KompaRe: A Knowledge Graph Comparative Reasoning System

Lihui Liu*, Boxin Du*, Yi Ren Fung*, Heng Ji*, Jiejun Xu†, Hanghang Tong*
*Department of Computer Science, University of Illinois at Urbana Champaign
†HRL Laboratories, LLC.
†jxu@hrl.com
*{lihuil2,boxindu2,yifung2,hengji,htong}@illinois.edu

ABSTRACT

Reasoning is a fundamental capability for harnessing valuable insight, knowledge and patterns from knowledge graphs. Existing work has primarily been focusing on point-wise reasoning, including search, link prediction, entity prediction, subgraph matching and so on. This paper introduces *comparative reasoning* over knowledge graphs, which aims to infer the commonality and inconsistency with respect to multiple pieces of clues. We envision that the comparative reasoning will complement and expand the existing point-wise reasoning over knowledge graphs. In detail, we develop KompaRe, the first of its kind prototype system that provides *comparative reasoning* capability over large knowledge graphs. We present both the system architecture and its core algorithms, including knowledge segment extraction, pairwise reasoning and collective reasoning. Empirical evaluations demonstrate the efficacy of the proposed KompaRe.

CCS CONCEPTS

• **Information systems applications** → **Data mining**; • **Artificial intelligence** → **Knowledge representation and reasoning**; • **Computing methodologies** → *Machine learning*.

ACM Reference Format:
Lihui Liu*, Boxin Du*, Yi Ren Fung*, Heng Ji*, Jiejun Xu†, Hanghang Tong*. 2021. KompaRe: A Knowledge Graph Comparative Reasoning System. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3447548.3467128

1 INTRODUCTION

Since its birth in 1980s [33] and especially its re-introduction by Google in 2012, knowledge graph has received more and more attentions, penetrating in a multitude of high-impact applications. To name a few, in fact checking, knowledge graph provides the vital background information about real-world entities and help a human fact checker corroborate or refute a claim [24]; in question answering, a question can be naturally formulated as a query graph, and the Q/A problem thus becomes the classic subgraph matching problem [12]; in recommender systems, knowledge graph offers the auxiliary information to improve the recommendation quality and/or explainability [36]; in computer vision, knowledge graph can be used to pre-optimize the model to boost its performance [8]. A fundamental enabling capability underlying these applications (and many more) lies in *reasoning*, which aims to identify errors and/or infer new conclusions from existing data [4]. The newly discovered knowledge through reasoning provides valuable input of these down stream applications, and/or can be used to further enrich the knowledge graph itself.

Most, if not all, of the existing work on knowledge graph reasoning belongs to the *point-wise* approaches, which perform reasoning w.r.t. *a single piece of clue* (e.g., a query). For example, in knowledge graph search [32], it returns the most relevant concepts for *a given entity*; in link prediction [14], given the 'subject' and the 'object' of *a triple*, it predicts the relation; in fact checking [22], given *a claim* (e.g., represented as a triple of the knowledge graph), it decides whether it is authentic or falsified; in subgraph matching [3], given *a query graph*, it finds exact or inexact matching subgraphs.

In this paper, we introduce *comparative reasoning* over knowledge graph, which aims to infer the commonality and/or the inconsistency with respect to multiple pieces of clues (e.g., multiple claims about a news article). We envision that the comparative reasoning will complement and expand the existing point-wise reasoning over knowledge graphs. This is because comparative reasoning offers a more complete picture w.r.t. the input clues, which in turn helps the users discover the subtle patterns (e.g., inconsistency) that would be invisible by point-wise approaches. Figure 1 gives an example to illustrate the power of comparative reasoning. Suppose there is a multi-modal news article and we wish to verify its truthfulness. To this end, two query graphs are extracted from the given news, respectively. One query graph contains all the information from the text, and the other contains the information from the image. If we perform point-wise reasoning to check each of these two query graphs *separately*, both seem to be true. However, if we perform reasoning w.r.t. both query graphs simultaneously, and by *comparison*, we could discover the subtle inconsistency between them (i.e., the different air plane types, the difference in maximum flying distances). In addition, comparative reasoning can also be used in knowledge graph expansion, integration and completion.

To be specific, we develop KompaRe, the first of its kind prototype system that provides comparative reasoning capability over large knowledge graphs. A common building block of comparative reasoning is *knowledge segment*, which is a small connection subgraph of a given clue (e.g., a triple or part of it) to summarize its semantic context. Based on that, we present core algorithms to enable both *pairwise* reasoning and *collective* reasoning. The key idea is to use influence function to discover a set of important elements in the knowledge segments. Then, the overlapping rate and the

KDD '21, August 14–18, 2021, Virtual Event, Singapore

ACM ISBN 978-1-4503-8332-5/21/08…$15.00
https://doi.org/10.1145/3447548.3467128

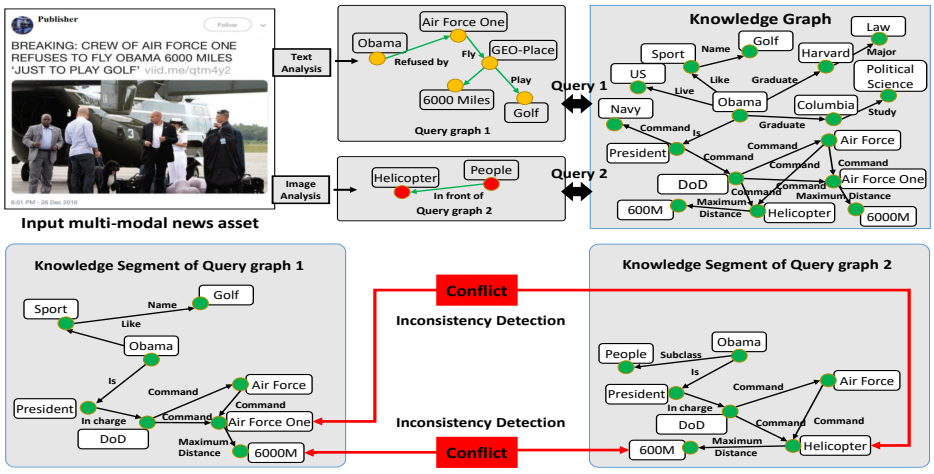


Figure 1: An illustrative example of using comparative reasoning for semantic inconsistency detection. Source of the image at the top-left: [6].

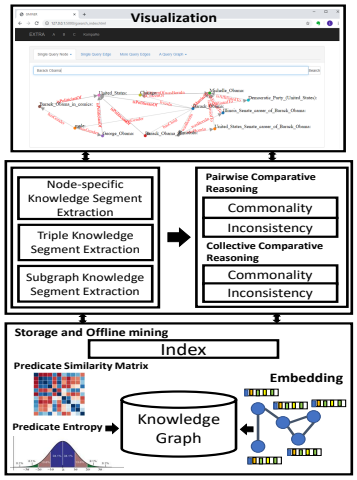


Figure 2: KompaRe architecture.

transferred information amount of these important elements will help reveal commonality and inconsistency.

The main contributions of the paper are

- **Problem Definition.** We introduce comparative reasoning over knowledge graphs, which complements and expands the existing point-wise reasoning capabilities.
- **Prototype Systems and Algorithms.** We develop the first of its kind prototype[1] for knowledge graph comparative reasoning, together with a suite of core enabling algorithms.
- **Empirical Evaluations.** We perform extensive empirical evaluations to demonstrate the efficacy of KompaRe.

2 KOMPARE OVERVIEW

A - Architecture and Main Functions. The architecture of KompaRe is shown in Figure 2 . Generally speaking, there are three key components in KompaRe, including (1) offline mining, (2) online reasoning and (3) User Interface (UI).

(1) Offline Mining. There are two main offline functions supported by KompaRe, including predicate entropy calculation and predicate-predicate similarity calculation.[2] These functions provide fundamental building blocks for KompaRe's online reasoning capabilities. For example, the predicate-predicate similarity will be used in both edge-specific knowledge segment extraction (Subsection 3.2) and subgraph-specific knowledge segment extraction (Subsection 3.3).

(2) Online Reasoning. In the online reasoning phase, KompaRe supports a variety of reasoning functions which are summarized in Table 1. First, it supports point-wise reasoning, which returns a small connection subgraph (referred to as 'knowledge segment' in this paper) for a single piece of clue provided by the user (f_1 to f_3 in Table 1). For example, if the given clue is a single entity, KompaRe finds a semantic subgraph to summarize the context of the given entity in the underlying knowledge graph; if the given clue is a single triple, KompaRe finds a connection subgraph to summarize the semantic proximity from the 'subject' of the triple to its 'object'; if the given clue is a subgraph, KompaRe finds a semantic matching subgraph where each edge of the query graph corresponds to a knowledge segment between the two matching nodes. Second, based on these point-wise reasoning functions, KompaRe further supports comparative reasoning (f_4 and f_5 in Table 1), which identifies both the commonality and the potential inconsistency w.r.t. multiple pieces of clues provided by the user. In addition, KompaRe also supports a number of common knowledge reasoning tasks, e.g., top-k query (i.e., given an entity, find the top-k most relevant entities), link prediction, subgraph matching, etc.

(3) User Interface (UI). KompaRe provides a user friendly interface to visualize the point-wise and/or comparative reasoning results. Basically, the interface supports three primary functions, including (i) function selection, where the user can select different kind of functions in Table 1 on the web page; (ii) query input, where the user can input various queries on the web page (e.g,. node, edge and query graph); and (iii) visualization, where KompaRe visualizes the reasoning results, and the user further modify their queries accordingly. The UI is implemented by HTML, Javascript and D3.js.

B - Key Challenges. There are several challenges to implement KompaRe which are listed below. First (C1 - challenge for point-wise reasoning), although there exists rich algorithms and tools to extract connection subgraphs on weighted graphs [10, 15, 30], they do not directly apply to knowledge graphs whose edges encode semantic relationship between different nodes. Second (C2 - challenges for comparative reasoning), different from *point-wise* reasoning which focuses on a single piece of clue, *comparative reasoning* aims to infer the commonality and/or the inconsistency w.r.t. multiple clues. Take knowledge graph based fact checking as an example. Even if each clue/claim could be true, we might still fail to detect the inconsistency between them without appropriately examining different clues/claims together. Third (C3 - scalability), a common challenge to both point-wise and comparative reasoning is how to support real-time or near real-time system response over large knowledge graphs.

3 KOMPARE BASICS

In this section, we introduce three basic functions in our KompaRe system, including `f1`, `f2` and `f3` in Table 1. These three functions,

[1]The developed methods have been integrated into a knowledge graph reasoning system [20]. A video demo can be found at https://github.com/lihuiliullh/KompaRe.
[2]KompaRe also contains other offline mining algorithms, e.g. TransE [2]. We omit the details of these algorithms due to the space limit.

Table 1: Summary of major functions in our system.

Name	Input	Output	Key techniques
f1	A single query node	A node-specific knowledge segment	Predicate entropy
f2	A single query edge	An edge-specific knowledge segment	Predicate-predicate similarity
f3	A query graph	A subgraph-specific knowledge segment	Semantic subgraph matching (*Edge-Table*)
f4	Two or more query edges	Commonality and inconsistency	Pairwise comparative reasoning (influence function, overlapping rate, transferred information)
f5	A query graph	Commonality and inconsistency	Collective comparative reasoning (influence function, overlapping rate, transferred information)

all of which belong to point-wise reasoning methods, form the basis of the comparative reasoning that will be introduced in the next section. Generally speaking, given a clue (e.g., a node, a triple or a query graph) from the user, we aim to extract a *knowledge segment* from the knowledge graph, which is formally defined as follows.

Definition 1. ***Knowledge Segment (KS for short)*** *is a connection subgraph of the knowledge graph that describes the semantic context of a piece of given clue (i.e., a node, a triple or a query graph).*

When the given clue is a node or an edge/triple, there exist rich algorithms to extract the corresponding knowledge segment[3] on weight graphs (e.g., a social network). To name a few, PageRank-Nibble [1] is an efficient local graph partition algorithm for extracting a dense cluster w.r.t. a seed node; K-simple shortest paths based method [10] or connection subgraph [7], [15] can be used to extract a concise subgraph from the source node of the querying edge to its target node. However, these methods do not directly apply to knowledge graphs because the edges (i.e., predicates) of a knowledge graph have specific semantic meanings (e.g., types, relations). To address this issue, we seek to convert the knowledge graph to a weighted graph by designing (1) a predicate entropy measure for node-specific knowledge segment extraction (Subsection 3.1), and (2) a predicate-predicate similarity measure for edge-specific knowledge segment extraction (Subsection 3.2), respectively.

When the given clue itself is a subgraph (Subsection 3.3), we propose to extract a *graph*. We would like to point out that semantic matching subgraph extraction is similar to but bears subtle difference from the traditional subgraph matching problem [21]. In subgraph matching, it aims to find a matching edge or path for each pair of matching nodes if they are required to be connected by the query graph; whereas in semantic subgraph matching, we aim to find a small connection subgraph (i.e., an edge-specific knowledge segment) for each pair of matching nodes that are required to be connected according to the query subgraph. In other words, a subgraph-specific knowledge segment consists of multiple interlinked edge-specific knowledge segments (i.e., one edge-specific knowledge segment for each edge of the input query subgraph). We envision that the subgraph-specific knowledge segment provides richer semantics, including both the semantics for each edge of the query graph and the semantics for the relationship between different edges of the input query graph. An example of semantic matching subgraph is given in the third column of Figure 3.

3.1 Node-specific Knowledge Segment

PageRank-Nibble [1] is a local graph partitioning algorithm to find a dense cluster near a seed node (i.e., the query node) on a weighted graph. It calculates the approximate PageRank vector with running time independent of the graph size. By sweeping over the PageRank vector, it finds a cut with a small conductance to obtain the local partition. In order to apply PageRank-Nibble to find node-specific knowledge segment, we propose to convert the knowledge graph into a weighted graph by *predicate entropy*.

To be specific, we treat each predicate in the knowledge graph as a random variable. The entropy of the predicates offers a natural way to measure its uncertainty and thus can be used to quantify the importance of the corresponding predicate. For example, some predicates have a high degree of uncertainty, e.g., `livesIn`, `isLocatedIn`, `hasNeighbor`, `actedIn`. This is because, in knowledge graph, different persons usually have different numbers of neighbors, and different actors may act in different movies. A predicate with high uncertainty indicates that it is quite common which offers little specific semantics of the related entity, and thus it should have low importance. On the other hand, some predicates have a low degree of uncertainty, e.g., `isPresident`, `isMarriedTo`. This is because only one person can be the president of a given country at a time, and for most of persons, they marry once in life. Such a predicate often provides very specific semantics about the corresponding entity and thus it should have high importance. Based on this observation, we propose to use predicate entropy to measure the predicate importance as follows.

We treat each entity and all the predicates surrounding it as the outcome of an experiment. In this way, we could obtain different distributions for different predicates. Let i denote a predicate in the knowledge graph, and D denote the maximal out-degree of a node. For a given node, assume it contains d out links whose label is i, we have $0 \leq d \leq D$. Let $\mathcal{V}_i^d$ denote the node set which contains d out links with label i, E denote the entropy, and $\mathbf{P}_i^d$ denote the probability of a node having d out links with label/predicate i. The entropy of a given predicate i can be computed as $\mathrm{E}(i) = \sum_{d=1}^{D} -\mathbf{P}_i^d \log(\mathbf{P}_i^d)$, where $\mathbf{P}_i^d = \frac{|\mathcal{V}_i^d|}{\sum_{d=1}^{D} |\mathcal{V}_i^d|}$. Finally, we compute the importance of a predicate i as $w(i) = 2\sigma(\frac{1}{\mathrm{E}(i)}) - 1$, where $\sigma()$ is the sigmoid function. We give an example in Appendix.

3.2 Edge-specific Knowledge Segment

Edge-specific knowledge segment extraction aims at finding a knowledge segment to best characterize the semantic context of the given edge (i.e., a triple). Several connection subgraph extraction methods exist for a weighted graph, e.g., [30], [15], [10]. We propose to use a TF-IDF based method[4] to measure the similarity between different predicates, and transfer the knowledge graph into a weighted graph whose edge weight represents the similarity

[3]It is worth pointing out that the extracted knowledge segment itself provides a powerful building block for several existing knowledge graph reasoning tasks, e.g. multi-hop method [11], minimum cost maximum flow method [24], etc.

[4]The TF-IDF based method was also used in [24] for computational fact checking.

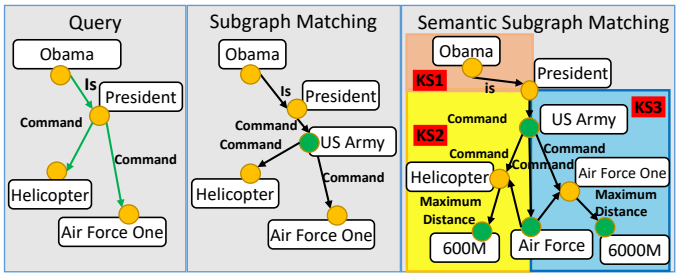


Figure 3: Difference between subgraph matching and semantic subgraph matching. For semantic subgraph matching, different color show different knowledge segments.

between the edge predicate and query predicate. Then, we find k-simple shortest paths [15] from the subject to the object of the given query edge as its knowledge segment.

The key idea behind predicate similarity is to treat each triple in the knowledge graph and its adjacent neighboring triples as a document, and use a TF-IDF like weighting strategy to calculate the predicate similarity. Consider a triple e_t = <s, `receiveDegreeFrom`, o> in the knowledge graph whose predicate is i = `receiveDegreeFrom`. In the neighborhood of e_t, there is a high probability that triples like <s, `major`, o> and <s, `graduateFrom`, o> also exist (adjacent to e_t). The predicates of these triples should have high similarity with each other. On the other hand, triples like <s, `livesIn`, o>, <s, `hasNeighbor`, o> may also occur in the adjacent neighborhood of triple e_t. This is because these predicates are very common in the knowledge graph, and occur almost everywhere. These predicates are like the stop words such as "the", "a", "an" in a document. Therefore, if we treat each predicate and its neighborhood as a document, we could use a TF-IDF like weighting strategy to find highly similar predicates and in the meanwhile penalize common predicates like `livesIn`, `hasNeighbor`.

To be specific, we use the knowledge graph to build a co-occurrence matrix of predicates, and calculate their similarity by a TF-IDF like weighting strategy as follows. Let i, j denote two different predicates. We define the TF between two predicates as $\text{TF}(i,j) = \log(1 + C(i,j)w(j))$, where $C(i,j)$ is the co-occurrence of predicate i and j. The IDF is defined as $\text{IDF}(j) = \log \frac{|M|}{|\{i:C(i,j)>0\}|}$, where M is the number of predicates in the knowledge graph. Then, we build a TF-IDF weighted co-occurrence matrix U as $U(i,j) = \text{TF}(i,j) \times IDF(j)$. Finally, the similarity of two predicates is defined as Sim(i, j) = Cosine(U_i, U_j), where U_i and U_j are the i^{th} row and j^{th} row of U, respectively. We give an example in Appendix.

3.3 Subgraph-specific Knowledge Segment

Given an attributed query graph Q={V^Q, E^Q, L^Q}, the traditional subgraph matching aims to find an edge or a path for each $e_i \in E^Q$. On the contrary, subgraph-specific knowledge segment extraction aims to find an edge-specific knowledge segment for each edge $e_i \in E^Q$. See Figure 3 for comparison.

To our best knowledge, there is no existing method for subgraph-specific knowledge segment extraction. We generalize a recent indexing-based subgraph matching algorithm called GFinder [21] in the following two aspects. First, we use the Core-Forest decomposition [9] in conjunction with root selection to minimize the query graph search diameter. Second, we design a new data structure called *Edge-Table* to index the knowledge segment between each pair of matching nodes to reduce memory cost. For each edge $e_i \in E^Q$, there is a corresponding *Edge-Table* to store the information of its corresponding knowledge segment, e.g., triples in the knowledge segments, the k-simple shortest paths and their costs. In order to find the edge-specific knowledge segments for each $e_i \in E^Q$, we again use the k-simple shortest path method to extract the paths with the lowest cost. The cost of a path is equal to the sum of the reciprocal of the predicate-predicate similarity of all edges in the path. Finally, all the edge-specific knowledge segments will be merged together to obtain the graph (i.e., the subgraph-specific knowledge segment).

Table 2: Notations and definition

Symbols	Definition
Q={V^Q, E^Q, L^Q}	an attributed query graph
G={V^G, E^G, L^G}	a knowledge graph
KS_i	knowledge segment i
A_i	adjacency matrix of KS_i
N_i	attribute matrix of KS_i, the j^{th} row denotes the attribute vector of node j in KS_i
$A_\times$	kronecker product of A_1 and A_2
N^l	diagonal matrix of the l^{th} node attribute
$N_\times$	combined node attribute matrix
$S^{i,j}$	single entry matrix $S^{i,j}(i,j) = 1$ and zeros elsewhere

4 KOMPARE COMPARATIVE REASONING

In this section, we introduce the technical details of comparative reasoning in KompaRe. We first introduce the pairwise reasoning (`f4` in Table 1) for two pieces of clues (e.g., two edges/triples), and then present the collective comparative reasoning (`f5` in Table 1). The main idea behind these two functions is that we use a knowledge segment to express the semantic meaning of each query triple, and use influence function to discover a set of important elements in the knowledge segments. Then, these important elements can be used to check the inconsistency. Table 2 summarizes the main notation used in this section.

4.1 Pairwise Comparative Reasoning

Pairwise comparative reasoning aims to infer the commonality and/or inconsistency with respect to a pair of clues according to their knowledge segments. Here, we assume that the two given clues are two edges/triples: E_1^Q =< $\mathsf{s}_1, \mathsf{p}_1, \mathsf{o}_1$ > and E_2^Q =< $\mathsf{s}_2, \mathsf{p}_2, \mathsf{o}_2$ > where $\mathsf{s}_1, \mathsf{o}_1, \mathsf{s}_2, \mathsf{o}_2 \in V^Q$ and $\mathsf{p}_1, \mathsf{p}_2 \in E^Q$. We denote their corresponding knowledge segments as KS_1 for E_1^Q and KS_2 for E_2^Q, respectively. The commonality and inconsistency between these two knowledge segments are defined as follows.

Definition 2. ***Commonality.*** *Given two triples (E_1^Q and E_2^Q) and their knowledge segments (KS_1 and KS_2), the commonality of these two triples refers to the shared nodes and edges between E_1^Q and E_2^Q, as well as the shared nodes and edges between KS_1 and KS_2:* $((V^{KS_1} \cap V^{KS_2}) \cup (V^{Q_1} \cap V^{Q_2}), (E^{KS_1} \cap E^{KS_2}) \cup (E^{Q_1} \cap E^{Q_2}))$.

Definition 3. ***Inconsistency.*** *Given two knowledge segments KS_1 and KS_2, the inconsistency between these two knowledge segments refers to any element (node, node attribute or edge) in KS_1 and KS_2 that contradicts with each other.*

In order to find out if these two given edges/triples are inconsistent, we first need to determine if they refer to the same/similar

thing/fact. Given a pair of clues $< s_1, p_1, o_1 >$ and $< s_2, p_2, o_2 >$, we divide it into the following six cases, including

C1. $s_1 \neq s_2$, $s_1 \neq o_2$, $o_1 \neq s_2$, $o_1 \neq o_2$. For this case, these two clues apparently refer to different things, e.g., <Alan Turing, wasBornIn, United Kingdom> and <Google, isLocatedIn, USA>.
C2. $s_1 = s_2$ and $o_1 = o_2$. If $p_1 = p_2$, these two clues are the same. If p_1 and p_2 are different or irrelevant, e.g., p_1 = wasBornIn, p_2 = hasWebsite, these two clues refer to different things. However, if p_1 contradicts p_2, they are inconsistent with each other.
C3. $s_1 = s_2$ but $p_1 \neq p_2$ and $o_1 \neq o_2$, e.g., <Alan Turing, wasBornIn, Maida Vale>, <Alan Turing, livesIn, United Kingdom>.
C4. $s_1 = s_2$, $p_1 = p_2$, but $o_1 \neq o_2$, e.g., <Alan Turing, wasBornIn, Maida Vale>, <Alan Turing, wasBornIn, United Kingdom>.
C5. $o_1 = o_2$, but $s_1 \neq s_2$. For this case, no matter what p_1 and p_2 are, these two clues refer to different things.
C6. $o_1 = s_2$. For this case, no matter what p_1 and p_2 are, they refer to different things. For example, <Alan Turing, wasBornIn, United Kingdom>, <United Kingdom, dealsWith, USA>.

Among these six cases, we can see that the clue pair in C1, C5 and C6 refer to different things. Therefore, there is no need to check the inconsistency between them. For C2, we only need to check the semantic meaning of their predicates, i.e., whether p_1 contradicts p_2. For example, p_1 = isFather and p_2 = isSon, they are inconsistent with each other. Otherwise, there is no inconsistency between them. We mainly focus on C3 and C4 where the two clues may be inconsistent with each other even if each of them is true. For example, either <Barack Obama, graduatedFrom, Harvard University> or <Barack Obama, majorIn , Political Science> could be true. But putting them together, they cannot be both true, since Barack Obama majored in law instead of Political Science when he studied at Harvard University. In other words, they are mutually exclusive with each other and thus are inconsistent. However, queries like <Alan Turing, wasBornIn, Maida Vale> and <Alan Turing, wasBornIn, United Kingdom> are both true, because Maida Vale belongs to United Kingdom. Alternatively, we can say that United Kingdom contains Maida Vale. Another example is <Alan Turing, wasBornIn, Maida Vale> and <Alan Turing, graduatedFrom, Princeton University>, both of which are true. Although they have the same subject, they refer to two different things. We summarize that if (1) the subjects of two clues are the same, and (2) their predicates are similar with each other or the same, they refer to the same thing. Furthermore, if their objects are two uncorrelated entities, it is highly likely that these two clues are inconsistent with each other.

Based on the above observations, we take the following three steps for pairwise comparative reasoning. First, given a pair of clues, we decide which of six cases it belongs to, by checking the subjects, predicates and objects of these two clues. Second, if this pair of clues belongs to C3 or C4, we need to decide whether they refer to the same thing or two different things. To this end, we first find a set of key elements (nodes or edges or node attributes) in these two knowledge segments. If most of these key elements belong to the commonality of these two knowledge segments, it is highly likely that they refer to the same thing. Otherwise, these two clues refer to different things. Third, if they refer to the same thing, we further decide whether they conflict with each other. Here, the key idea is as follows. We build two new query triples <o_1, isTypeOf, o_2> and <o_2, isTypeOf, o_1>. If one of them is true, the original two triples are consistent with each other. Otherwise, they are inconsistent.

In order to find the key elements, we propose to use the influence function w.r.t. the knowledge segment similarity [38]. The basic idea is that if we perturb a key element (e.g., change the attribute of a node or remove a node/edge), it would have a significant impact on the overall similarity between these two knowledge segments. Let KS_1 and KS_2 be the two knowledge segments. We can treat the knowledge segment as an attributed graph, where different entities have different attributes. We use random walk graph kernel with node attribute to measure the similarity between these two knowledge segments [38].

$$\text{Sim}(KS_1, KS_2) = q'_\times (I - cN_\times A_\times)^{-1} N_\times p_\times \tag{1}$$

where $q'_\times$ and $p_\times$ are the stopping probability distribution and the initial probability distribution of random walks on the product matrix, respectively. $N_\times$ is the combined node attribute matrix of the two knowledge segments $N_\times = \sum_{j=1}^{d} N_1^j \otimes N_2^j$ where N_i^j ($i \in \{1, 2\}$) is the diagonal matrix of the j^{th} column of attribute matrix N_i. $A_\times$ is the Kronecker product of the adjacency matrices of knowledge segments A_1 and A_2. $0 < c < 1$ is a parameter.

We propose to use the influence function of $\text{Sim}(KS_1, KS_2)$ w.r.t. knowledge segment elements $\frac{\partial Sim(KS_1, KS_2)}{\partial e}$, where e represents an element of the knowledge segment KS_1 or KS_2. The element with a high absolute influence function value is treated as a key element, and it can be a node, an edge, or a node attribute. Specifically, we consider three kinds of influence functions w.r.t. the elements in KS_1, including edge influence, node influence and node attribute influence, which can be computed according to the following lemma. Note that the influence function w.r.t. elements in KS_2 can be computed in a similar way, and thus is omitted for space.

Lemma 1. *(Knowledge Segment Similarity Influence Function [38].) Given $Sim(KS_1, KS_2)$ in Eq. (1). Let $Q = (I - cN_\times A_\times)^{-1}$ and $S^{j,i}$ is a single entry matrix defined in Table 2. We have that*
(1.) The influence of an edge $A_1(i, j)$ in KS_1 can be calculated as $I(A_1(i, j)) = \frac{\partial Sim(KS_1, KS_2)}{\partial A_1(i,j)} = cq'_\times QN_\times [(S^{i,j} + S^{j,i}) \otimes A_2] QN_\times p_\times$.
(2.) The influence of a node i in KS_1 can be calculated as $I(N_1(i)) = \frac{\partial Sim(KS_1, KS_2)}{\partial N_1(i)} = cq'_\times QN_\times [\sum_{j|A_1(i,j)=1} (S^{i,j} + S^{j,i}) \otimes A_2] QN_\times p_\times$.
(3.) The influence of a node attribute j of node i in KS_1 can be calculated as $I(N_1^j(i, i)) = \frac{\partial Sim(KS_1, KS_2)}{\partial N_1^j(i,i)} = q'_\times Q[S^{i,i} \otimes N_2^j](I + cA_\times QN_\times)p_\times$.

Note that according to Lemma 1, if an element only belongs to KS_1 or KS_2, its influence function value will be 0. In order to avoid this, we introduce a fully connected background graph to KS_1 and KS_2, respectively. This background graph contains all the nodes in KS_1 and KS_2, and it is disconnected with KS_1 and KS_2. If we treat KS_1 and KS_2 as two documents, we can think of this background graph as the background word distribution in a language model.

For a given knowledge segment, we flag the top 50% of the elements (e.g., node attribute, node and edge) with the highest absolute influence function values as key elements. We would like to check whether these key elements belong to the commonality of these two knowledge segments. If most of them (e.g., 60% or more) belong to the commonality of these two knowledge segments, we

say the two query clues describe the same thing. Otherwise, they refer to different things and thus we do not need to check the inconsistency between them.

If we determine that the query clues refer to the same thing, the next step is to decide whether they are inconsistent with each other. That is, given query clues <s_1, p_1, o_1> and <s_1, p_2, o_2>, we need to decide whether o_1 belongs to o_2 or o_2 belongs to o_1. To this end, we build two new queries <o_1, isTypeOf, o_2> and <o_2, isTypeOf, o_1>. Then, we extract the knowledge segments for these two queries, and check whether these two segments are true. If one of them is true, we say the original clues are consistent with each other, otherwise they are inconsistent. After we extract the knowledge segments for <o_1, isTypeOf, o_2> and <o_2, isTypeOf, o_1>, we treat each knowledge segment as a directed graph, and calculate how much information can be transferred from the subject to the object. We define the transferred information amount as:

$$\text{infTrans}(o_1, o_2) = \max_{1 \le j \le k} \text{pathValue}(j) \tag{2}$$

where pathValue(j) is defined as the multiplication of the weights in the path. For an edge, its weight is the predicate-predicate similarity Sim(isTypeOf, e_i). If $\max\{\text{infTrans}(o_1, o_2), \text{infTrans}(o_2, o_1)\}$ is larger than a threshold T, then we say o_1 belongs to o_2 or o_2 belongs to o_1. We set $T = 0.700$ in our experiment.

4.2 Collective Comparative Reasoning

Different from pairwise comparative reasoning, collective comparative reasoning aims to find the commonality and/or inconsistency inside a query graph which consists of a set of inter-connected edges/triples. We first give the corresponding definition below.

Definition 4. ***Collective Commonality.*** *For each edge E_i^Q in a query graph Q, let KS_i be its knowledge segment. The collective commonality between any triple pair in the query graph is the intersection of their knowledge segments.*

Definition 5. ***Collective Inconsistency.*** *For each edge E_i^Q in a query graph Q, let KS_i be its knowledge segment. The collective inconsistency refers to any elements (node or edge or node attribute) in these knowledge segments that contradict with each other.*

To check the inconsistency, one naive method is using the pairwise comparative reasoning method to check the inconsistency for each pair of edges in the query graph. However, this method is neither computationally efficient nor sufficient. For the former, if two clues (e.g., two claims from a news article) are weakly or not related with each other on the query graph, we might not need to check the inconsistency between them at all. For the latter, in some subtle situation, the semantic inconsistencies could *only* be identified when we collectively reason over multiple (more than two) knowledge segments. For example, given the following three claims, including (1) *Obama is refused by Air Force One*; (2) *Obama is the president of the US*; (3) *The president of US is in front of a helicopter*. Only if we reason these three claims collectively, can we identify the semantic inconsistency among them.

Based on the above observation, we propose the following method to detect the collective inconsistency.

First, we find a set of key elements inside the semantic matching subgraph. Different from pair-wise comparative reasoning, the importance/influence of an element for collective comparative reasoning is calculated by the entire semantic matching subgraph. More specifically, we first transform the query graph and its semantic matching subgraph (i.e., subgraph-specific knowledge segment) into two line graphs, which are defined as follows.

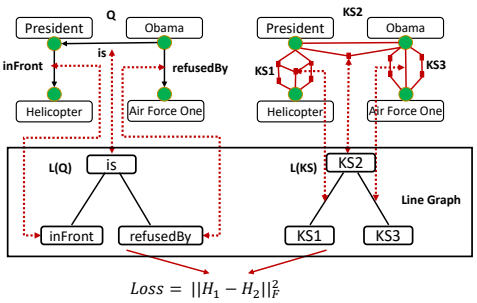


Figure 4: Collective comparative reasoning workflow.

Definition 6. **Line Graph [24].** *For an arbitrary graph $G = (V, E)$, the line graph $L(G) = (V', E')$ of G has the following properties: (1) the node set of $L(G)$ is the edge set of G ($V' = E$); (2) two nodes V_i', V_j' in $L(G)$ are adjacent if and only if the corresponding edges e_i, e_j of G are incident on the same node in G.*

Figure 4 gives an example of the line graph. For the line graph $L(Q)$, the edge weight is the predicate-predicate similarity of the two nodes it connects. For the line graph $L(KS)$, the edge weight is the knowledge segment similarity by Eq. (1) of the two nodes it connects. The rationality of building these two line graphs is that if the semantic matching subgraph is a good representation of the original query graph, the edge-edge similarity in $L(Q)$ would be similar to the knowledge segment similarity in $L(KS)$.

To measure the importance of an element, we propose to use the influence function w.r.t. the distance between $L(Q)$ and $L(KS)$. We assume that a key element, if perturbed, would have a great effect on the distance Loss $= ||H_1 - H_2||_F^2$, where H_1 is the weighted adjacency matrix of $L(Q)$, and H_2 is the weighted adjacency matrix of $L(KS)$. We use the influence function $\frac{\partial \text{Loss}(H_1, H_2)}{\partial e}$, where e represents an element of the knowledge segment graph and it could be a node, an edge, or a node attribute. Lemma 2 provides the details on how to compute such influence functions. The proof of Lemma 2 is shown in Appendix.

Lemma 2. *Given the loss function Loss $= ||H_1 - H_2||_F^2$. Let n, k denote two different nodes in $L(Q)$, and KS_n, KS_k denote their corresponding knowledge segments. Let $h_{e_{k,n}}$ denote the weight of edge between node k and n, and $h_{c_{k,n}}$ denote the weight of edge between KS_k and KS_n. We have*
(1.) The influence of an edge $A_n(i,j)$ in knowledge segment KS_n can be calculated as $I(A_n(i,j)) = \sum_{k \in N(n)} -2(h_{e_{k,n}} - h_{c_{k,n}}) \frac{\partial sim(KS_n, KS_k)}{\partial A_n(i,j)}$.
(2.) The influence of a node i in knowledge segment KS_n can be calculated as $I(N_n(i)) = \sum_{k \in N(n)} -2(h_{e_{k,n}} - h_{c_{k,n}}) \frac{\partial sim(KS_n, KS_k)}{\partial N_n(i)}$.
(3.) The influence of a node attribute j in knowledge segment KS_n can be calculated as $I(N_n^j(i,i)) = \sum_{k \in N(n)} -2(h_{e_{k,n}} - h_{c_{k,n}}) \frac{\partial sim(KS_n, KS_k)}{\partial N_n^j(i,i)}$.

Second, after we find all the key elements, we check the consistency of the semantic matching subgraph according to these key elements. The steps are as follows. For each pair of knowledge segments of the semantic matching subgraph, if their key elements overlapping rate is greater than a threshold (60%), we check the consistency of this pair. Suppose the corresponding triples are <s_1, p_1,

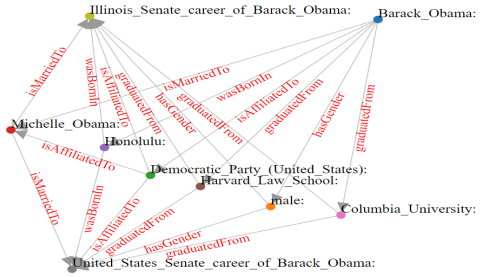


Figure 5: Node-specific knowledge segment of `Barack Obama`.

$o_1>$ and $<s_2, p_2, o_2>$, respectively. We check if $<s_1$, `isTypeOf`, $s_2>$ or $<s_2$, `isTypeOf`, $s_1>$ is true. If both of them are false, we skip this pair of clues because it does not belong to C3 or C4. Otherwise, we check if $<o_1$, `isTypeOf`, $o_2>$ or $<o_2$, `isTypeOf`, $o_1>$ is true. If both of them are false, we say this query graph has collective inconsistency. When checking the truthfulness of triples (e.g., $<s_1$, `isTypeOf`, $s_2>$, $<s_2$, `isTypeOf`, $s_1>$, $<o_1$, `isTypeOf`, $o_2>$ and $<o_2$, `isTypeOf`, $o_1>$), we use the same method (i.e., transferred information amount in Eq. (2)) as in pairwise comparative reasoning.

5 EXPERIMENTAL RESULTS

In this section, we present the experimental evaluations. All the experiments are designed to answer the following two questions:

- **Q1. Effectiveness.** How effective are the proposed reasoning methods, including both point-wise methods (KompaRe basics) and comparative reasoning methods?
- **Q2. Efficiency.** How fast are the proposed methods?

Two data graphs are used in the experiments: Yago [28] [5] and Covid-19 [6]. Yago [28] is a widely used knowledge graph which contains 12,430,705 triples, 4,295,825 entities and 39 predicates. The Covid-19 data graph contains three types of entities which are `Gene`, `Disease` and `Chemical`. In our experiments, we use a subset of the Covid-19 dataset which contains 55,434 core entities and 5,527,628 triples. We compare our method to two existing algorithms for fact checking (Knowledge Linker [5], short for KL, and KGMiner [25]), one link prediction algorithm (Jaccard coefficient [18]), and one algorithm for knowledge graph completion (TransE [2]). For TransE [2], we set the embedding dimension to 64 and use a margin of one and a learning rate of 0.01 for 1,200 epochs.

All the experiments are conducted on a moderate desktop with an Intel Core-i7 3.00GHz CPU and 64GB memory. The system was deployed in November 17 2020. The source code will be released upon publication of the paper. [7]

5.1 KompaRe Basics

We evaluate the proposed predicate-predicate similarity. The top similar predicates w.r.t. `exports` by our method include `imports`, `hasOfficialLanguage`, `dealsWith`, all of which have a high similarity with `exports`. They all provide specific semantic information about `exports`. Likewise, the top similar predicates w.r.t. `livesIn` include `wasBornIn`, `isCitizenOf`, `diedIn`, all of which are closely related to `livesIn`. These results show that the proposed TF-IDF based method can effectively measure the similarity between different predicates.

Table 3: Accuracy of pair-wise comparative reasoning.

Dataset	# of queries	TransE	Jaccard	KL	KGMiner	Kompare
Family members positive	300	0.682	0.831	0.618	**0.983**	0.944
Family members negative	300	0.335	0.169	**1.000**	**1.000**	0.941
Graduated college positive	300	0.686	0.335	0.502	0.769	**0.794**
Graduated college negative	300	0.626	0.993	0.947	0.901	**0.994**
Live place positive	300	0.567	0.415	0.489	**0.834**	0.762
Live place negative	300	0.802	0.585	**0.907**	0.900	0.888
Birth place positive	300	0.590	0.435	0.537	0.698	**0.800**
Birth place negative	300	0.845	**1.000**	0.973	0.927	0.927
Work place positive	300	**0.751**	0.319	0.445	0.698	0.720
Work place negative	300	0.624	0.994	0.942	0.927	**0.995**
mean ± *std*	-	0.651 ± 0.424	0.608 ± 0.302	0.736 ± 0.221	0.864 ± 0.105	**0.877 ± 0.095**

Figure 5 shows the node-specific knowledge segment found by our method w.r.t. the query node `Barack Obama`. We can see that the extracted knowledge segment provides critical semantics of the query node `Barack Obama`. For example, `Barack Obama` graduated from `Harvard Law School` and `Columbia University`; his wife is `Michelle Obama`; he is affiliated to the democratic party; he was the senator of united states and he was born in `Honolulu`. The result shows that node-specific knowledge segment is able to capture semantic context of the query node/entity.

5.2 Pair-wise Comparative Reasoning

Here, we evaluate the effectiveness of the proposed pair-wise comparative reasoning. Ten query sets are used in the experiments. For each positive query set, it contains a set of queries which describe the true claim, while for each negative query set, it contains a set of queries which describe the false claim. For example, in query set "Birth Place", <`Alan Turing`, `wasBornIn`, `Maida Vale`> and <`Alan Turing`, `wasBornIn`, `United Kingdom`> is a positive query pair, while <`Alan Turing`, `wasBornIn`, `Maida Vale`> and <`Alan Turing`, `wasBornIn`, `Canada`> is an negative query pair. The positive queries are generated by sampling some true claims in the knowledge graph. The negative queries are generated by substituting one subject of the positive queries. The accuracy is defined as $\frac{N}{M}$ where N is the number of queries correctly classified by pair-wise comparative reasoning and M is the total number of queries. When checking the consistency of a query pair $< s_1, p_1, o_1 >$ and $< s_2, p_2, o_2 >$, because none of the baseline methods is designed for pair-wise comparative reasoning, we use them to check each triple in the pair, if any triple is classified as false, this query pair is treated as false. Otherwise, we further check the truthness of $< o_1$, `isTypeOf`, $o_2 >$ and $< o_2$, `isTypeOf`, $o_1 >$, if one of them is classified as consistency, this query pair is treated as consistency. Table 3 gives the detailed results. As we can see, KompaRe and KGMiner [25] have the highest accuracy most of the time. But KompaRe has the highest average accuracy and the lowest variance compared with other methods.

5.3 Collective Comparative Reasoning

We test collective comparative reasoning method on Yago dataset, using 6 query sets . Different from the queries of pair-wise comparative reasoning which only contain two edges, each query of collective comparative reasoning contains 3 edges. For example, in query

[5] It is publicly available at https://www.mpi-inf.mpg.de/de/departments/databases-and-information-systems/research/yago-naga/yago/downloads. We use the core version.

[6] The dataset can be found at http://blender.cs.illinois.edu/covid19/.

[7] The developed methods have been integrated into a knowledge graph reasoning system [20]. The video demonstration of the system can be found at https://github.com/lihuiliullh/KompaRe.

Table 4: Accuracy of collective comparative reasoning.

Dataset	# of queries	TransE	Jaccard	KL	KGMiner	Kompare
Birth place positive	300	0.542	0.418	0.389	0.678	**0.795**
Birth place negative	300	0.465	**0.996**	0.968	0.970	0.829
Live place positive	300	0.448	0.451	0.465	0.635	**0.989**
Live place negative	300	0.558	**1.000**	0.860	0.924	0.743
Graduated college positive	300	0.488	0.269	0.335	0.585	**0.963**
Graduated college negative	300	0.545	**0.996**	0.928	0.907	0.829
$mean \pm std$	-	0.508 ± **0.045**	0.688 ± 0.313	0.658 ± 0.265	0.783 ± 0.155	**0.858** ± 0.089

Table 5: Accuracy of collective comparative reasoning for Covid-19.

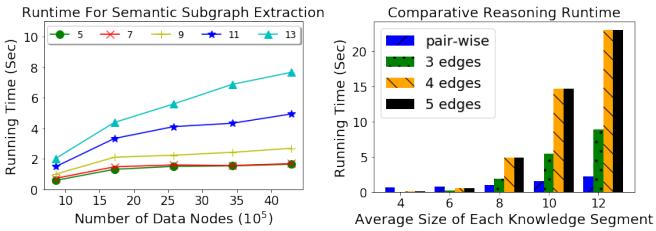
Dataset	# of queries	TransE	Jaccard	KL	KGMiner	Kompare
Positive	36	0.667	0.611	**1.000**	0.694	**1.000**
Negative	36	0.528	0.361	0.722	0.553	**0.863**
Average accuracy	-	0.598 ± 0.071	0.486 ± 0.126	0.861 ± 0.138	0.623 ± 0.071	**0.932 ± 0.063**

set "live Place", <Barack Obama, livesIn, Washington,D.C.>, <Barack Obama, is, United States Senate Barack Obama> and <United States Senate Barack Obama, livesIn, United States> is a positive query triad, while <Barack Obama, livesIn, Washington,D.C.>, <Barack Obama, is, United States Senate Barack Obama> and <United States Senate Barack Obama, livesIn, Canada> is an negative query triad. The definition of the accuracy is the same as the previous section. Following the setting of pair-wise reasoning, when checking the consistency of the query graph $< s_1, p_1, o_1 >$, $< s_1, is, s_2 >$ and $< s_2, p_2, o_2 >$, we use baseline methods to check the truthness of this query triad, if any edge is classified as false, this query triad is treated as false. Otherwise, we further check the truthness of $< o_1, isTypeOf, o_2 >$ and $< o_2, isTypeOf, o_1 >$, if one of them is classified as true , this query pair is treated as consistency. Table 4 gives the detailed results. As we can see, Jaccard [18] prefers to classify all queries as inconsistency and has the largest variance. TransE [2] has the lowest variance, but its average accuracy is very low. KompaRe has the highest accuracy most of the time. It also has the highest average accuracy, and the second lowest variance.

We further provide experimental results on Covid-19 dataset. We use queries which contain connections between drugs and genes/chemicals related to covid-19.[8] Among all these queries, we use queries which contain less than 8 nodes, and treat them as positive queries. For each of the positive queries, we randomly select one node inside the query and substitute it with a randomly selected entity in the data graph, and treat the new query as the negative query. For all the baseline methods, we use them to check all the edges inside the query, if any edge is classified as false, the whole query is treated as false. Table 5 shows the accuracy of different methods. As we can see, KompaRe has the highest accuracy on both the positive and negative datasets, it also has the highest average accuracy and the lowest variance compared with other baseline methods.

5.4 KompaRe Efficiency

The runtime of knowledge segment extraction depends on the size of the underlying knowledge graphs. Among the three types of knowledge segments (f1, f2 and f3 in Table 1), subgraph-specific knowledge segment (f3) is most time-consuming. Figure 6(a) shows that its runtime scales sub-linearly w.r.t. the number of nodes in the knowledge graph. Different lines show the runtime w.r.t. different query graph size. Figure 6(b) shows the runtime of comparative reasoning, where 'Pair-wise' refers to the pairwise comparative reasoning, and the remaining bars are for collective comparative reasoning with 3, 4 and 5 edges in the query graphs respectively. Note that the runtime of comparative reasoning only depends on the size of the the corresponding knowledge segments which typically have a few or a few tens of nodes. In other words, the runtime of comparative reasoning is *independent* of the knowledge graph size.



(a) Subgraph-specific KS extraction **(b) Comparative reasoning**

Figure 6: Runtime of KompaRe

6 RELATED WORK

A - Knowledge Graph Search. Many efforts have been made for searching and browsing large knowledge graphs. Wang et al. [32] proposed a Bayesian probability model combined with random walks to find the most similar concepts for a given query entity. Wu et al. [27] discovered that the background knowledge graph can be described by many small-sized patterns. They developed an effective mining algorithm to summarize the large knowledge graph according to small-sized patterns. Yang et al. [35] found that due to the lack of insight about the background knowledge graph, it is often hard for a user to precisely formulate a query. They developed a user-friendly knowledge graph search engine to support query formation and transformation. Jayaram et al. [13] proposed a knowledge graph query system called *GQBE*. Different from other graph query systems, *GQBE* focuses on entity tuple query which consists of a list of entity tuples. Zhang et al. [37] developed a comprehensive multi-modality knowledge extraction and hypothesis generation system which supports three types of queries, including (1) class-based queries (2) zero-hop queries and (3) graph-queries.

B - Knowledge Graph Reasoning. Generally speaking, there are two types of knowledge graph reasoning methods, including (1) embedding based approaches and (2) multi-hop approaches. For the former, the main idea is to learn a low dimensional vector for each entity and predicate in the embedding space, and use these embedding vectors as the input of the reasoning tasks (e.g., [2], [29], [16], [31]). For the latter, the main idea is to learn missing rules from a set of relational paths sampled from the knowledge graph (e.g., [17], [34], [23]). Many effective reasoning methods have been developed for predicting the missing relation (i.e., link prediction) or the missing entity (i.e., entity prediction). In link prediction, given the 'subject' and the 'object' of *a triple*, it predicts the existence and/or the type of relation. For example, *TransE* [2] learns the low dimensional embedding of both entities and predicates in the knowledge graph; *TransR* [19] learns the embedding of entities and predicates in two separate spaces. The learned embedding (either by *TransE* or *TransR*) can be used for both link predication and

[8]The query graphs can be found at http://blender.cs.illinois.edu/covid19/visualization.html.

entity predication. In entity prediction, given the 'subject' and the 'predicate' of *a triple*, it predicts the missing 'object'. For example, *GQEs* [16] embeds the graph nodes in a low dimensional space, and treats the logical operators as learned geometric operations.

In recent years, knowledge graph reasoning has demonstrated strong potential for computational fact checking. Given *a claim* in the form of a triple of the knowledge graph, it reasons whether the claim is authentic or falsified. For example, in [24], the authors focused on checking the truthfulness of a given triple/claim, by first transforming the knowledge graph into a weighted directed graph, and then extracting a so-called knowledge stream based on maximum flow algorithm. It is worth mentioning that the extracted knowledge stream can be viewed as an edge-specific knowledge segment in KompaRe. In [26], an alternative method was developed to detect fake claims by learning the discriminative paths of specific predicates. Different from [24], this is a supervised reasoning method since it requires different training datasets for different predicates. If the predicate in the claim does not exist in the training data, which is likely to be the case for detecting falsified claims in emerging news, the algorithm becomes inapplicable. As mentioned before, these methods belong to point-wise reasoning. Therefore, they might fall short in detecting the semantic inconsistency between multiple claims which can be solved by knowledge graph comparative reasoning.

7 CONCLUSIONS

In this paper, we present a prototype system (KompaRe) for knowledge graph comparative reasoning. KompaRe aims to complement and expand the existing point-wise reasoning over knowledge graphs by inferring commonalities and inconsistencies of multiple pieces of clues. The developed prototype system[9] consists of three major components, including its UI, online reasoning and offline mining. At the heart of the proposed KompaRe are a suite of core algorithms, including predicate entropy, predicate-predicate similarity and semantic subgraph matching for knowledge segment extraction; and influence function, commonality rate, transferred information amount for both pairwise reasoning and collective reasoning. The experimental results demonstrate that the developed KompaRe (1) can effectively detect semantic inconsistency, and (2) scales near linearly with respect to the knowledge graph size.

8 ACKNOWLEDGEMENT

This work is supported by National Science Foundation under grant No. 1947135, by the United States Air Force and DARPA under contract number FA8750-17-C-0153 [10], and IBM-ILLINOIS Center for Cognitive Computing Systems Research (C3SR) - a research collaboration as part of the IBM AI Horizons Network. The content of the information in this document does not necessarily reflect the position or the policy of the Government or Amazon, and no official endorsement should be inferred. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

[9]The developed methods have been integrated into a knowledge graph reasoning system [20]. A video demo can be found at https://github.com/lihuiliullh/KompaRe.
[10]Distribution Statement "A" (Approved for Public Release, Distribution Unlimited)

REFERENCES

[1] R. Andersen, F. Chung, and K. Lang. 2006. In *2006 47th Annual IEEE FOCS 06.*
[2] B. Antoine, U. Nicolas, G. Alberto, W Jason, and Y. Oksana. [n. d.]. Translating Embeddings for Modeling Multi-relational Data. In *NIPS '13*. 2787–2795.
[3] N Cao B Du, S Zhang and H Tong. 2017. FIRST: Fast Interactive Attributed Subgraph Matching. In *KDD'17.*
[4] X. Chen, S. Jia, and Y. Xiang. 2020. A review: Knowledge reasoning over knowledge graph. *Expert Systems with Applications* 141 (2020), 112948.
[5] Giovanni Luca Ciampaglia, Prashant Shiralkar, and Rocha. 2015. Computational Fact Checking from Knowledge Networks. (2015).
[6] Limeng Cui, Suhang Wang, and Dongwon Lee. 2019. SAME : Sentiment-Aware Multi-Modal Embedding for Detecting Fake News.
[7] C. Faloutsos, K. McCurley, and A. Tomkins. 2004. Fast Discovery of Connection Subgraphs. In *KDD '04.* ACM, New York, NY, USA, 118–127.
[8] Y. Fang, K. Kuan, J. Lin, C. Tan, and V. Chandrasekhar. 2017. Object Detection Meets Knowledge Graphs *(IJCAI-17).* https://doi.org/10.24963/ijcai.2017/230
[9] B. Fei, L. Chang, X. Lin, and L. Qin. 2016. Efficient Subgraph Matching by Postponing Cartesian Products *(SIGMOD '16).* 1199–1214.
[10] S. Freitas, N. Cao, Y. Xia, D. H. P. Chau, and H. Tong. 2018. Local Partition in Rich Graphs *(BigData '19).* 1001–1008. https://doi.org/10.1109/BigData.2018.8622227
[11] C. Giovanni, S. Prashant, R. Luis, B. Johan, M. Filippo, and F. Alessandro. 2015. Computational Fact Checking from Knowledge Networks. *PloS one* 10 (01 2015).
[12] S. Hu, L. Zou, J. X. Yu, H. Wang, and D. Zhao. 2018. Answering Natural Language Questions by Subgraph Matching over Knowledge Graphs. 30, 5 (May 2018).
[13] N. Jayaram, A. Khan, C. Li, X. Yan, and R. Elmasri. 2015. Querying Knowledge Graphs by Example Entity Tuples. 27, 10 (Oct 2015), 2797–2811.
[14] S. Mehran Kazemi and P. David. 2018. SimplE Embedding for Link Prediction in Knowledge Graphs. In *Advances in Neural Information Processing Systems 31.*
[15] Y. Koren, S. North, and C. Volinsky. 2006. Measuring and Extracting Proximity in Networks *(KDD '06).* ACM, New York, NY, USA, 245–255.
[16] H. William L., B. Payal, Z. Marinka, J. Dan, and L. Jure. 2018. Embedding Logical Queries on Knowledge Graphs *(NIPS'18).* Red Hook, NY, USA, 2030–2041.
[17] Ni L, Tom M, and William W. C. 2011. Random Walk Inference and Learning in a Large Scale Knowledge Base *(EMNLP '11).* USA, 11.
[18] David Liben-Nowell and Jon Kleinberg. [n. d.]. The Link Prediction Problem for Social Networks *(CIKM '03).*
[19] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu. 2015. Learning Entity and Relation Embeddings for Knowledge Graph Completion *(AAAI'15).* AAAI Press, 7.
[20] Lihui Liu, Boxin Du, Heng Ji, and Hanghang Tong. 2020. A Knowledge Graph Reasoning Prototype. *NeurIPS (demo track)* (2020).
[21] L. Liu, B. Du, and H. Tong. 2019. GFinder: Approximate Attributed Subgraph Matching *(BigData '19).*
[22] H. Naeemul, A. Fatma, and C. Li. 2017. Toward Automated Fact-Checking: Detecting Check-Worthy Factual Claims by ClaimBuster *(KDD '17).* 10.
[23] D. Rajarshi, N. Arvind, B. David, and M. Andrew. 2017. Chains of Reasoning over Entities, Relations, and Text using Recurrent Neural Networks *(ACL '17).*
[24] Prashant S, Alessandro F, Filippo M, and Giovanni C. 2017. Finding Streams in Knowledge Graphs to Support Fact Checking. 859–864.
[25] Baoxu Shi and Tim Weninger. [n. d.]. Discriminative Predicate Path Mining for Fact Checking in Knowledge Graphs. ([n. d.]).
[26] B. Shi and T. Weninger. 2016. Discriminative Predicate Path Mining for Fact Checking in Knowledge Graphs. *Know.-Based Syst.* 104, C (July 2016), 123–133.
[27] Q. Song, Y. Wu, P. Lin, L. X. Dong, and H. Sun. 2018. *IEEE Transactions on Knowledge and Data Engineering* 30, 10 (Oct 2018), 1887–1900.
[28] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: A Core of Semantic Knowledge *(WWW '07).* Association for Computing Machinery, 10.
[29] Z. Sun, Z. Deng, J. Nie, and J. Tang. 2019. RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space. *ArXiv* abs/1902.10197 (2019).
[30] Hanghang Tong and Christos Faloutsos. 2006. Center-piece Subgraphs: Problem Definition and Fast Solutions *(KDD '06).* ACM, New York, NY, USA, 404–413.
[31] William Yang Wang and William W. Cohen. 2016. Learning First-Order Logic Embeddings via Matrix Factorization *(IJCAI'16).* AAAI Press, 2132–2138.
[32] Z. Wang, K. Zhao, H. Wang, X. Meng, and J. Wen. 2015. Query Understanding Through Knowledge-based Conceptualization *(IJCAI'15).* AAAI Press, 3264–3270.
[33] wikipedia. [n. d.]. Knowledge graph. https://en.wikipedia.org/wiki/Knowledge_graph
[34] W Xiong, T Hoang, and W Wang. 2017. DeepPath: A Reinforcement Learning Method for Knowledge Graph Reasoning. In *EMNLP.*
[35] Shengqi Yang, Yinghui Wu, Huan Sun, and Xifeng Yan. 2014. Schemaless and Structureless Graph Querying. *Proc. VLDB Endow.* 7, 7 (March 2014), 565–576.
[36] F. Zhang, J. Yuan, D. Lian, X. Xie, and W. Ma. 2016. Collaborative Knowledge Base Embedding for Recommender Systems *(KDD '16).* ACM, 353–362.
[37] T. Zhang, G. Shi, L. Huang, and D. Lu and. 2018. GAIA - A Multi-media Multi-lingual Knowledge Extraction and Hypothesis Generation System *(TAC' 18).*
[38] Q. Zhou, L. Li, N. Cao, L. Ying, and H. Tong. 2019. adversarial attacks on multi-network mining: problem definition and fast solutions *(ICDM '19).*

APPENDIX: REPRODUCIBILITY

Reproducibility. Two datasets are used in our experiments, including Yago core version and Covid-19 knowledge graph. All experiments are performed on a machine with an Intel Core-i7 3.00GHz CPU and 64GB memory. The details of datasets, machine and parameters can be found in Section 5. All datasets are publicly available. The source code will be released upon publication of the paper. The video demonstration of the system can be found at https://github.com/lihuiliullh/KompaRe.

A – Predicate Entropy and Similarity Examples

Here, we use Figure 1 to give two examples on how to calculate the predicate entropy and predicate-predicate similarity. For the predicate entropy, suppose we want to calculate the entropy for command. Three entities (President, US Army and Air Force) have command as their out links, and the numbers of out links of command are 2, 3 and 2, respectively. Therefore, $V_i^2 = \{\texttt{President}, \texttt{AirForce}\}$, $V_i^3 = \{\texttt{USArmy}\}$ and $V_i^d = \emptyset$ otherwise. Therefore, we have $E(\texttt{command}) = -\frac{2}{3}\log(\frac{2}{3}) - \frac{1}{3}\log(\frac{1}{3}) = 0.92$ and $w(\texttt{command}) = 0.43$.

For the predicate-predicate similarity, suppose we want to calculate the similarity between major and study. Both major and study have only one adjacent neighboring predicate graduate. This means that for any predicate $i \neq \texttt{graduate}$, $U(\texttt{major}, i) = U(\texttt{study}, i) = 0$. Since E(graduate) = 0, we have $w(\texttt{graduate}) = 2\sigma(\infty) - 1 = 1$. We have TF(major, graduate) = TF(study, graduate) = log(1+1×1) = 1, and $U(\texttt{major}, \texttt{graduate}) = U(\texttt{study}, \texttt{graduate}) = IDF(\texttt{graduate}) = \log\frac{8}{4} = 1$. If we compare the two vectors, $U_{\texttt{major}}$ and $U_{\texttt{study}}$, we find that they are the same. Therefore, we have that Sim(major, study) = 1.

B – Proof of Lemma 2

Proof. We rewrite the loss function as

$$Loss = ||H_1 - H_2||_F^2 = \sum_{i,j}(h_{e_{i,j}} - h_{c_{i,j}})^2$$

Take the derivative, together with Lemma 1, we have

$$\begin{aligned} I(A_n(i,j)) &= \sum_{k\in N(n)} -2(h_{e_{k,n}} - h_{c_{k,n}})\frac{\partial sim(KS_n, KS_k)}{\partial A_n(i,j)} \\ (N_n(i)) &= \sum_{k\in N(n)} -2(h_{e_{k,n}} - h_{c_{k,n}})\frac{\partial sim(KS_n, KS_k)}{\partial N_n(i)} \\ I(N_n^l(i,i)) &= \sum_{k\in N(n)} -2(h_{e_{k,n}} - h_{c_{k,n}})\frac{\partial sim(KS_n, KS_k)}{\partial N_n^l(i,i)} \end{aligned} \quad (3)$$

which completes the proof. □

C – Predicate Entropy

The top-*10* predicates with the highest predicate entropy in Yago dataset are edited, isConnectedTo, actedIn, playsFor, dealsWith, directed, hasNeighbor, isAffiliatedTo, wroteMusicFor and exports. Predicates like actedIn, playFor, hasNeighbor have a very high entropy. The reason is that these predicates not only occur commonly in the Yago knowledge graph, but also have a high degree of uncertainty. It is consistent with our hypothesis that these predicates provide little semantic information about the entities around them. On the contrary, The top-*10* predicates with the lowest predicate entropy in Yago dataset are diedIn, hasGender, hasCurrency, wasBornIn, hasAcademicAdvisor, isPoliticianOf, isMarriedTo, hasCaptal, hasWebsite, and isCitizenOf. Predicates like diedIn, wasBornIn, isMarriedTo, isPoliticianOf have a very low entropy. They provide specific and detailed background information about the entities around them.

D – Predicate Similarity Results

Table 6 shows the predicate similarity between isTypeOf and other predicates.

Table 6: Predicate similarity of isTypeOf with others

predicate	sim	predicate	sim	predicate	sim	predicate	sim
isCitizenOf	0.840	isLeaderOf	0.955	isAffiliatedTo	0.808	isPoliticianOf	0.917
livesIn	0.972	owns	0.945	exports	0.706	dealsWith	0.697
hasCapital	0.786	command	0.216	happenedIn	0.767	participatedIn	0.869
worksAt	0.752	isLocatedIn	0.870				

E – Pair-wise Comparative Reasoning Examples

Here, we give an example of the pair-wise comparative reasoning. Consider a fake news which says *"The white house will participate in the operation mountain thrust, because the white house wants to punish the iraqi army."* From this news, we can extract two query clues, including <White House, participatedIn, Operation Mountain Thrust> and <White House, punish, Iraqi Army>. Figure 7 shows these two corresponding knowledge segments. Table 7 and Table 8 show the node attribute influence value, node influence value and edge influence value of KS_1 and KS_2, respectively. We can see from Table 7 that for KS_1, the top-*50%* elements with the highest node attribute influence are Washington,D.C, United States President, White House and United States. For KS_2, the top-*50%* elements with the highest node attribute influence are White House, Washington,D.C, and United States. Because all the import elements with the highest node attribute influence of KS_2 also belong to KS_1, the key elements overlapping rate for node attribute is 100%. For the top-*50%* elements with the highest node influence, we obtain the same result. As for the top-*50%* edges of KS_1 with the highest influence, there is one edge (<United States, hasCapital, Washington,D.C>) which also belongs to the top-50% edges of KS_2. Therefore, the key elements overlapping rate of KS_1 and KS_2 is $\frac{1+1+\frac{1}{3}}{3} = \frac{7}{9} > 60\%$. This means that these two clues refer to the same thing.

We further check if there is any inconsistency between them. To this end, we extract the knowledge segment for <Operation Mountain Thrust, isTypeOf, Iraqi Army> and <Iraqi Army, isTypeOf, Operation Mountain Thrust>. The right part of Figure 7 shows the knowledge segment for <Operation Mountain Thrust, isTypeOf, Iraqi Army>. The proposed TF-IDF predicate-predicate similarities between isTypeOf and other predicates are shown in Table 6. Based on that, we have infTrans(Operation Mountain Thrust, Iraqi Army) = infTrans(Iraqi Army, Operation Mountain Thrust) = 0.505 < 0.700. This means that "Operation Mountain Thrust" and "Iraqi Army" are two different things. Therefore, we conclude that the two given clues are inconsistent.

F – Collective Comparative Reasoning Examples

Here, we evaluate the effectiveness of the proposed collective comparative reasoning. We test a query graph with three edges, including <White House, punish, Iraqi Army>, <Washington,D.C, means, White House> and <Washington,D.C, participatedIn, Operation Mountain Thrust>. Figure 8 shows the query graph and the corresponding semantic matching subgraph. As we can see,

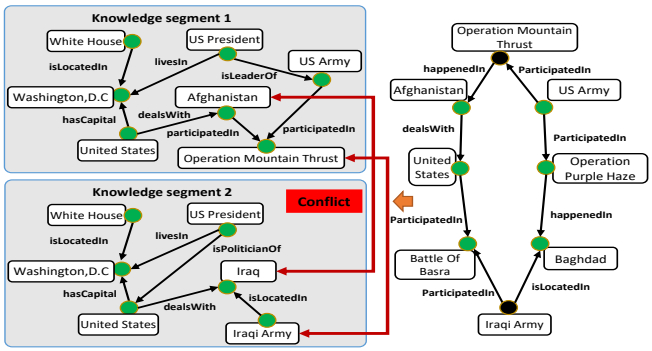


Figure 7: Pair-wise comparative reasoning results.

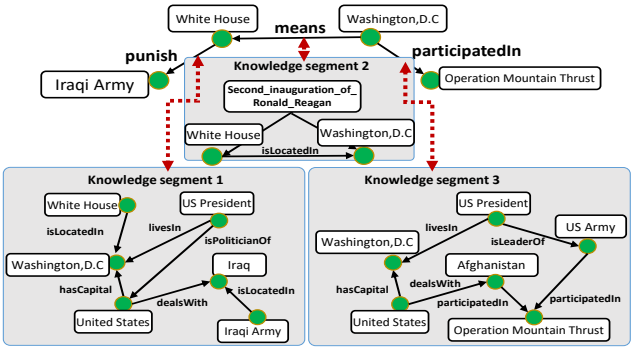


Figure 8: Collective comparative reasoning results.

Table 7: Pairwise node attribute influence ranking

(a) Node attribute influence: KS_1

Rank	Predicate	value
1	Washington, D.C	$4.49\ e^{-5}$
2	United States President	$3.92\ e^{-5}$
3	White House	$3.90e^{-5}$
4	United States	$3.87e^{-5}$
5	United States Army	$1.96e^{-5}$
6	Afghanistan	$1.95e^{-5}$
7	Operation Mountain Thrust	$1.95e^{-5}$

(b) Node attribute influence: KS_2

Rank	Predicate	value
1	Washington, D.C	$4.51e^{-5}$
2	White House	$3.90e^{-5}$
3	United States	$3.87e^{-5}$
4	United States President	$3.83e^{-5}$
5	Iraq	$1.97e^{-5}$
6	Iraqi Army	$1.86e^{-5}$

(c) Node influence: KS_1

Rank	Predicate	value
1	Washington, D.C	$1.54e^{-5}$
2	United States President	$6.88\ e^{-6}$
3	United States	$6.18e^{-6}$
4	White House	$5.37e^{-6}$
5	United States Army	$3.06e^{-6}$
6	Operation Mountain Thrust	$3.06\ e^{-6}$
7	Afghanistan	$3.06e^{-6}$

(d) Node influence: KS_2

Rank	Predicate	value
1	Washington, D.C	$1.47e^{-5}$
2	United States	$6.19e^{-6}$
3	White House	$5.37e^{-6}$
4	United States President	$4.66e^{-6}$
5	Iraq	$3.04e^{-6}$
6	Iraqi Army	$1.50e^{-6}$

Table 8: Pairwise edge influence ranking

Edge influence of KS_1		
Rank	Triple	value
1	<United States President, livesIn, Washington, D.C>	$4.28e^{-4}$
2	<United States, hasCapital, Washington, D.C>	$4.28\ e^{-4}$
3	<United States President, isLeaderOf, US Army>	$3.67e^{-4}$
4	<United States, dealsWith, Afghanistan>	$3.67\ e^{-4}$
5	<White House, isLocatedIn, Washington, D.C>	$3.59\ e^{-4}$
6	<Afghanistan, participatedIn, Operation Mountain Thrust>	$3.59\ e^{-4}$
7	<US Army, participatedIn, Operation Mountain Thrust>	$3.59e^{-4}$
Edge influence of KS_2		
Rank	Triple	value
1	<United States, hasCapital, Washington, D.C>	$4.51\ e^{-4}$
2	<United States, dealsWith, Iraq>	$3.89\ e^{-4}$
3	<White House, isLocatedIn, Washington, D.C>	$3.83\ e^{-4}$
4	<United States President, livesIn, Washington, D.C>	$3.83\ e^{-4}$
5	<United States President, politician, United States>	$3.31e^{-4}$
6	<Iraqi Army, isLocatedIn, Iraq>	$3.20e^{-4}$

if we use the pair-wise comparative reasoning method to check each pair of them, all of them are true. However, if we use the collective comparative reasoning method, we could detect the inconsistency in the query graph as follows.

Table 9 and Table 10 show the node attribute influence, the node influence, and the edge influence of these three knowledge segments, respectively. If we check each pair of clues in the query graph, we find that the key elements overlapping rate between KS_1 and KS_3 is more than 60%. This is because the overlapping rates are 66.6% for node attribute influence, 100% for node influence and 66.6% for edge influence, which give the average overlapping rate $\frac{\frac{2}{3}+1+\frac{2}{3}}{3} = \frac{7}{9} > 60\%$.

Based on this, we future check <Washington,D.C, isTypeOf, White House> or <White House, isTypeOf, Washington,D.C>.

Table 9: Collective node attribute influence ranking

(a) Node attribute influence

Node attribute influence: KS_1		
Rank	Predicate	value
1	Washington,D.C	$2.93e^{-5}$
2	White House	$2.71e^{-5}$
3	United States	$1.59e^{-5}$
4	United States President	$1.47e^{-5}$
5	Iraq	$1.47e^{-5}$
6	Iraqi Army	$1.36e^{-5}$
Node attribute influence: KS_2		
Rank	Predicate	value
1	Washington,D.C	$2.29\ e^{-4}$
2	White House	$1.31\ e^{-4}$
Node attribute influence: KS_3		
Rank	Predicate	value
1	Washington,D.C	$2.08\ e^{-4}$
2	United States President	$1.10\ e^{-4}$
3	United States	$1.10e^{-4}$
4	United States Army	$1.09e^{-4}$
5	Operation Mountain Thrust	$1.09e^{-4}$
6	Afghanistan	$1.09\ e^{-4}$

(b) Node influence

Node influence: KS_1		
Rank	Predicate	value
1	Washington,D.C	$4.57e^{-6}$
3	United States	$3.57e^{-6}$
4	United States President	$2.35e^{-6}$
5	Iraq	$2.26e^{-6}$
2	White House	$2.17e^{-6}$
6	Iraqi Army	$1.09e^{-6}$
Node influence: KS_2		
Rank	Predicate	value
1	Washington,D.C	$7.68\ e^{-6}$
2	White House	$7.68\ e^{-6}$
Node influence: KS_3		
Rank	Predicate	value
1	Washington,D.C	$1.19\ e^{-5}$
2	United States President	$1.19\ e^{-5}$
3	United States	$1.19e^{-5}$
4	United States Army	$1.18e^{-5}$
5	Operation Mountain Thrust	$1.18e^{-5}$
6	Afghanistan	$1.18\ e^{-5}$

Table 10: Collective edge influence ranking

Edge influence of KS_1		
Rank	Triple	value
1	<United States, hasCapital, Washington, D.C>	$1.20\ e^{-4}$
2	<United States President, isPoliticianOf, United States>	$1.07\ e^{-4}$
3	<United States President, livesIn, Washington, D.C>	$1.05\ e^{-4}$
4	<United States, dealsWith, Iraq>	$9.94\ e^{-5}$
5	<White House, isLocatedIn, Washington, D.C>	$8.22\ e^{-5}$
6	<Iraqi Army, isLocatedIn, Iraq>	$6.47\ e^{-5}$
Edge influence of KS_2		
Rank	Triple	value
1	<White House, isLocatedIn, Washington, D.C>	$2.23\ e^{-4}$
Edge influence of KS_3		
Rank	Triple	value
1	<United States President, livesIn, Washington, D.C>	$4.99e^{-4}$
2	<United States, hasCapital, Washington, D.C>	$4.99\ e^{-4}$
3	<United States President, isLeaderOf, US Army>	$4.99\ e^{-4}$
4	<United States, dealsWith, Afghanistan>	$4.99,\ e^{-4}$
6	<Afghanistan, participatedIn, Operation Mountain Thrust>	$4.99\ e^{-4}$
7	<US Army, participatedIn, Operation Mountain Thrust>	$4.99e^{-4}$

Our TF-IDF based predicate-predicate similarity between "isTypeOf" and "isLocatedIn" is 0.870. Thus, we have infTrans(Washington,D.C, White House) = 0.870 > 0.700. This means that these two knowledge segments have the same subject. Finally, we check <Operation Mountain Thrust, isTypeOf, Iraqi Army> or <Iraqi Army, isTypeOf, Operation Mountain Thrust>. According to the results in the previous subsection, we have that Iraqi Army and Operation Mountain Thrust are two different things. Therefore, we conclude that this query graph is inconsistent.