V$ of a graph $G = (V, E)$. We construct a mapping from graph G to conflict triples set H, that each vertex in G corresponds to each triple in H, and each edge (u, v) corresponds to a ring $R_i = \{T_u, T_v\}$ with two triples. In this way, for each edge in G, there is at least one vertex in the vertex cover set VC. So, $\forall R_i, R_i \cap VC \neq \varnothing$, which means that VC with b vertices corresponds to H. In turn, because $\forall R_i, R_i \cap H \neq \varnothing$, there is at least one vertex from each edge in the final set. So, H with b vertices corresponds to VC. As vertex cover is NP-hard, conflict location with limit set length only is already NP-hard, let alone when the conflict condition is added. □

As a complete process, we first attempt to location. While the conflicts have been located, we next define the conflict correction problem that can tell the correct relation.

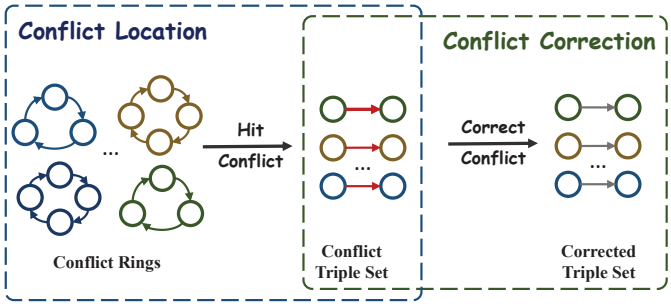


Fig. 2. Overview of the conflict location and correction problem.

Problem Statement (Conflict Correction). The correction problem of conflict triples in the KG is as follows.

- *Input*: a conflict triples set $H = \{T_1', T_2', \cdots, T_b'\}$.
- *Output*: a correct triples set $H' = \{T_1'', T_2'', \cdots, T_b''\}$ for each triple in H, where each T_i'' represents the correct triple for T_i', ensuring that each triple in H' is not conflict triple.

It is worth mentioning that this work only considers correcting the relation r of each triple. We consider the correction in both the direction and relations (predicates) of r.

III. Exact Conflict Location

We first introduce algorithms to solve the conflict location problem. In order to tackle the first challenge, which aims to get the semantically related sub-graph, we first introduce a sub-graph generation process (Sec. III-A). We next design some metrics that can be used to evaluate the properties of the conflicts (Sec. III-B). To effectively locate conflicts, we introduce an optimal algorithm with some pruning techniques (Sec. IV). To further improve the efficiency, we propose an approximate algorithm ApproxLoc in Sec. IV.

A. Sub-graph Generation

Before locating and correcting conflicts within the KG, it is essential to get all conflict rings in G. However, discovering all rings and finding those with conflicts is not feasible. the complexity of discovering all rings reaches $O(2^{|V|} \cdot (|V| + |E|))$. Thus, it requires an efficient way to discover rings.

Given the substantial challenge of finding all rings in G, one can only find a few conflict rings (called small conflict rings set) in finite time. This is far from enough to find the conflict triples among the small conflict rings set.

Remark. The ratio of conflict triples in a real-world KG is always low, for instance, 5% in YAGO [55]. Thus, the number of conflict triples in the conflict rings set is much smaller than the correct triples. Moreover, one conflict triple can cause many rings to be conflict. Thus, we can find conflict triples by a *recurring condition*.

However, the triples within only a few conflict rings cannot meet the recurring condition. Therefore, it is necessary to generate a related sub-graph first, thereby enabling the finding of more additional conflict rings to fulfill the recurring condition. To achieve this, We provide a sub-graph generation method followed by finding all conflict rings within this sub-graph to fulfill the recurring condition, which is the foundation for conflict location, as illustrated in Fig. 3.

Practically, we use breadth-first search (BFS) on the entire KG across all entities within the triples from the small conflict rings set above, to find out all entities and triples within a λ hops link. This λ-hops limit is used to balance the expansion of the number of rings available for both exact and approximate algorithms with the risk of overly enlarging the sub-graph, which can increase the overhead in ring discovery. Thus, using Equ. 3 on all rings in this sub-graph, can provide enough conflict rings for conflict location.

B. Conflict Assessment

We want "real" conflict triples from the algorithms, assessed by Set Cover Ratio (SCR) and Conflict Cover Ratio (CCR).

Set Cover Ratio (SCR). SCR is a metric used to evaluate the prevalence of a triple across all rings. A higher SCR indicates that the triple requires greater attention for conflict location.

$$SCR = \frac{|AR|}{|R|} \tag{5}$$

where $|AR| \leq |R|$, $|AR|$ is the number of rings containing a particular triple and $|R|$ is the total number of rings in KG.

Conflict Cover Ratio (CCR). CCR quantifies the proportion of conflict within the rings that contain a specific triple. A higher CCR shows that the triple is more likely to be conflict.

$$CCR = \frac{|ER|}{|AR|} \tag{6}$$

where $|ER| \leq |AR|$, $|ER|$ is the number of conflict rings containing the given triple.

Obviously, for any given triple, $|AR| \leq |AR| \leq |R|$, with $|R|$ remaining constant.

Remark. Theoretically, for conflict triples, $CCR = 1$ as the definition of "Conflict Ring". However, in practice, conflict rings can be misjudged as correct by both humans and automatic methods when they are complex, making $CCR < 1$.

Example 3: As Example. 1 shows, there are three rings containing T_0, thus, $|AR|$ for T_0 is 3. Moreover, the three rings R_0, R_1 and R_2 are all conflict rings, which means $|ER|$ for T_0 is 3. Besides the rings in Example. 1, there is one more ring $R_3 = \{T_1, T_5, T_{12}, T_{13}, T_{14}, T_{15}\}$. Thus, we have $|R| = 4$. Thus, in this small part of KG, the SCR of T_0 is $\frac{3}{4}$ and the CCR is $\frac{3}{3} = 1$. Similarly, for T_1, $SCR = 1$, $CCR = \frac{3}{4}$. □

C. Exact Conflict Location Algorithm

Next, we will discuss the exact algorithms for conflict location. The approximate algorithm to reduce the time complexity to scale with large graphs is in Sec. IV.

The exact algorithms consider all conflict rings from the sub-graph as input. Output is an optimal conflict triples set $\mathcal{H}$, which is composed of conflict triples that minimize the set size and satisfy the conflict condition in Equ. 3. It is not practical to enumerate all combinations of triples in the conflict rings, and evaluate whether each combination satisfies the cover and conflict condition. However, we can still design algorithms to get optimal results.

Algorithm 1: ExactLoc

Input: $\mathcal{R}^*$: rings set, $\mathcal{R} \subseteq \mathcal{R}^*$: conflict rings set.
Output: $\mathcal{H}$: optimal conflict triples set.

1 $\mathcal{H} := \varnothing$; $\mathcal{H}' := \varnothing$; $Metric := 0$;
2 Get all triples $\mathcal{T}$ in $\mathcal{R}$; Get all sets $\mathcal{O}$ from $\mathcal{T}$;
3 Initialize a dict $Metric_{\mathcal{T}}$ for each $\tau \in \mathcal{T}$, value is 0;
4 **for** *each* $\tau \in \mathcal{T}$ **do**
5 Calculate SCR and CCR based on $\mathcal{R}^*$ and $\mathcal{R}$;
6 Get the metric $Metric_{\tau}[\tau]$;
7 **for** *each* $O \in \mathcal{O}$ **do**
8 **for** *each* $R \in \mathcal{R}$ **do**
9 Check whether $R \cap O$ is empty;
10 **if** O *satisfies cover condition* **then**
11 Add set O to $\mathcal{H}'$.
12 **for** *each* $h' \in \mathcal{H}'$ **do**
13 Get the metric $Metric_{h'}$;
14 **if** $Metric_{h'}$ *performs better* **then**
15 $Metric := Metric_{h'}$; $\mathcal{H} := h'$;
16 **return** $\mathcal{H}$.

1) Computing Optimal Conflict Triples Set with Enumeration: Our first algorithm, denoted as ExactLoc, is illustrated in Alg. 1. It takes an enumeration scheme to get the optimal conflict triples set.

1) It first gets all triples $\mathcal{T}$ to generate all combinations $\mathcal{O}$, and each combination is a triples set. Because it aims to find one set to cover all rings in $\mathcal{R}$, it is easy to find that the upper bound of the set size is the number of conflict rings ($|\mathcal{R}|$). But, the number of triples ($|\mathcal{T}|$) can be less than $|\mathcal{R}|$, thus, the upper bound of $|\mathcal{H}|$ is $\min(|\mathcal{T}|, |\mathcal{R}|)$ (lines 1-2).
2) It then verifies each triples set in $\mathcal{O}$, finds those sets that satisfy the cover condition, and records them into a candidate set $\mathcal{H}'$ (lines 7-11).
3) Because it also needs to satisfy the conflict condition. It uses $Metric$ to find the best set in $\mathcal{H}'$ (lines 12-15). $Metric$ is pre-calculated at initialization to avoid repeated calculation, which is the same as the approximate algorithm in Sec. IV based on SCR and CCR (lines 3-6).

Correctness & Complexity. ExactLoc guarantees two conditions below: (1) lines 7 to 11 together with line 2 correctly generate and verify the cover condition, and (2) lines 4 to 6 and lines 12 to 15 correctly select the best set that satisfies the conflict condition. Thus, the correctness of ExactLoc follows.

We next analyze the time cost of ExactLoc. It is easy to find that the main time cost is from lines 7-11 in Alg. 1. The number of combinations is $|\mathcal{O}| = \sum_{k=1}^{min(|\mathcal{T}|,|\mathcal{R}|)} C(|\mathcal{R}|, k)$, where $C(\cdot,\cdot)$ is the combinatorial number in mathematics. Thus, $|\mathcal{O}| = 2^{min(|\mathcal{T}|,|\mathcal{R}|)} - 1$. So the lower bound of the time complexity is $O(2^{|\mathcal{R}|})$, and the upper bound of it is $O(2^{|\mathcal{T}|})$. In the conflict location task in a large graph, the $|\mathcal{R}|$ and $|\mathcal{T}|$ can be very large, which cannot be exactly verified in PTIME.

Algorithm 2: ExactLoc-Pu

Input: $\mathcal{R}^*$: rings set, $\mathcal{R} \subseteq \mathcal{R}^*$: conflict rings set.
Output: $\mathcal{H}$: optimal conflict triples set.

1 Get all triples $\mathcal{T}$ contained in $\mathcal{R}$; Get all sets $\mathcal{O}$ in $\mathcal{T}$;
2 Initialize a dict $Metric_{\mathcal{T}}$ for each $\tau \in \mathcal{T}$, value is 0;
3 **for** *each* $\tau \in \mathcal{T}$ **do**
4 Calculate SCR and CCR based on $\mathcal{R}^*$ and $\mathcal{R}$;
5 Get the metric $Metric_{\tau}[\tau]$;
6 Initialize a dict $Metric_{\tau}$ for each $r' \in \mathcal{R}$, value is 0;
7 Sort $Metric_{\tau}$ in positive order of the value part.
8 **for** *each* $\tau \in Metric_{\tau}.keys$ **do**
9 **call** ExactLoc-Pu(BackTracking) to get Cur_best_Tri;
10 $\mathcal{H} := Cur\_best\_Tri$;
11 **return** $\mathcal{H}$.

Algorithm 3: ExactLoc-Pu(BackTracking)

Input: $\mathcal{R} \subseteq \mathcal{R}^*$: conflict rings set, $Metric_{\mathcal{R}}$: the sorted triple list, Cur_Tri: current triples set, Cur_Metric: current metric of Cur_Tri, Cur_Index: current index of tracking, Cur_Cover: current covered rings, $Best\_Metric$: current optimal metric.

1 Expand Cur_Tri with the next triple in $Metric_{\mathcal{R}}$;
2 Update Cur_Cover and Cur_Metric;
3 **if** $|Cur\_Tri| > |\mathcal{R}|$ **then**
4 **return**
5 **if** $Metric_{Cur\_Tri} < Best\_Metric$ *and* $|Cur\_Tri| < |Cur\_best\_Tri|$ **then**
6 **return**
7 **if** $|Cur\_Tri| == len(\mathcal{R})$ **then**
8 $Best\_Metric := Cur\_Metric$;
9 Record Cur_Tri as Cur_best_Tri;
10 **return**
11 **for** *Smallest index in* $Metric_{\mathcal{R}} > Cur\_Index$ **do**
12 **call** ExactLoc-Pu(BackTracking)

2) Computing Optimal Conflict Triples Set with Pruning: The algorithm, denoted as ExactLoc-Pu is illustrated in Alg. 2. It takes two pruning strategies following backtracking to avoid unnecessary verifications.

1) Different from ExactLoc, ExactLoc-Pu uses $Metric$ to sort triples at the begining (line 7 in Alg. 2).
2) It calls ExactLoc-Pu(BackTracking) to avoid unnecessary verifications and get the optimal result $\mathcal{T}$ by using the backtracking method (lines 8-10 in Alg. 2).
3) *First pruning*: It verifies the triples in positive order of $Metric$. It continues to add the next triple in positive order into the set until the chosen triples satisfy the cover condition. It compares $Metric$ and size with the current best one to trigger the backtracking procedure, because the triples in the positive order are worse than the previous. This avoids the increase of the set (lines 3-4 in Alg. 3).
4) *Second pruning*: When the size of $Current\ \_Triples$ is greater than the current best, the backtracking procedure returns. Because any addition will make the result worse (lines 5-6 in Alg. 3).
5) The return operation in the backtracking procedure is used to remove the current last element in $Current\_Triples$ and add the next element to $Current\_Triples$. But if $Current\_Triples$ doesn't trigger pruning, it then verifies the cover conditions, and when satisfied, it compares and records the current best one. It prunes with the best metric when it satisfies the cover condition, so it satisfies the conflict condition (lines 5-10 in Alg. 3).

Correctness & Complexity. The correctness of ExactLoc-Pu follows as ExactLoc. The time complexity of ExactLoc-Pu is determined by the backtracking procedure. The depth of the recursive tree is $|\mathcal{R}|$, thus, in the worst case, when the two pruning strategies don't work, the complexity is $2^{|\mathcal{R}|}$ which is the same as ExactLoc. This means the "Return" operation cannot be triggered. But the "Return" operation tends to be triggered frequently, the average time complexity can be $O(2^k), k \in [|\mathcal{R}|, |\mathcal{T}|], |\mathcal{T}| \gg |\mathcal{R}|$. k is the average size of the candidate set, which means that when the length of it increases to k, the "Return" operation will be triggered. Thus, ExactLoc-Pu can reduce the time complexity from $O(2^{|\mathcal{T}|})$ to $O(2^k)$. The time complexity is also at index level, which is still intolerable for large graphs.

IV. ApproxLoc: Approximate Conflict Location

Algorithm ExactLoc and ExactLoc-Pu need to find and verify nearly every combination of triples in $\mathcal{T}$, which is computationally expensive. To mitigate these costs, we develop a feasible approximation algorithm that can avoid the cost of enumeration and verification.

Theorem 3: *There is an algorithm for conflict location with approximation ratio* $O(\log|\mathcal{R}|)$ *and time cost* $O(|\mathcal{T}|^2)$. □

That is, the algorithm guarantees a *relative* approximation ratio. Moreover, the algorithm is feasible for large graphs. To prove Theorem 3, we next show that the conflict location problem can be reduced to hitting set problem [56]. Each conflict ring R_i in $\mathcal{R}$ corresponds to an input set in the hitting set problem, with the triples in R_i representing the elements of each set. The goal is to find the smallest conflict triples set H that intersects (hits) every set in $\mathcal{R}$.

If the semantic conflict in rings could always be accurately detected, the CCR of each conflict triple would theoretically remain at 1. However, it is common for conflict rings to be mistakenly identified as correct, resulting in CCRs for these conflict triples falling below 1. Relying solely on "CCR equals 1" to locate conflicts would result in missing many conflict triples. To address this, ApproxLoc employs Lemma. 4 and 5 to design a more effective conflict assessment metric to get all candidate conflict triples. The correctness of these lemmas is also supported by our experiments, as illustrated in Fig. 5(b).

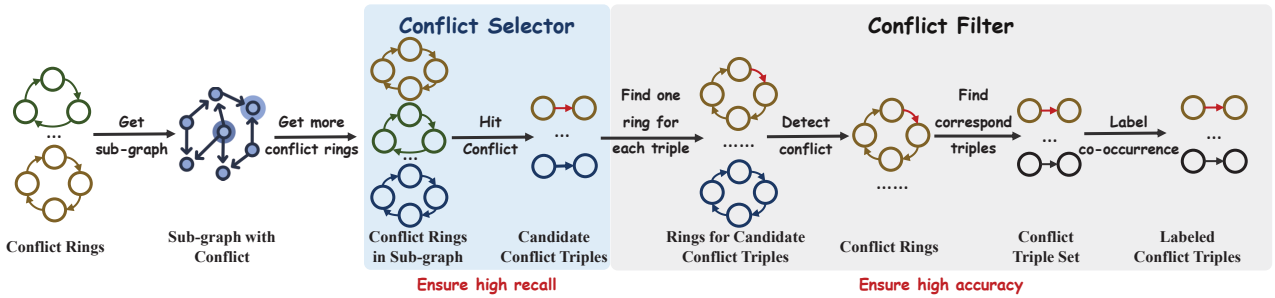


Fig. 3. ApproxLoc: Selector and Filter.

Lemma 4: *For each conflict triple, the SCR is less than or equal to the CCR, expressed as* $SCR - CCR \leq 0$. □

Proof sketch: Given a conflict triple $T_{conflict}$. The proof will be made from two categories. (To simplify the expression, we omit subscripts $T_{conflict}$ of all variables.)

- *Theoretically*, $CCR = 1$. According to the second property of the conflict ring in Sec. II, all rings containing the conflict triples are conflict rings, which means $|ER| = |AR|$, and $CCR = 1$. Obviously, the $SCR \leq 1$, because of $|AR| \leq |R|$. Thus, $SCR \leq CCR$.
- *Practically*, $CCR<1$. When there are rings containing the conflict triples are detected incorrectly, these rings are not considered as conflict, thus $CCR<1$. For $T_{conflict}$, we have the following:

$$SCR{-}CCR = \frac{|AR|}{|R|} - \frac{|ER|}{|AR|} = \frac{|AR| \times |AR| - |ER| \times |R|}{|R| \times |AR|} \tag{7}$$

Because $0 \leq |ER| \leq |AR| \leq |R|$, $|R| \times |AR| > 0$. The key is the value of $|AR| \times |AR| - |ER| \times |R|$. Although some of the conflict rings will be missed, $|ER|$ and $|AR|$ will be close. But $|AR| \leq |R|$, $|ER| \times (|AR| - |R|) \leq 0$, So, $|AR| \times |AR| - |ER| \times |ER| \leq 0$.

To sum up, all conflict triples have $SCR - CCR \leq 0$. □

Lemma 5: *All triples for which the SCR exceeds the CCR are considered correct triples* ($SCR - CCR > 0$). □

Proof sketch: Given a correct triple $T_{correct}$. The $|ER|$ is 0, thus, $CCR_{T_{correct}} = 0$. And $SCR_{T_{correct}} \geq 0$. Thus, the triples with $SCR - CCR>0$ are correct. □

Remark. It is important to note that not all correct triples necessarily satisfy the condition $SCR - CCR>0$. There are indeed correct triples where $SCR - CCR \leq 0$.

Following the second property of conflict rings in Sec. II, it is only necessary to identify a single ring for each triple to form the small rings set, which is used to determine semantic consistency or inconsistency to get the small conflict rings set. As a ring is composed of multiple triples, the size of the small rings set in a KG is much smaller than the number of triples. Consequently, as illustrated in Fig. 3, the input for ApproxLoc is the small conflict rings set, which effectively reduces the complexity of the conflict location problem. Before locating the conflict triples, Fig. 3 shows that we do sub-graph generation as Sec. III-A to meet the recurring condition. Subsequently, we propose *Selector* to get a candidate set of conflict triples that contain as many conflict triples as possible. However, as highlighted in Challenge C2, the converse of the second property of the conflict ring is not valid, i.e., a triple included in more than one ring is not necessarily conflict. To address this, we propose *Filter* to filter the candidate set, ensuring that each triple in the final set is conflict triple, thereby maintaining high precision without sacrificing recall. The final set of triples is the output of ApproxLoc.

Next, we will show the details of *Selector* and *Filter*.

A. Conflict Selector

The *Selector* operates based on the second property of the conflict ring in Sec. II, which necessitates that only the small conflict rings set is input, though it often fails to meet the recurring condition. Typically, the rings within this small set do not share common triples, which is a requisite for *Selector*. As illustrated in Fig. 3 and Sec. III-A, we propose a sampling strategy across the entire KG before *Selector* to generate a sub-graph. ApproxLoc focuses on the sub-graph containing all triples from the small conflict rings set, to get additional conflict rings for the *Selector*.

Thus, the input of *Selector* is the conflict rings in this sub-graph. Following Lemma. 4, ApproxLoc employs the $SCR - CCR$ as the metric (line 5-10 in Alg. 4). We have designed two distinct dynamic strategies to select triples in *Selector* based on their $SCR - CCR$ values:

- *Meet*: During each round, the *Selector* selects an unselected triple with the lowest $SCR - CCR$ value and subsequently updates the $SCR - CCR$ for all remaining unselected triples. This selection process continues until the selected triples meet the cover condition in Equ. 4. According to Lemma 4, this strategy efficiently includes conflict triples into the candidate set with high probability.
- *No-left*: In each round, the *Selector* picks an unselected triple that not only has the lowest $SCR - CCR$ value but also satisfies the condition $SCR - CCR \leq 0$. After each selection, the $SCR - CCR$ is recalculated for all remaining unselected triples. This selection process continues until there are no unselected triples left with $SCR - CCR \leq 0$. According to Lemma. 4, this strategy effectively ensures that all conflict triples are included in the candidate set, thus achieving a recall rate of 1.

In Alg. 4, we compute $SCR - CCR$ for all triples not yet included in the final set H after each selection round, in accordance with Equ. 8, which is the optimized version of

Equ. 4 with more metrics. This ensures that all conflict triples are in the candidate set. However, according to Lemma. 5, $SCR - CCR \leq 0$ may also occur on some correct triples. which could significantly compromise precision. To address this issue, we design a *Filter* to clean the candidate set by removing these correct triples.

$$\arg\min \frac{\sum_{tri \in H}(SCR_{tri} - CCR_{tri})}{|H|} \tag{8}$$

B. Conflict Filter

Given that the converse of the second property of conflict ring in Sec. II is not valid, it is imperative to filter the candidate set generated by the *Selector*. ApproxLoc employs two stages of filtering: ConflictDetection and co-occurrence. The ConflictDetection stage is designed to verify the semantic consistency of each triple in the candidate set (lines 11-14 in Alg.4). According to the second property of conflict ring in Sec. II, the input for ConflictDetection involves only one ring per triple, which is sufficient for accurate assessment. Then, the co-occurrence stage aims to identify pseudo-conflict triples that consistently appear with another conflict triple, and uses a co-occurrence label to change the original correctness of these triples when the number of co-occurrence is up to η (lines 15-23 in Alg. 4). This stage tells that the correctnesses of a co-occurring triple pair are the same, i.e. if one triple in a co-occurring triple pair is erroneous, the other must also be. Thus, both are considered conflict triples. Due to their co-occurrence, these pseudo-conflict triples are considered together with the corresponding conflict triples as real-world conflicts.

The algorithm, denoted as ApproxLoc and illustrated in Alg. 4, executes through the following steps.

1) It initializes set C_R and UC_R to record the current covered and uncovered rings in $\mathcal{R}$ (line 1).
2) It then computes SCR and CCR for each triple within conflict rings. It removes those where both SCR and CCR are 0, which are not in any rings and whose correctness cannot affect the semantic consistency (lines 2-4).
3) It then continuously selects a triple with the lowest $SCR - CCR$ among those with $SCR - CCR \leq 0$. This is the selection strategy of ApproxLoc to ensure this selected triple has the highest probability of being the conflict triple. It also updates the SCR and CCR for unselected triples, to reduce the chance of selecting frequent but conflict-free triples. This process continues until no unselected triples remain with $SCR - CCR \leq 0$, indicating that the remaining triples are correct following Lemma. 5 with No-left strategy (for meet strategy, change line 5 to "until meeting the cover condition"). Thus, it puts all conflict triples into the candidate set (lines 5-10).
4) It then finds a ring for each triple in the candidate set and verifies them. Correct triples are removed, which ensures the semantic consistency of each ring. This step results in the final set H (lines $11 - 14$).

Algorithm 4: ApproxLoc

Input: $\mathcal{R}^*$: rings set, $\mathcal{R} \subseteq \mathcal{R}^*$: conflict rings set, η: co-occurence threshold.
Output: H: the conflict triples set.

```
1  H := 0; C_R := ∅; UC_R := R;
2  Get all triples T contained in R;
3  Calculate SCR and CCR for all triples in T;
4  Remove the triples SCR = 0 and CCR = 0 from T;
5  while exist triples in T that SCR − CCR ≤ 0 do
6      Add triple (tri) with minimize SCR − CCR to H;
7      Add the rings cover by tri into C_R;
8      Remove tri from T;
9      Remove the rings cover by tri from UC_R;
10     Update SCR − CCR for triples in T;
11 for each triple tri in H do
12     Find a ring R_tri for tri in sub-graph;
13     if ConflictDetection (R_tri) == False then
14         Remove tri from H;
15 for each triple pair (tri, Tri) in H do
16     if Co-occurrences (tri, Tri) ≥ η then
17         Label the triple tri;
18 return H
```

5) It then finds the co-occurrence triple pair and labels them, in practice, it considers the number of co-occurrences up to η as co-occurrence triple pair (lines $15 - 18$).

Analysis. Above we construct a reduction from locating conflict triples to the hitting set problem [56], the approximation ratio thus follows. The time complexity of the *Selector* depends on the number of triples in the candidate set, denoted as $|\mathcal{T}|$, which is the upper bound when all the triples in conflict rings are in the candidate set. And the time cost of updating $SCR - CCR$ is also associated with $|\mathcal{T}|$. So, the time complexity of the *Selector* is $O(|\mathcal{T}|^2)$. The time complexity of the *Filter* is from the detection method, which is $O(|H|)$, and the co-occurrence label method, which is $O(|H|^2)$. Thus, the overall time complexity of the *Filter* is $O(|H| + |H|^2)$. Therefore, the total time complexity of ApproxLoc is $O(|H|+|H|^2+|\mathcal{T}|^2)$. Given that $|\mathcal{T}|$ is typically much larger than $|H|$, the dominant term in the time complexity of ApproxLoc is $O(|\mathcal{T}|^2)$. Hence, ApproxLoc significantly reduces the computational complexity from exponential terms $O(2^{|\mathcal{T}|})$ and $O(2^k)$ to $O(|\mathcal{T}|^2)$.

The above analysis completes the proof of Theorem 3.

V. CorrectCon: Conflict Correction Method

We next propose a conflict correction framework, CorrectCon, which utilizes relation embedding and prediction. To effectively handle all scenarios, we categorize the conflict into the following three types:

- **Direction Conflict:** The relation (predicate) of the edge is correct, but the direction is incorrect.

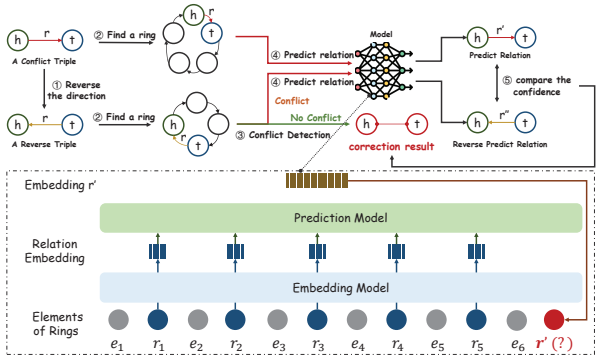


Fig. 4. The workflow and the model details of CorrectCon. The upper part is the workflow, and the lower part is the details of the model.

- **Relation Conflict:** The relation (predicate) of the edge is incorrect, but the direction is correct.
- **Direction + Relation Conflict:** Both the relation (predicate) and direction of the edge are incorrect.

The CorrectCon model is shown in Fig. 4. Considering the direction conflict, it is crucial to focus on not only the original triple (h, r, t), but also the reverse triple $(t, ?, h)$. We then run the prediction for both rings. To optimize the efficiency and minimize the number of prediction model invocations, CorrectCon first handles direction conflicts. If semantic conflict is detected in (h, r, t), CorrectCon searches for multiple rings that contain $(t, ?, h)$ to verify the correctness of these rings. If all rings are semantically consistent, the conflict is classified as a *direction conflict*, and the direction is adjusted accordingly. Otherwise, CorrectCon puts both rings as the input of the model to get two correction results. Then, both results are evaluated to choose the most accurate correction. CorrectCon can be equipped with various prediction models to correct the relation between h and t.

CorrectCon employs a cascading two-model approach for relation prediction, as illustrated in Fig. 4. The process begins with an embedding model, which generates embeddings for all relations within the ring. Commonly used embedding models include TransE and RotatE. Following the embedding model, a prediction model is employed. This model uses the relation embeddings generated by the embedding model as its input and is specifically trained on all rings in the sub-graph in ApproxLoc. Its primary role is to integrate these pre-obtained relation embeddings for each ring. During training, this model learns to fuse the embeddings of correct relations within a ring to infer the most probable correction for any conflict triple by assessing the fused embeddings from other relations in the same ring. Experiments also show that correction on rings by integration can simplify the correction task.

VI. Evaluation

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

Experiment Setting. We start with the settings.

Datasets. We use two real-life graphs: (1) YAGO3-10 [57] with sample rate 1% on the entities. We did a BFS for all triples to get rings and triples in λ hops, which can get enough triples for experiments. (2) NELL-995 [58] with sample rate 5% and the same λ hops BFS as YAGO to get triples, which can get the same size of data as (1). The details of the datasets are summarized in Tab. I. Since there is no known graph dataset for conflict detection that has available ground truth and labeled conflicts, we treated the original graph datasets without available ground truth as correct data, and manually injected erroneous as follows: (1) It randomly picks a set of triples with a specific error ratio. (2) For each triple, it modifies the relation to another relation, which is randomly selected. (3) Each modified (resp. original) triple is labeled as conflict (resp. correct), which uses 0 (resp. 1) in experiments. We thus construct the dataset used for quantitative analysis, while at the same time, we use the case study to show that the proposed method can locate conflicts for real-world knowledge graphs.

TABLE I
Datasets Statistics

Error Ratio	# Rings	# Triples	# Shortest Conflict Rings (# Conflict Triples) 10%	20%	30%
NELL (5%-3hops)	102574	39165	16 (16)	29 (29)	40 (40)
YAGO (1%-3hops)	92079	32917	436 (427)	885 (856)	1231 (1165)

"# conflict triples" is the number of conflict triples in all small conflict rings set for all conflict triples. This is not all the conflict triples in the sub-graphs. The error ratio means the ratio of all conflict triples in all triples.

Algorithms. For conflict location, we implemented the following algorithms and related baselines.

1) Exact algorithms: we implemented two variants, (a) ExactLoc enumerates all combinations of all triples in input rings, and (b) ExactLoc-Pu uses backtracking for pruning.
2) Approximate algorithms: we implemented ApproxLoc using different selection strategies: (a) ApproxLoc-λhops (Meet) runs until satisfying the cover condition, (b) ApproxLoc-λhops (No-left) runs until there are no triple with $SCR - CCR \leq 0$ in the unselected set. Shorest-only means that only use the original small conflict rings set as the input of the *Selector* and *Filter*.
3) Baselines: while existing error detection methods do not need to consider location, we implemented TransE [19], KGTtm [12] and CAGED [26] to compare the detection abilities on complex semantic conflicts with our methods.

For conflict correction, we implemented CorrectCon using three cascading models, which are TransE +TransE, TransE +LSTM, and RotatE+LSTM. Because RotatE is in the complex number space, it can not be used to predict.

Configuration. We implemented all the above algorithms in Python. We ran all experiments on a Linux machine powered by an NVIDIA GeForce RTX 4090 GPU, an Intel 3.0 GHz CPU with 128 GB of memory. Unless otherwise stated, we set $\lambda = 3$ and $\eta = 2$ as the default hops for sub-graph generation, and thresholds for co-occurrence, respectively. All experiments were repeated 10 times, and the average is reported here.

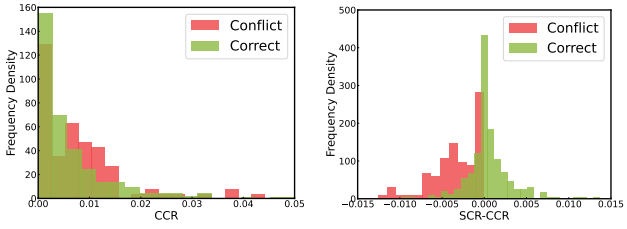
Experimental results. We next present our findings.

Before the experiments on effectiveness and efficiency, we first verified the rationality of the selection metric in our selection strategy, by using the the statistical results of the

TABLE II
ApproxLoc: EFFECTIVENESS

Datasets	YAGO-1-3					NELL-5-3				
Metric	Recall	Precision		F1-score		Recall	Precision		F1-score	
		Original	Filter	Original	Filter		Original	Filter	Original	Filter
TransE [19] -3hops	1	0.04		0.08		0.98	0.02		0.04	
KGTtm [12] -3hops	0.615	1		0.762		0.611	1		0.762	
CAGED [26] -3hops	0.656	0.261		0.373		0.470	0.328		0.386	
ApproxLoc-shortest only	0.266	0.090	0.736	0.135	0.391	0	0	0	-	-
ApproxLoc-3hops (Meet)	0.733	**0.828**	**0.946**	**0.778**	0.826	0.737	**0.875**	**1**	**0.8**	0.848
ApproxLoc-3hops (No-left)	**1**	0.308	0.908	0.471	**0.952**	**0.789**	0.246	**1**	0.375	**0.882**

YAGO-1-3 means 1% sample ratio with 3 hops BFS for sampling on YAGO 3-10, and NELL-5-3 means 5% sample ratio with 3 hops BFS on NELL-995. The bolds represent the best performances, and the underlines represent the second. The results of the baselines are on the 3 hops sub-graphs.



(a) CCR (b) SCR-CCR

Fig. 5. Statistic results of metrics selection.

metrics CCR and $SCR - CCR$. There is a clear separation between the correct and conflict triples in $SCR - CCR$ as shown in Fig. 5(b), which satisfies Lemma. 4 and 5 in Sec. IV.

Exp-1: Effectiveness (ApproxLoc). We first evaluate the effectiveness of our algorithms, including ApproxLoc with only the small conflict rings set and λ hops in Meet and No-left strategies. We compared our algorithms with the past error detection methods on the λ hops sub-graph. Tab. II shows the results and verifies the following. (1) Only the small conflict rings set can not satisfy the recurring condition of the hitting set algorithm because the size of the set is so small, and triples contained in each error ring are almost disjoint. (2) The past error detection methods can hardly successfully locate the complex conflicts, while our algorithms can successfully locate them. Although TransE [19] shows good recall, the F1-scores are low. This is because TransE [19] tends to detect all the triples it doesn't meet before to be conflict. Thus, only a few triples are correct, which makes the recall to be 1. For KGTtm [12], the precision is 1, while the recall is low, but location requires getting all the conflict triples and filtering irrelevant triples as much as possible. (3) Both Meet and No-left strategies can get up to 70% recall and 90% precision, which make the F1-score up to 80%. The results of the no-left strategy are always better than those of the Meet one. (4) ApproxLoc-3hops (Meet) and ApproxLoc-3hops (No-left) show the balance of recall and precision. In our design, the no-left strategy can recall all the conflict triples, while the meet strategy only recalls those with high conflict probabilities. So, we show a balance of recall and precision in the experiments among meet and no-left strategies. (5) All ApproxLoc strategies show the effectiveness of the *Filter*, and the *Filter* makes an up to 10% increase in the precision.

Comparing with exact algorithms. Given the complexity of ExactLoc and ExactLoc-Pu, we used a sampled dataset to compare the effectiveness of ApproxLoc with the exact algorithms. The dataset contains 10, 15, and 23 triples as shown in Tab. III. The results show that (1) ApproxLoc can successfully recall all the conflict triples, and the sets are the same as the output of ExactLoc and ExactLoc-Pu. (2) By using *Filter*, ApproxLoc can successfully filter the irrelevant triples. (3) ApproxLoc can get the result faster.

TABLE III
ApproxLoc: COMPARISONS WITH EXACT ALGORITHMS

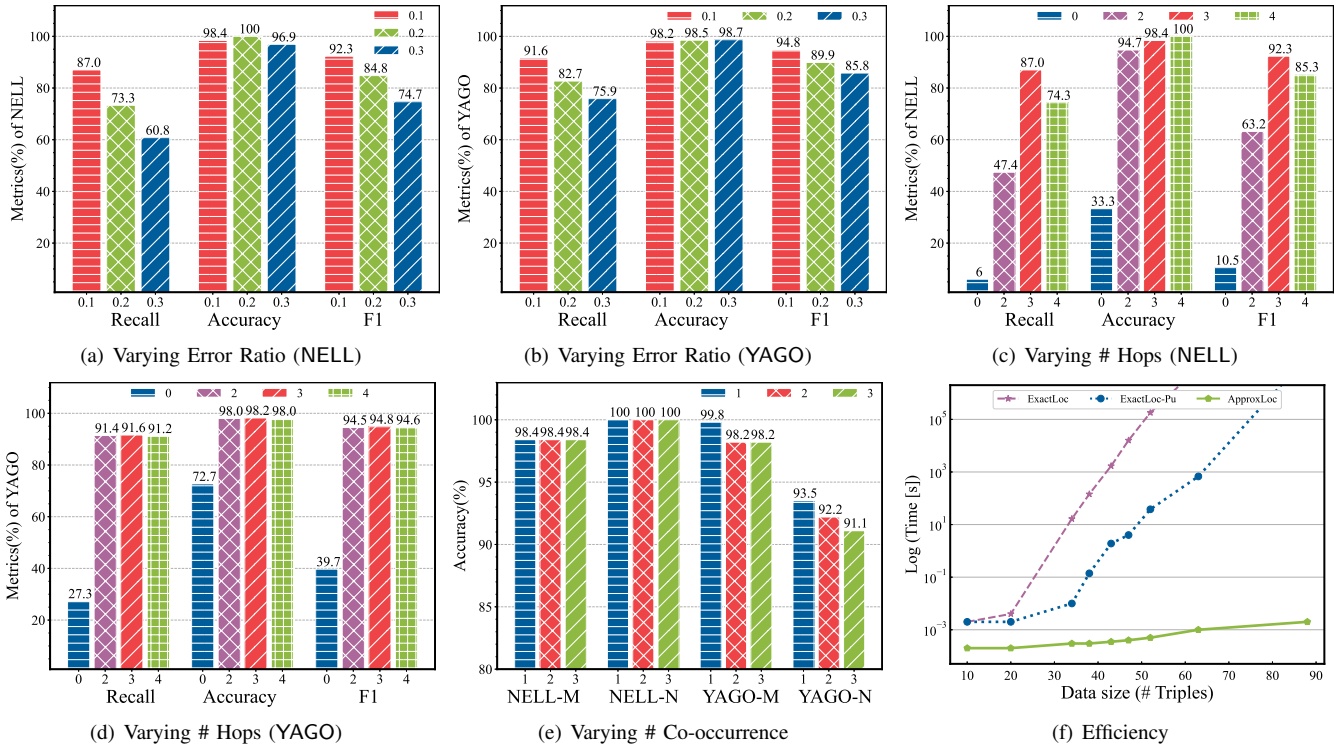
# Triples (# Conflicts)	# Rings (# Conflicts)	Recall	Precision	Speedup(time)
10 (2)	14 (5)	100%	100%	1.05×10^4
15 (3)	15 (12)	100%	100%	2.7×10^3
23 (6)	24 (21)	100%	60% (100%)	7.22×10^5

The ratio outside (resp. in) the bracket refers to the result using only ApproxLoc (resp. after *Filter*). Speedup is the running time of the exact divide approximate algorithm.

To demonstrate the capabilities of ApproxLoc well, we run all variable-parameter experiments using the meet strategy, as shown below. Since the no-left strategy always recalls all conflict triples, experiments cannot show the impact of changing parameters.

Varying Error Ratio. We used three different error ratios, 10%, 20%, and 30%. The corresponding number of conflict triples is shown in Tab. I. As shown in Fig. 6(a) and Fig. 6(b), (1) Under a fixed number of hops ($\lambda = 3$), we know that the higher the error ratio, the larger the size of the rings set. The precision remains high because error ratios do not affect the *Filter*. However, the recalls and F1 scores decrease in proportion to the error ratios. (2) For NELL (resp. YAGO), the recall decreases about 15% (resp. 10%), the F1-score decreases about 10% (resp. 5%) with the 10% increase of the error ratio. Since the magnitude of conflict triples in NELL is less than that of YAGO, we can see that the larger the order of magnitude, the less sensitive the metric is to the error ratio.

Varying # Hops (λ). We used the small conflict rings set as the input and varied λ (from 2 to 4) to get different sizes of sub-graphs with conflict as the first step in Fig. 3. We report the effectiveness with different hops in Fig. 6(c) and Fig. 6(d).



(a) Varying Error Ratio (NELL) (b) Varying Error Ratio (YAGO) (c) Varying # Hops (NELL)

(d) Varying # Hops (YAGO) (e) Varying # Co-occurrence (f) Efficiency

Fig. 6. ApproxLoc: Effectiveness and Efficiency. (c)(d) 0 means no-hop, which uses only the sub-graph consisting of all triples of the shortest conflict rings. (e) NELL-M (YAGO-M) means NELL (YAGO) with 3hops (Meet) strategy. NELL-N (YAGO-N) means NELL (YAGO) with the 3hops (No-left) strategy. (f) The Y-axis represents time in logarithmic scale (seconds), and data size means the number of triples in the sub-graph.

The results show that, (1) Only the small conflict rings are not enough to do conflict location. (2) The precision remains almost the same among different hops (except 0), which shows that the *Filter* works well. However, the recalls and F1 scores vary. For NELL, the sub-graph with 2 hops is not enough to meet the recurring condition. Thus, only a subset of conflict triples can be recalled. The sub-graph with 4 hops includes more distractions (frequency but correct triples in rings). Thus, the metrics get worse than the sub-graph with 3 hops. The experiments on NELL show that the sub-graph with 3 hops is the best choice to get the best results. The result over YAGO shows the same trends, but the impact is much smaller (0.2%). This means that the larger the order of magnitude, the less sensitive the metric is to the size of the sub-graph.

Varying # Co-occurrence (η). As the co-occurrence relation is used in the *Filter*, it does not impact the recall. We thus report the impact on precision varying η from 1 to 3. As shown in Fig. 6(e), (1) The precision on YAGO changes while the result over NELL remains the same. For NELL, we find that the co-occurrence relation for each triple pair is stable. Then in co-occurrence triple pairs, if one triple occurs in a ring, the other triple must occur. This is the strictest definition of co-occurrence. Thus the larger the order of magnitude, the more sensitive the precision is to η. (2) The precision decreases with η. This verifies that the larger the number, the stricter the condition is. Moreover, as the number of co-occurrences increases, the decrease in precision of YAGO-N becomes smaller. Since the co-occurrence pairs set of 3 must contain that of 4, the precision of 4 cannot be larger than 3. Thus, 2 is a good choice for practice.

Exp-2: Efficiency (ApproxLoc). Using the same setting in Exp-1, we report the efficiency of ApproxLoc. Fig. 6(f) verifies that it is feasible to get conflict triples set for large KGs. On average, ApproxLoc outperforms ExactLoc and ExactLoc-Pu by up to 5.7×10^8 times and 2.4×10^6 times, respectively. It takes, on average, 45 seconds (resp. 14 minutes) to achieve near-optimal results by meet strategy and 50 seconds (resp. 13 minutes) by no-left strategy for NELL (resp. YAGO). ExactLoc and ExactLoc-Pu don't end within 2 weeks, even over a sampled graph at sample rates 1%, 5%, and 10%. So we reduce the amount of data with the number of rings and triples as $[2, 10]$, $[4, 20]$, $[7, 34]$, $[8, 38]$, $[9, 43]$, $[10, 47]$, $[11, 52]$, $[14, 63]$ and $[20, 88]$ to compare the efficiency among ExactLoc, ExactLoc-Pu and ApproxLoc in Fig. 6(f).

TABLE IV
ApproxLoc: EFFICIENCY (VARING λ)

Datasets	λ	# Triples	# Conflict Rings	Spend time(s)
NELL	2	3937	5774	17.7 + 1.9
	3	4279	11777	40.7 + 3.7
	4	4503	12545	43.3 + 3.2
YAGO	2	11862	168168	663 + 28.3
	3	11979	170697	734 + 27.6
	4	12070	171062	755 + 28.9

The spend time is the running time of the Selector plus the Filter.

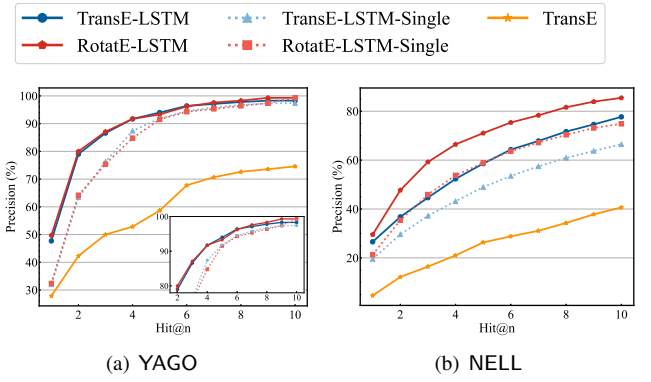
Varying # Hops (λ). Tab. IV shows the efficiency of *Selector* and *Filter* in ApproxLoc varying data size. λ means the size of sub-graphs used in the algorithms. The high connectivity of triples within small conflict rings in YAGO causes the data size to be basically the same when varying λ. Thus the

time consumption over YAGO does not change too much for different data sizes. With just 2 hops, the algorithm can cover most connected triples, and additional hops do not significantly expand the data size due to this dense interconnectivity. On the other hand, when varying λ from 2 to 3 over NELL, the time consumption doubled and the recall almost doubled, as shown in Fig. 6(c). This sharp increase is primarily because NELL contains only 16 small conflict rings, far fewer than YAGO's 436. Consequently, a minimum of 3 hops is necessary to find most of the connected triples, and the number of triples is doubled from 2 to 3, reflecting the even distribution of links throughout NELL.

Exp-3: Effectiveness (CorrectCon)**.** We next evaluate the effectiveness of conflict correction. We use Hit@n to measure the correction result. Hit@n is defined as the percentage of test conflict triples where the correct correction relation appears within the top n predictions made by the prediction model. As shown in Fig. 4, the result shows, (1) While TransE excels in generating embeddings, it performs less effectively in prediction with Hit@10 40% (resp. 75%) over NELL (resp. YAGO). By contrast, replacing the prediction model with an LSTM significantly enhances the precision. (2) RotatE-LSTM performs the best, with Hit@5 91.5% and Hit@10 97.7% (resp. 71.2% and 85.3%) over YAGO (resp. NELL). (3) The performance of all algorithms over NELL is worse than those over YAGO. This is because NELL has nearly twice as many relations as that of YAGO.

Varying # directions. We evaluated the necessity of using two rings with different directions for each prediction by comparing only one ring with the original direction used. The dashed lines in Fig. 4 show the results when only the original directional ring. The result shows that by using two rings from different directions, the precision increases about 15% (resp. 10%) for YAGO (resp. NELL). This means there are about 15% (resp. 10%) conflicts, which are direction+relation conflicts.

Exp-4: Case Analysis. The above experiments show that ApproxLoc and CorrectCon perform well for conflict location and correction. We next conducted two case studies to further demonstrate the ability of ApproxLoc and CorrectCon. Fig. 8 shows an example on NELL-995. We also provide an example from the above-mentioned industrial KG.



(a) YAGO (b) NELL

Fig. 7. CorrectCon: Effectiveness. The same color with different shades is for the same models, darker colors and solid lines indicate prediction results among two rings from different directions.

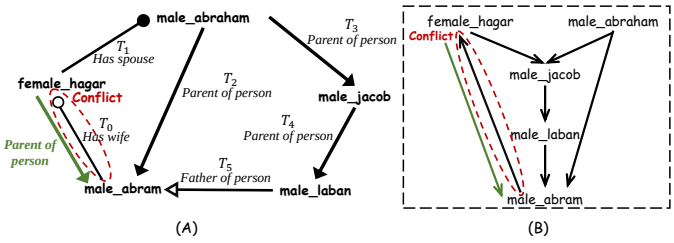


Fig. 8. Case Study on NELL-995. (A) shows the conflict in NELL-995 along with the correct triple. (B) is the kinship tree of (A).

Case 1: NELL-995. Note that these conflicts can not be detected by any baselines. In NELL-995, "parent" refers to other elders besides mother and father. Firstly, ApproxLoc finds that T_0 is a conflict triple as illustrated in Fig. 8. We then search the conflict rings set and find two conflict rings containing this triple. They are $R_1^{NELL} = \{T_0, T_1, T_2\}$ and $R_2^{NELL} = \{T_0, T_1, T_3, T_4, T_5\}$. Both R_1^{NELL} and R_2^{NELL} indicate 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 (entities), so T_0 may be a conflict triple. All conflict rings containing T_0 contain T_1, which meet the co-occurrence condition. Thus, we notice that T_1 is also in the result set. Secondly, CorrectCon finds that only changing the direction of T_1 cannot correct the relation. Thus, the type of this conflict is "direction + relation conflict". The correct triple is "(female_hagar, Parent of person, male_abram)", which changes both the direction and relation as the green arrow shows in Fig. 8. In this way, the semantics of this part of KG is correct.

Case 2: Industrial KG. ApproxLoc tells us that T_0 is a conflict triple. In Fig. 1, we find that there are 4 rings in this part of KG, and 3 of them are conflict, which is mentioned in Example. 3. There is a correct ring R_3. Thus, we find that the $SCR - CCR$ of T_0 is $\frac{3}{4} - 1 \leq 0$, satisfying Lemma. 4. In addition, the $SCR - CCR$ of T_2 equals $\frac{1}{4} - \frac{1}{3} \leq 0$, which is in the candidate set from *Selector*, but when searching in the sub-graph we input, we find R_3 is correct. Thus, (!analyze -v, Use instruction, !thread) is successfully filtered by *Filter*. Secondly, CorrectCon successfully corrects T_0 to (0x2m, Bug solution, Input "acp -r"), which means this is a "direction + relation conflict".

VII. CONCLUSION

We have formalized the conflict location and correction problem and shown that the location problem is NP-hard, which is, in general, intractable. We have developed a sub-graph generation process and an exact algorithm with pruning strategies to locate conflicts. In order to improve the efficiency, we further design an approximate algorithm ApproxLoc. Finally, we propose CorrectCon, a relation correction framework for conflict triples that uses embedding and prediction models. We experimentally verified the efficiency and effectiveness of the proposed algorithms. A case study is used to show that the proposed method can detect real conflicts. Future work includes model optimization for CorrectCon and a combination of previous error detection methods.

REFERENCES

[1] Google. Introducing the knowledge graph: things, not strings. https://blog.google/products/search/introducing-knowledge-graph-things-not/, 2012. Access: 2024-07.

[2] Dieter Fensel, Umutcan Şimşek, Kevin Angele, Elwin Huaman, Elias Kärle, Oleksandra Panasiuk, Ioan Toma, Jürgen Umbrich, et al. Introduction: what is a knowledge graph? *Knowledge graphs: Methodology, tools and selected use cases*, pages 1–10, 2020.

[3] Yuhan Sun et al. Demonstrating spindra: A geographic knowledge graph management system. In *IEEE ICDE*, pages 2044–2047, 2019.

[4] Akari Asai, Sewon Min, Zexuan Zhong, and Danqi Chen. Retrieval-based language models and applications. In *ACL*, pages 41–46, 2023.

[5] Yacheng He, Qianghuai Jia, Lin Yuan, Ruopeng Li, et al. A concept knowledge graph for user next intent prediction at alipay. *WWW*, 2023.

[6] Da Zheng, Xiang Song, Chao Ma, Zeyuan Tan, Zihao Ye, Jin Dong, Hao Xiong, Zheng Zhang, and George Karypis. Dgl-ke: Training knowledge graph embeddings at scale. In *ACM SIGIR*, pages 739–748, 2020.

[7] Bilal Abu-Salih. Domain-specific knowledge graphs: A survey. *Journal of Network and Computer Applications*, 185:103076, 2021.

[8] Duy Le, Kris Zhao, Mengying Wang, and Yinghui Wu. Graphlingo: Domain knowledge exploration by synchronizing knowledge graphs and large language models. In *IEEE ICDE*, pages 5477–5480, 2024.

[9] Haiying Liu, Ruizhe Ma, Daiyi Li, Li Yan, and Zongmin Ma. Machinery fault diagnosis based on deep learning for time series analysis and knowledge graphs. *JSPS*, 93:1433–1455, 2021.

[10] Huihui Han, Jian Wang, et al. Construction and evolution of fault diagnosis knowledge graph in industrial process. *TIM*, 71:1–12, 2022.

[11] Ruobing Xie, Zhiyuan Liu, Fen Lin, and Leyu Lin. Does william shakespeare really write hamlet? knowledge representation learning with confidence. In *AAAI/IAAI/EAAI*. AAAI Press, 2018.

[12] Shengbin Jia, Yang Xiang, et al. Triple trustworthiness measurement for knowledge graph. In *ACM WWW*, pages 2865–2871, 2019.

[13] Kewei Cheng, Xian Li, Yifan Ethan Xu, Xin Luna Dong, and Yizhou Sun. Pge: Robust product graph embedding learning for error detection. *VLDB*, 15(6):1288–1296, 2022.

[14] Wenfei Fan, Yinghui Wu, and Jingbo Xu. Functional dependencies for graphs. In *ACM SIGMOD*, pages 1843–1857, 2016.

[15] Wenfei Fan, Wenzhi Fu, Ruochun Jin, Muyang Liu, Ping Lu, and Chao Tian. Making it tractable to catch duplicates and conflicts in graphs. *ACM SIGMOD*, 1(1), 2023.

[16] Kashif Rabbani et al. Extraction of validating shapes from very large knowledge graphs. *VLDB*, 16(5):1023–1032, 2023.

[17] Kashif Rabbani, Matteo Lissandrini, and Katja Hose. Shactor: Improving the quality of large-scale knowledge graphs with validating shapes. In *ACM SIGMOD*, page 151–154, 2023.

[18] Full version. https://yolanda-w98.github.io/icde2025full.pdf.

[19] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. *NeurIPS*, 26, 2013.

[20] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *AAAI*, volume 29, 2015.

[21] Guoliang Ji, Shizhu He, Liheng Xu, et al. Knowledge graph embedding via dynamic mapping matrix. In *ACL/IJCNLP*, pages 687–696, 2015.

[22] Zhen Wang, Jianwen Zhang, Jianlin Feng, et al. Knowledge graph embedding by translating on hyperplanes. In *AAAI*, volume 28, 2014.

[23] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *ACM ICML*, pages 2071–2080. PMLR, 2016.

[24] Bishan Yang, Wen-tau Yih, Xiaodong He, et al. Embedding entities and relations for learning and inference in knowledge bases. *arXiv*, 2014.

[25] Sheng Guan, Hanchao Ma, Sutanay Choudhury, and Yinghui Wu. Gedet: Detecting erroneous nodes with a few examples. *VLDB*, 14(12), 2021.

[26] Qinggang Zhang, Junnan Dong, Keyu Duan, Xiao Huang, Yezi Liu, and Linchuan Xu. Contrastive knowledge graph error detection. In *ACM CIKM*, pages 2590–2599, 2022.

[27] Weijian Yu, Jie Yang, et al. Robust link prediction over noisy hyper-relational knowledge graphs via active learning. In *WWW*, 2024.

[28] Xu Chu, Ihab F Ilyas, and Paraschos Koutris. Distributed data deduplication. *VLDB*, 9(11):864–875, 2016.

[29] Mohammad Mahdavi, Ziawasch Abedjan, et al. Raha: A configuration-free error detection system. In *ACM SIGMOD*, pages 865–882, 2019.

[30] Alireza Heidari, Joshua McGrath, et al. Holodetect: Few-shot learning for error detection. In *ACM SIGMOD*, pages 829–846, 2019.

[31] Wenfei Fan, Chao Tian, Yanghao Wang, et al. Parallel discrepancy detection and incremental detection. *VLDB*, 14(8):1351–1364, 2021.

[32] Jiacheng Huang, Yao Zhao, Wei Hu, Zhen Ning, Qijin Chen, Xiaoxia Qiu, Chengfu Huo, and Weijun Ren. Trustworthy knowledge graph completion based on multi-sourced noisy data. In *WWW*, page 956–965. Association for Computing Machinery, 2022.

[33] Bingcong Xue and Lei Zou. Knowledge graph quality management: A comprehensive survey. *IEEE TKDE*, 35(5):4969–4988, 2023.

[34] Jinglun Cai, Mingda Li, Ziyan Jiang, Eunah Cho, Zheng Chen, Yang Liu, Xing Fan, and Chenlei Guo. Kg-eco: Knowledge graph enhanced entity correction for query rewriting. In *ICASSP*, pages 1–5, 2023.

[35] Arushi Raghuvanshi, Vijay Ramakrishnan, Varsha Embar, Lucien Carroll, and Karthik Raghunathan. Entity resolution for noisy ASR transcripts. In *EMNLP-IJCNLP*, pages 61–66, 2019.

[36] Haoyu Wang, John Chen, Majid Laali, Kevin Durda, Jeff King, William Campbell, and Yang Liu. Leveraging ASR n-best in deep entity retrieval. In *ISCA*, pages 261–265. ISCA, 2021.

[37] Xingwei He, Qianru Zhang, A-Long Jin, Jun Ma, Yuan Yuan, and Siu Ming Yiu. Improving factual error correction by learning to inject factual errors. In *AAAI*, volume 38, pages 18197–18205, 2024.

[38] Jiangjie Chen, Rui Xu, Wenxuan Zeng, Changzhi Sun, et al. Converge to the truth: Factual error correction via iterative constrained editing. In *AAAI*, volume 37, pages 12616–12625, 2023.

[39] Kun Yi and Jianxin Wu. Probabilistic end-to-end noise correction for learning with noisy labels. In *CVPR*, pages 7010–7018, 2019.

[40] Eric Arazo, Diego Ortego, Paul Albert, Noel O'Connor, and Kevin McGuinness. Unsupervised label noise modeling and loss correction. In *ACM ICML*, pages 312–321. PMLR, 2019.

[41] Yikai Zhang, Songzhu Zheng, Pengxiang Wu, Mayank Goswami, and Chao Chen. Learning with feature-dependent label noise: A progressive approach. In *ICLR*, 2020.

[42] Yangdi Lu and Wenbo He. Selc: Self-ensemble label correction improves learning with noisy labels. In *IJCAI*, pages 3278–3284, 2022.

[43] Seyed Mehran Kazemi and David Poole. Simple embedding for link prediction in knowledge graphs. *NIPS*, 31, 2018.

[44] Zhiqing Sun, Zhi-Hong Deng, et al. Rotate: Knowledge graph embedding by relational rotation in complex space. *arXiv*, 2019.

[45] Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and S Yu Philip. A survey on knowledge graphs: Representation, acquisition, and applications. *TNNLS*, 33(2):494–514, 2021.

[46] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2016.

[47] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, et al. Graph attention networks. In *ICLR*, 2018.

[48] Bohang Zhang, Shengjie Luo, et al. Rethinking the expressive power of gnns via graph biconnectivity. In *ICLR*, 2022.

[49] Zhanqiu Zhang et al. Rethinking graph convolutional networks in knowledge graph completion. In *WWW*, pages 798–807, 2022.

[50] Jaejun Lee, Chanyoung Chung, and Joyce Jiyoung Whang. Ingram: Inductive knowledge graph embedding via relation graphs. In *ACM ICML*, pages 18796–18809. PMLR, 2023.

[51] Wanxu Wei, Yitong Song, et al. Enhancing heterogeneous knowledge graph completion with a novel gat-based approach. *TKDE*, 2024.

[52] Xiaozhi Wang, Tianyu Gao, Zhaocheng Zhu, Zhengyan Zhang, Zhiyuan Liu, et al. Kepler: A unified model for knowledge embedding and pre-trained language representation. *TACL*, 9:176–194, 2021.

[53] Xintao Wang, Qianyu He, Jiaqing Liang, and Yanghua Xiao. Language models as knowledge embeddings. In *IJCAI*, pages 2291–2297, 2022.

[54] Peng Huang, Meihui Zhang, Ziyue Zhong, Chengliang Chai, and Ju Fan. Representation learning for entity alignment in knowledge graph: A design space exploration. In *IEEE ICDE*, pages 3462–3475, 2024.

[55] Stefano Marchesin and Gianmaria Silvello. Efficient and reliable estimation of knowledge graph accuracy. *VLDB*, 17(9):2392–2404, 2024.

[56] Sander Aarts and David B Shmoys. Hitting sets when the shallow cell complexity is small. In *International Workshop on Approximation and Online Algorithms*, pages 160–174. Springer, 2023.

[57] Farzaneh Mahdisoltani, Joanna Biega, and Fabian M Suchanek. Yago3: A knowledge base from multilingual wikipedias. In *CIDR*, 2013.

[58] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam Hruschka, and Tom Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, volume 24, pages 1306–1313, 2010.

APPENDIX

Proof of Lemma 1. If h and t are interconnected, 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 called "one ring". Consider any node k within the set of nodes N. It is guaranteed that a path exists from node k to entity t. Moreover, path E' represents the path from entity t to entity h. Therefore, a path must exist from node k to entity h. Finally, since k is a node in E, it is evident that a path exists from entity h to node k. Consequently, there must be a path from node k to itself, forming a ring.
- If $\exists f \in (e_1, e_2, \ldots, e_g), f \in (e'_1, e'_2, \ldots, e'_q)$, which can be called "multiple rings". 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, there is a ring 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 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, rings 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.

As the defination of conflict rings, if there are conflicts between h and t, there is at least one conflict triple in the paths pair. Then, the ring(s) containing this(these) conflict triple(s) is(are) conflict ring(s). Thus, if there is conflicts between h and t, there is at least one conflict ring among the ring(s) among the paths pair.

Proof of Theorem 2. We show a strong result that the conflict loca- tion problem is already NP-hard when the conflict condition is not considered, by a reduction from "vertex cover".

Let $G = (V, E)$ be a graph where V is the set of vertices and E is the set of edges. A vertex cover in this graph is a subset $V' \subseteq V$ such that for every edge $(u, v) \in E$, at least one of u or v is in V'.

We will construct an instance of the conflict location problem from this graph. Define the conflict location instance as follows:

- Let there be a set of conflict triples $H = \{T_v : v \in V\}$, where each vertex v in graph G corresponds to a triple T_v in H.
- Define a set of conflict rings $\mathcal{R} = \{R_{uv} : (u, v) \in E\}$, where each edge $(u, v) \in E$ corresponds to a ring $R_{uv} = \{T_u, T_v\}$ with two triples.

The objective in the conflict location problem is to find the H such that $|H| \leq b$ and for ever ring $R_i \in \mathcal{R}, R_i \cap H = \varnothing$.

Thus, (1) each vertex $v \in V'$ corresponds to a triple T_v in H, (2) each edge $(u, v) \in E$ corresponds to a ring $R_{uv} = \{T_u, T_v\}$ in $\mathcal{R}$.

Since Vertex Cover is NP-hard, and we can reduce Vertex Cover to the conflict location problem in polynomial time, the conflict location problem is also NP-hard. This hardness persists even without considering the conflict condition.

Proof of Lemma 4. Given a conflict triple $T_{conflict}$. The proof will be made from two categories. (To simplify the expression, we omit subscripts $T_{conflict}$ of all variables.)

- *Theoretically*, according to the second property of the conflict ring in Sec. II, all rings containing the conflict triples are conflict rings, which means $|ER| = |AR|$, and $CCR = 1$. Since $|AR| \leq |R|$, thus, $SCR \leq 1$. Hence, $SCR \leq CCR$.
- *Practically*, not all rings that contain conflict triples are correctly identified. This misidentification leads to $|ER| < |AR|$, thus, $CCR < 1$. We have the following:

$$SCR - CCR = \frac{|AR|}{|R|} - \frac{|ER|}{|AR|} = \frac{|AR| \times |AR| - |ER| \times |R|}{|R| \times |AR|} \quad (9)$$

  - Given that $0 \leq |ER| \leq |AR| \leq |R|$, $|R| \times |AR| > 0$.
  - Since $|ER|$ and $|AR|$ will be relatively close and $|AR| \leq |R|$, $|ER| \times (|AR| - |R|) \leq 0$.
  - This implies $|AR| \times |AR| - |ER| \times |ER| \leq 0$, reinforcing that $SCR - CCR \leq 0$.

Proof of Lemma 5. A correct triple $T_{correct}$ where no conflict is expected, thus will not cause the ring to be conflict, $|ER| = 0$. Thus, $CCR_{T_{correct}} = \frac{|ER|}{|AR|} = \frac{0}{|AR|} = 0$.

Since SCR is not related to conflict, thus $SCR_{T_{correct}} = \frac{|AR|}{|R|}$, $|AR| \leq |R|$. Thus, $SCR_{T_{correct}} \geq 0$.

To sum up, $SCR - CCR = SCR - 0 = SCR \geq 0$.

Description of Alg. 1. It uses conflict rings set as input, and outputs the optimal conflict triples set.

1) It first gets all triples $\mathcal{T}$ to generate all combinations $\mathcal{O}$, and each combination is a triples set. Because it aims to find one set to cover all rings in $\mathcal{R}$, it is easy to find that the upper bound of the set size is the number of conflict rings ($|\mathcal{R}|$). But, the number of triples ($|\mathcal{T}|$) can be less than $|\mathcal{R}|$, and at least $|\mathcal{R}| - |\mathcal{T}|$ (when $|\mathcal{R}| > |\mathcal{T}|$) triples appears in more than one ring. Thus, the upper bound of $|\mathcal{H}|$ is $\min(|\mathcal{T}|, |\mathcal{R}|)$ (lines 1-2).
2) To avoid repeated calculation, it calculates the $Metric$ ($SCR - CCR$)which is the same as the approximate algorithm based on SCR and CCR (lines 3-6).
3) It then verifies each triples set in $\mathcal{O}$ with each ring in $\mathcal{R}$, finds those sets that satisfy the cover condition, which is the intersection with each ring is not empty (lines 8-9). It then records those sets that satisfy the cover condition into a candidate set $\mathcal{H}'$ (lines 7-11).
4) Because it also needs to satisfy the conflict condition. It uses $Metric$ to find the triples set with the lowest $Metric$ in $\mathcal{H}'$ (lines 12-15). And the triples set with the lowest $Metric$ is the result of this algorithm.

Description of Alg. 2. The input and output of this algorithm are the same as Alg. 1.

1) The same as Alg. 1, it first finds all triples in the conflict rings set and get all combinations of triples (line 1 in Alg. 2). It then calculates the $Metric$ for every triple and initialize a dict to record them (lines 2-5 in Alg. 2).
2) Different from ExactLoc, ExactLoc-Pu uses $Metric$ to sort triples with the positive order at the begining (line 7 in Alg. 2). The lower the $Metric$ is, the higher the probability the triple is conflict.
3) It calls ExactLoc-Pu(BackTracking) to avoid unnecessary verifications and get the optimal result $\mathcal{T}$ by using the backtracking method (lines 8-10 in Alg. 2).
4) *First pruning*: It verifies the triples in positive order of $Metric$. It continues to add the next triple in positive order into the set until the chosen triples satisfy the cover condition. It compares $Metric$ and size with the current best one to trigger the backtracking procedure, because the triples in the positive order are worse than the previous. This avoids the increase of the set (lines 3-4 in Alg. 3).
5) *Second pruning*: When the size of $Current\_Triples$ is greater than the current best, the backtracking procedure returns. Because any addition will make the result worse (lines 5-6 in Alg. 3).
6) The return operation in the backtracking procedure is used to remove the current last element in $Current\_Triples$ and add the next element to $Current\_Triples$. But if $Current\_Triples$ doesn't trigger pruning, it then verifies the cover conditions, and when satisfied, it compares and records the current best one. It prunes with the best metric when it satisfies the cover condition, so it satisfies the conflict condition (lines 5-10 in Alg. 3).
7) It continues to expand $Current\_Triples$ when both the cover and pruning conditions are not met (lines 11-12 in Alg. 3).

Description of Alg. 4. The input and the output are also the conflict rings set and the conflict triples set, respectively.

1) It initializes set C_R and UC_R to record the current covered and uncovered rings in $\mathcal{R}$ (line 1). So, UC_R is initalized as $\mathcal{R}$ with all rings are not covered.

2) The same as the exact algorithms, it then get all triples in the conflict rings set to be selected (line 2).
3) It then computes SCR and CCR for each triple within conflict rings. It removes those where both SCR and CCR are 0, which are not in any rings and whose correctness cannot affect the semantic consistency (lines 2-4).
4) It then continuously selects a triple with the lowest $SCR - CCR$ among those with $SCR - CCR \leq 0$. This is the selection strategy of ApproxLoc to ensure this selected triple has the highest probability of being the conflict triple. It also updates the SCR and CCR for unselected triples, to reduce the chance of selecting frequent but conflict-free triples. This process continues until no unselected triples remain with $SCR - CCR \leq 0$, indicating that the remaining triples are correct following Lemma. 5 with No-left strategy (for meet strategy, change line 5 to "until meeting the cover condition"). Thus, it puts all conflict triples into the candidate set (lines 5-10). Because when one triple is selected, the R, $|AR|$ and $|ER|$ of each remained triple will change, thus, it updates $SCR - CCR$ for each remained triples (line 10). Lines 5 to 10 achieve the recall of 1.
5) It then finds a ring for each triple in the candidate set and verifies them. Correct triples are removed, which ensures the semantic consistency of each ring. This step results in the final set H (lines $11 - 14$).
6) It then finds the co-occurrence triple pair and labels them for every triple, in practice, it considers the number of co-occurrences up to η as co-occurrence triple pair (lines $15 - 18$).