KompaRe: A Knowledge Graph Comparative Reasoning System

Lihui Liu, Boxin Du, Heng Ji, Hanghang Tong
Department of Computer Science, University of Illinois at Urbana Champaign
{lihuil2,boxindu2,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 predicition, 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.

KEYWORDS
knowledge graph, knowledge graph reasoning, system, comparative reasoning

ACM Reference Format:
Lihui Liu, Boxin Du, Heng Ji, Hanghang Tong. 2021. KompaRe: A Knowledge Graph Comparative Reasoning System. In Online '21: The 14th ACM International WSDM Conference, March 8-12, 2021. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION
Since its birth in 1995 [18] 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 [14]; 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 [9]; in recommendation systems, knowledge graph offers the auxiliary information to improve the recommendation quality and/or explainability [19]; in computer vision, knowledge graph can be used to pre-optimize the model to boost its performance [6]. 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 [3]. The newly discovered knowledge through reasoning provides valuable input of these downstream 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 [17], it returns the most relevant concepts for a given entity; in link prediction [10], given the ‘subject’ and the ‘object’ of a triple, it predicts the relation; in fact checking [13], given a claim (e.g., represented as a triple of the knowledge graph), it decides whether it is authentic or falsified; in subgraph matching [9], 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 asset 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 plan 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 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 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) UI.

(1) Offline Mining. There are two main offline functions supported by KOMPARE, including predicate entropy calculation and predicate-predicate similarity calculation. 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 (f1 to f3 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 (f4 and f5 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) 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 [7, 11, 16], 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, 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 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 [7] or connection subgraph [3], [11] 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 (i.e., 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 semantic matching subgraph. We would like to point out that semantic matching subgraph extraction is similar to but bears subtle difference from the traditional subgraph matching problem [12]. 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 inter-linked 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.

### 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-edge 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 \( V_i^d \) denote the node set which contains \( d \) out links with label \( i \), \( E \) denote the entropy, and \( P_i^d \) 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 \( E(i) = \sum_{d=1}^{D} -P_i^d \log(P_i^d) \), where \( P_i^d = \frac{|V_i^d|}{\sum_i |V_i^d|} \). Finally, we compute the importance of a predicate \( i \) as \( w(i) = 2\sigma(\frac{1}{E(i)}) - 1 \), where \( \sigma() \) is the sigmoid function.

### 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. [16], [11], [7]. We propose to use a TF-IDF based method \(^3\) to measure the similarity between different predicates, and transfer the knowledge graph into a weighted graph whose edge weight represents the similarity between the edge predicate and query predicate. Then, we find \( k \)-simple shortest paths [11] 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 \( e_t = <s, \text{receiveDegreeFrom}, o> \) in the knowledge graph whose predicate is \( i = \text{receiveDegreeFrom} \). In the neighborhood of \( e_t \), there is a high probability that triples like \(<s, \text{major}, o> \) and \(<s, \text{graduateFrom}, o> \) also exist (adjacent

\(^3\)The TF-IDF based method was also used in [14] for computational fact checking.

| 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 |
| 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) |

### Table 1: Summary of major functions in our system.
to \( e_i \). The predicates of these triples should have high similarity with each other. On the other hand, triples like \( <s, livesIn, o> \), \( <s, hasNei
gbor, o> \) may also occur in the adjacent neighborhood of triple \( e_i \). 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
\[
U(i, j) = \text{TF}(i, j) \times \text{IDF}(j).
\]
Finally, the similarity of two predicates is defined as
\[
\text{Sim}(i, j) = \text{Cosine}(U_i, U_j),
\]
where \( U_i \) and \( U_j \) are the \( i^{th} \) row and \( j^{th} \) row of \( U \), respectively.

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 \( E_Q \). To our best knowledge, there is no existing method for subgraph-specific knowledge segment extraction. In order to find the edge-specific knowledge segment 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 semantic matching subgraph (i.e., the subgraph-specific knowledge segment).

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). 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: 
\[
E_1^Q \leftarrow <s_1, p_1, o_1> \quad \text{and} \quad E_2^Q \leftarrow <s_2, p_2, o_2>
\]
where \( s_1, o_1, s_2, o_2 \in V_Q \) and \( p_1, p_2 \in E_Q \). We denote their corresponding knowledge segments as \( K_{S_1} \) for \( E_1^Q \) and \( K_{S_2} \) 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 \( (K_{S_1} \) and \( K_{S_2} \)), 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 \( K_{S_1} \) and \( K_{S_2} \):
\[
= (V_{K_{S_1}} \cap V_{K_{S_2}}) \cup (V_{E_{1}} \cap V_{E_{2}}), (E_{K_{S_1}} \cap E_{K_{S_2}}) \cup (E_{E_{1}} \cap E_{E_{2}}))
\]

Definition 3. Inconsistency. Given two knowledge segments \( K_{S_1} \) and \( K_{S_2} \), the inconsistency between these two knowledge segments refers to any element (node, edge attribute) in \( K_{S_1} \) and \( K_{S_2} \) that contradicts with each other.

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

1. \( s_1 = s_2, o_1 \neq o_2 \)
2. \( s_1 \neq s_2, o_1 \neq o_2 \)
3. \( s_1 = s_2, o_1 = o_2 \)
4. \( s_1 \neq s_2, o_1 = o_2 \)

In 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., if \( p_1 \) is the opposite of \( p_2 \). For example, \( p_1 = "inFather" \) and \( p_2 = "inSon" \) 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, in Figure 1, either \(<"Barack Obama", wasBornIn, Honolulu>\) or \(<"Barack Obama", inFront, Helicopter>\) could be true. But putting them together, they cannot be both true, since these two claims could not happen at the same time. In other words, they are mutually exclusive with each other and thus are inconsistent. However, queries like \(<"Barack Obama", wasBornIn, Honolulu>\) and \(<"Barack Obama", wasBornIn, Hawaii>\) are both true, because Honolulu belongs to Hawaii. Alternatively, we can say that Hawaii contains Honolulu. Another example is \(<"Barack Obama", wasBornIn, Honolulu>\) and \(<"Barack Obama", isPresident, White House>\), both of which are true. Although they have the same subject, they
refer to 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 high 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 high 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 <o1, isTypeOf, o2> and <o2, isTypeOf, o1>. 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 [20]. 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 KS1 and KS2 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 [20].

$$\text{Sim}(KS_1, KS_2) = q'_k(I - cQ_NA_k)^{-1}Q_NP_x$$

(1)

where $q'_k$ and $P_x$ are the stopping probability distribution and the initial probability distribution of random walks on the product matrix, respectively. $Q_N$ is the combined node attribute matrix of the two knowledge segments $Q_N = \sum_{j=1}^N Q_N^j \otimes Q_N^j$ where $Q_N^j (i \in \{1, 2\})$ is the diagonal matrix of the $i$th column of attribute matrix $Q_N$. $A_k$ 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 Sim(KS1, KS2) w.r.t. knowledge segment elements $\frac{\partial \text{Sim}(KS_1, KS_2)}{\partial A_k(i,j)}$, where $e$ represents an element of the knowledge segment KS1 or KS2. 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 KS1 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 KS2 can be computed in a similar way, and thus is omitted for space.

Lemma 1. (Knowledge Segment Similarity Influence Function [20].) Given Sim(KS1, KS2) in Eq. (1). Let $Q = (I - cQ_NA_k)^{-1}$ and $S^{ij}$ is a single entry matrix defined in Table 2. We have that

(1) The influence of an edge $A_1(i,j)$ in KS1 can be calculated as $$I(A_1(i,j)) = \frac{\partial \text{Sim}(KS_1, KS_2)}{\partial A_1(i,j)} = cq'_kQ_N\{S_i^{ij} \oplus S_j^{ij}\}Q_NP_x.$$

(2) The influence of a node $i$ in KS1 can be calculated as $$I(N_i) = \frac{\partial \text{Sim}(KS_1, KS_2)}{\partial N_i} = cq'_kQ_N\{\sum_{j=1}^N A_1(i,j) = S_i^{ij} \oplus S_i^{ij}\}Q_NP_x.$$
edge between node $k$ and $n$, and $h_{k,n}$ denote the weight of edge between $K_S^k$ and $K_S^n$. We have

1. The influence of an edge $A_{SN}(i,j)$ in knowledge segment $K_S^n$ can be calculated as

   $I(A_{SN}(i,j)) = \sum_{k \in N(n)} (h_{k,n} - h_{k,n}) \frac{\partial \mathbb{d}(K_S^k,K_S^n)}{\partial s_{ij}}$.

2. The influence of a node $i$ in knowledge segment $K_S^n$ can be calculated as

   $I(N_S(i)) = \sum_{k \in N(n)} (h_{k,n} - h_{k,n}) \frac{\partial \mathbb{d}(K_S^k,K_S^n)}{\partial s_{ni}}$.

3. The influence of a node attribute $j$ in knowledge segment $K_S^n$ can be calculated as

   $I(N_S^j(i,i)) = \sum_{k \in N(n)} (h_{k,n} - h_{k,n}) \frac{\partial \mathbb{d}(K_S^k,K_S^n)}{\partial s_{ni}}$.

---

**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 $<s_1, p_1, o_1>$ and $<s_2, p_2, o_2>$, respectively. We check if $<s_1, isTypeOf, o_2>$ or $<s_2, isTypeOf, o_1>$ is true. If both of them are false, we skip this pair clues because this clue pair 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., $<s_1, isTypeOf, o_2>$, $<s_2, isTypeOf, o_1>$, $<o_1, isTypeOf, o_2>$ 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?

We use the Yago dataset [15]. It contains 12,430,705 triples, 4,295,825 entities and 39 predicates. All the experiments are conducted on a moderate desktop with an Intel Core-i7 3.00GHz CPU and 64GB memory. The source code will be released upon publication of the paper.

### 5.1 KOMPARe Basics

We start with evaluating the effectiveness of the proposed predicate entropy. The top-10 predicates with the highest predicate entropy in Yago dataset are edited, isConnectedTo, actedIn, playsFor, dealsWith, directed, exports, isAffiliatedTo, wroteMusicFor and hasNeighbor. 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, hasCapital, hasWebsite, and isCitizenOf. Predicates like diedIn, wasBornIn, isMarriedTo, isPoliticianOf have

---

4It is publicly available at https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga.
5.2 Pair-wise Comparative Reasoning

Here, we evaluate the effectiveness of the proposed pair-wise comparative reasoning. We first give an example to show how it works, then we evaluate it on several subsets of Yago. 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 5 shows these two corresponding knowledge segments. Table 3 shows the node attribute influence value of $KS_1$ and $KS_2$, respectively. We can see from Table 3 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 $KS_2$.

| Table 3: Pairwise node attribute influence ranking |
|-----------------------------------------------|
| (a) Node attribute influence: $KS_1$ | (b) Node attribute influence: $KS_2$ |
| Rank | Predicate | Value | Rank | Predicate | Value |
| 1 | Washington, D.C | 4.99 e-5 | 1 | Washington, D.C | 4.51e-5 |
| 2 | United States President | 3.92 e-5 | 2 | White House | 3.90e-5 |
| 3 | White House | 3.90e-5 | 3 | United States | 3.87e-5 |
| 4 | United States | 3.87e-5 | 4 | United States President | 3.83e-5 |
| 5 | Afghanistan | 1.95e-5 | 5 | Iraq | 1.97e-5 |
| 6 | Operation Mountain Thrust | 1.95e-5 | 6 | Iraqi Army | 1.86e-5 |

We skip the node influence value and the edge influence value due to space limit.

5.3 Collective Comparative Reasoning

Here, we evaluate the effectiveness of the proposed collective comparative reasoning. We first give an example to show how it works, then we evaluate it on several subsets of Yago. We test a query graph with three edges, including <White House, punish, Iraqi Army>, <Washington, D.C, means, White House> and <Washington, D.C, White House, participatesIn, Operation Mountain Thrust>, which have high similarity with <Washington, D.C, Washington, D.C, participatesIn, Operation Mountain Thrust>. Therefore, we conclude that the two given clues are inconsistent.
participatedIn, Operation Mountain Thrust>. Figure 6 shows the query graph and the corresponding semantic matching subgraph. As we can see, if we use the pair-wise comparative reasoning method to check each pair of them, all of them are true, because none of them belong to C3 or C4. However, if we use the collective comparative reasoning method, we could detect the inconsistency in the query graph as follows.

If we check each pair of clues in the query graph, we find that the key elements overlapping rate between $K_S^1$ and $K_S^2$ 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 $\frac{1}{3} + \frac{1}{3} + \frac{1}{3} > 60%$.

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

We test collective comparative reasoning method on 3 query sets. Table 5 gives the details of the results. 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 set "Birth Place", <Barack Obama, wasBornIn, Honolulu>, <Barack Obama, means, United States Senate Barack Obama> and <United States Senate Barack Obama, wasBornIn, United States> is a positive query triad, while <Barack Obama, wasBornIn, Honolulu>, <Barack Obama, means, United States Senate Barack Obama> and <United States Senate Barack Obama, wasBornIn, Canada> is an negative query triad. The definition of the accuracy is the same as the previous section. As we can see, the average accuracy of collective comparative reasoning is more than 82%.

5.4 KNOWPARE 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 7(a) shows that its runtime scales near-linearly wrt. the number of nodes in the knowledge graph. Figure 7(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. Notice 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. If the query has been searched before, the runtime is less than 0.5 second. 6

6 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 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.

6The system was deployed in May 2020.
REFERENCES

[1] R. Andersen, F. Chung, and K. Lang. 2006. In 2006 47th Annual IEEE FOCS 06 (FOCS ’06). 475–486.
[2] B. Antoine, U. Nicolai, G. Alberto, W. Jason, and Y. Okana. [n.d.]. Translating Embeddings for Modeling Multi-relational Data. In NIPS ’13. 2797–2795.
[3] X. Chen, S. Jia, and Y. Xiang. 2020. A review: Knowledge reasoning over knowledge graph. Expert Systems with Applications 141 (2020), 112948.
[4] Limeng Cui, Suhang Wang, and Dongwon Lee. 2019. SAME: Sentiment-Aware Multi-Modal Embedding for Detecting Fake News.
[5] C. Faloutsos, K. McCurley, and A. Tomkins. 2004. Fast Discovery of Connection Subgraphs. In KDD ’04. ACM, New York, NY, USA, 118–127.
[6] 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
[7] 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
[8] C. Giovanni, S. Prashant, R. Luis, B. Johan, M. Filippo, and P. Alessandro. 2015. Computational Fact Checking from Knowledge Networks. PloS one 10 (01 2015).
[9] 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).
[10] S. Mehran Kazemi and P. David. 2018. Simple Embedding for Link Prediction in Knowledge Graphs. In Advances in Neural Information Processing Systems 31.
[11] Y. Koren, S. North, and C. Volinsky. 2006. Measuring and Extracting Proximity in Networks (KDD ’06). ACM, New York, NY, USA, 245–255.
[12] L. Liu, B. Du, and H. Tong. 2019. GFinder: Approximate Attributed Subgraph Matching (BigData ’19).
[13] H. Naeemul, A. Fatma, and C. Li. 2017. Toward Automated Fact-Checking: Detecting Check-Worthy Factual Claims by ClaimBuster (KDD ’17). 10.
[14] Prashant S, Alessandro F, Filippo M, and Giovanni C. 2017. Finding Streams in Knowledge Graphs to Support Fact Checking. 859–864.
[15] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: A Core of Semantic Knowledge (WWW ’07). Association for Computing Machinery, 10.
[16] Hanghang Tong and Christos Faloutsos. 2006. Center-piece Subgraphs: Problem Definition and Fast Solutions (KDD ’06). ACM, New York, NY, USA, 404–413.
[17] Z. Wang, K. Zhao, H. Wang, X. Meng, and J. Wen. 2015. Query Understanding Through Knowledge-based Conceptualization (IJCAI’15). AAAI Press, 3264–3270.
[18] S Yoo and O Jeong. 2020. Automating the expansion of a knowledge graph. Expert Systems with Applications 141 (2020), 112965.
[19] F. Zhang, J. Yuan, D. Lian, X. Xie, and W. Ma. 2016. Collaborative Knowledge Base Embedding for Recommender Systems (KDD ’16). ACM, 353–362.
[20] 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).