K-Join: Knowledge-Aware Similarity Join

Zeyuan Shang Yaxiao Liu Guoliang Li Jianhua Feng

Abstract—Similarity join is a fundamental operation in data cleaning and integration. Existing similarity-join methods utilize the string similarity to quantify the relevance but neglect the knowledge behind the data, which plays an important role in understanding the data. Thanks to public knowledge bases, e.g., Freebase and Yago, we have an opportunity to use the knowledge to improve similarity join. To address this problem, we study knowledge-aware similarity join, which, given a knowledge hierarchy and two collections of objects (e.g., documents), finds all knowledge-aware similar object pairs. To the best of our knowledge, this is the first study on knowledge-aware similarity join. There are two main challenges. The first is how to quantify the knowledge-aware similarity. The second is how to efficiently identify the similar pairs. To address these challenges, we first propose a new similarity metric to quantify the knowledge-aware similarity using the knowledge hierarchy. We then devise a filter-and-verification framework to efficiently identify the similar pairs. We propose effective signature-based filtering techniques to prune large numbers of dissimilar pairs and develop efficient verification algorithms to verify the candidates that are not pruned in the filter step. Experimental results on real-world datasets show that our method significantly outperforms baseline algorithms in terms of both efficiency and effectiveness.

1 Introduction

As an important operation in data cleaning and integration, similarity join has attracted significant attention from the database community. It has widespread real applications such as web clustering, duplicate detection, and collaborative filtering [3], [16], [21]. Given two collections of objects, similarity join aims to find all similar pairs from the two collections. There are many functions to quantify the similarity between objects, such as Jaccard, Cosine and edit distance [7], [24], [15], [18]. However these functions only utilize the string similarity to quantify the similarity between objects but neglect the knowledge behind the data, which plays an important role in understanding the data. For example, a startup, Factual (www.factual.com), aims to integrate the crawled points of interests (POIs) to remove duplicates. Consider two crawled POIs "`Californian food at Fillmore st`" and "`American food at Ellis dr`". Their string similarity is rather small. However they actually refer to the same POI, because `Californian food` is a sub-category of `American food`, `Ellis st` and `Fillmore dr` are very close, and the POI is at their intersection. If we have such background knowledge, we can utilize the knowledge to quantify their similarity and integrate them as the same POI. The knowledge hierarchy can also benefit other applications, such as entity resolution and clustering.

Thanks to the public knowledge bases, e.g., Freebase and Yago, we have an opportunity to utilize the knowledge to improve similarity join. For example, given Freebase, we can infer that the similarity between "`Californian food`" and "`American food`" is large, and so are "`Fillmore st`" and "`Ellis dr`." To achieve this goal, in this paper we study the problem of knowledge-aware similarity join, which, given a knowledge hierarchy and two collections of objects (e.g., POIs), finds all knowledge-aware similar object pairs. Note that our method can facilitate many real-world applications. For example, Yelp wants to classify similar restaurants together to improve restaurant recommendations, and Amazon wants to classify similar products together using the knowledge information.

To the best of our knowledge, this is the first study on knowledge-aware similarity join. There are two main challenges to address this problem. The first is how to quantify the knowledge-aware similarity. The second is how to efficiently identify the similar pairs. To address these challenges, we utilize the knowledge hierarchy to quantify the knowledge-aware similarity and propose a new similarity metric to compute knowledge-aware similarity. We then devise a filter-and-verification framework to efficiently identify the similar pairs. We devise signature-based filtering techniques to prune large numbers of dissimilar pairs and develop efficient verification algorithms to verify the candidates that are not pruned in the filter step.

To summarize, we make the following contributions.
(1) We propose a knowledge-aware similarity metric to quantify the similarity based on knowledge hierarchy and formulate the knowledge-aware similarity join problem. To the best of our knowledge, this is the first work on knowledge-aware similarity join.
(2) We propose a filter-and-verification framework to efficiently identify the similar pairs. The filter step prunes many dissimilar pairs and the verification step verifies the candidate pairs that are not pruned in the filter step.
(3) In the filter step, we generate high-quality signatures based on the knowledge hierarchy such that if two objects have no common signatures, they cannot be similar. We utilize these signatures to prune dissimilar pairs.
(4) It is rather expensive to directly compute the knowledge-aware similarity and we propose an adaptive framework to verify the candidates. We estimate the upper bounds and lower bounds of candidate pairs. We utilize upper bounds to prune dissimilar pairs and use lower bounds to avoid computing the knowledge-aware similarity.
(5) We have conducted an extensive set of experiments on

• *Zeyuan Shang, Yaxiao Liu, Guoliang Li, and Jianhua Feng are with the Department of Computer Science, Tsinghua University, Beijing, China. E-mail: zeyuanxy@gmail.com; liuyx12@mails.tsinghua.edu.cn;{liguoliang,fengjh}@tsinghua.edu.cn. Corresponding Author: Yaxiao Liu.*

real datasets. Experimental results show that our method significantly outperforms the baseline algorithms in terms of both efficiency and effectiveness.

The rest of the paper is organized as follows. We formulate the problem and review related works in Section 2. We introduce the filter-and-verification framework in Section 3. The filtering techniques are proposed in Section 4 and the adaptive verification algorithm is presented in Section 5. We make discussions in Section 6. Experimental results are reported in Section 7. We conclude in Section 8.

2 PRELIMINARIES

In this section, we first define the knowledge-aware similarity (Section 2.1) and then formulate our problem (Section 2.2). Finally, we review the related work (Section 2.3).

2.1 Knowledge-Aware Similarity

We model each object (e.g., a POI) as a set of elements (e.g., tokens) by tokenizing the object. We first discuss how to quantify the similarity between elements and then propose a knowledge-aware similarity metric for objects.

2.1.1 Knowledge-Aware Similarity For Elements

We model a knowledge hierarchy as a tree structure $\mathcal{T}$ and how to support the directed acyclic graph (DAG) structure will be discussed in Section 6.5. Given two elements e_x, e_y*, we first map them to tree nodes in $\mathcal{T}$. Here we assume that each element matches a single node and how to support the case that each element matches multiple tree nodes will be discussed in Section 6.4. If the context is clear, we also use e_x and e_y to denote the corresponding matched nodes. Let $\texttt{LCA}_{e_x,e_y}$ denote their lowest common ancestor (i.e., the common ancestor of the two nodes and any of its descendant will not be a common ancestor of the two nodes), and d_{e_x} denote the depth of node e_x (the depth of the root is 0). Intuitively, the larger $d_{e_x,e_y} = d_{\texttt{LCA}_{e_x,e_y}}$ is, the two elements are more similar. Next we define the knowledge-aware similarity.

Definition 1 (Knowledge-Aware Similarity for Elements)**.** *Given a knowledge hierarchy $\mathcal{T}$ and two elements e_x and e_y, their knowledge-aware similarity is defined as*

$$\textsc{Sim}(e_x, e_y) = \frac{d_{e_x,e_y}}{\max(d_{e_x}, d_{e_y})}. \quad (1)$$

Consider two elements `BurgerKing` and `KFC`. Their depths are 4 and their LCA is node `Fastfood` as shown in Figure 1. Their knowledge-aware similarity is $\frac{3}{4}$.

We can compute the knowledge-aware similarity between two elements e_x and e_y as follows. We first get the depths of e_x and e_y and then compute their LCA in a bottom-up manner. The time complexity is $\mathcal{O}(d_{e_x} + d_{e_y})$.

An element may map to multiple tree nodes due to (1) an element may appear in multiple nodes; (2) an element may have synonyms; and (3) an element may have typos and may map to multiple tree nodes that approximately match the element so as to tolerate typos). In this case, we enumerate every node of an element and compute the maximum similarity, i.e.,

$$\textsc{Sim}(e_x, e_y) = \max_{(e'_x,e'_y)} \frac{d_{e'_x,e'_y}}{\max(d_{e'_x}, d_{e'_y})} \varphi(e_x, e'_x)\varphi(e_y, e'_y) \quad (2)$$

where e'_x and e'_y are any mapping nodes of e_x and e_y respectively, and $\varphi(e_x, e'_x)$ is the similarity between e_x and e'_x. If $e_x = e'_x$ or they are synonyms, $\varphi(e_x, e'_x) = 1$; otherwise, we utilize normalized edit distance (edit similarity) to quantify their similarity, i.e., $\varphi(e_x, e'_x) = 1 - \frac{\texttt{ED}(e_x,e'_x)}{\max(|e_x|,|e'_x|)}$, where $\texttt{ED}(e_x, e'_x)$ is the edit distance of e_x and e'_x and $|e_x|$ is the length of e_x. For example, the edit distance of "`PizzaHut`" and "`PizzaHat`" is 1. Their edit similarity is $\frac{7}{8}$.

2.1.2 Knowledge-Aware Similarity For Objects

Given two objects S_x and S_y, to compute their similarity, we construct a bigraph $G = ((S_x, S_y), E)$, where E is the edge set. If an element in S_x is similar to an element in S_y, there is an edge between them whose weight is the knowledge-aware similarity between the two elements. To avoid involving dissimilar pairs, we remove all the edges whose weights are smaller than a given threshold δ.

To avoid mapping an element from one object to multiple elements in the other object, we use the graph matching to compute the similarity. A matching in a bigraph is a set of edges without common elements, and the *maximum weight matching* is the matching with the maximum edge weight. We use the *maximum weight matching* of G as the *fuzzy overlap* of S_x and S_y, denoted by $S_x \widetilde{\cap}_\delta S_y$. The Hungarian algorithm can be used to solve the maximum weight matching problem, with the time complexity of $\mathcal{O}(|V|^2|E|)$, where $|V|$ is the number of elements in bigraph G and $|E|$ is the number of edges in bigraph G. If we denote $|S_x|(|S_y|)$ as the number of elements in $S_x(S_y)$, the time complexity of finding the maximum weight matching is $\mathcal{O}((|S_x| + |S_y|)^2|E|)$.

Using the *fuzzy overlap*, we define knowledge-aware similarity on two objects. Here we take Jaccard as an example and how to support other metrics is discussed in Section 6.3.

Definition 2 (Knowledge-Aware Similarity for Objects)**.** *Given a knowledge hierarchy $\mathcal{T}$, two objects S_x and S_y, and an element similarity threshold δ, the knowledge-aware similarity between S_x and S_y is defined as*

$$\textsc{Sim}_\delta(S_x, S_y) = \frac{||S_x \widetilde{\cap}_\delta S_y||}{|S_x| + |S_y| - ||S_x \widetilde{\cap}_\delta S_y||}. \quad (3)$$

where $|S_x|$ is the size of S_x and $||S_x \widetilde{\cap}_\delta S_y||$ is the sum of the weights of edges in the maximum matching.

Consider two objects S_1 and S_4 in Table 1. If $\delta = 0.5$, their bigraph is shown in Figure 2. The fuzzy overlap of S_1 and S_4 is $||S_1 \widetilde{\cap}_\delta S_4|| = \frac{3}{4} + \frac{3}{5} = \frac{27}{20}$. Thus $\textsc{Sim}_\delta(S_1, S_4) = \frac{||S_1 \widetilde{\cap}_\delta S_4||}{|S_1|+|S_4|-||S_1 \widetilde{\cap}_\delta S_4||} = \frac{\frac{27}{20}}{2+3-\frac{27}{20}} = \frac{27}{73}$.

2.2 Knowledge-Aware Similarity Join

Based on the above notations (also shown in Figure 3), we define the problem of knowledge-aware similarity join.

Definition 3 (Knowledge-Aware Similarity Join)**.** *Given a knowledge hierarchy $\mathcal{T}$, two object sets $\mathcal{R}$ and $\mathcal{S}$, an element similarity threshold δ and an object similarity threshold τ, a knowledge-aware similarity join finds all similar pairs $\langle r, s\rangle \in \mathcal{R} \times \mathcal{S}$, such that $\textsc{Sim}_\delta(r, s) \geq \tau$.*

*. We use the leaf nodes of KB as a set of entities, and then for each record, we extract the entities as the elements. (For the tokens that do not match any entity, we also take them as elements.)

TABLE 1
Examples ($\delta = 0.7, \tau = 0.6$, Underline: Removed by Weighted Path Prefix).

ID	Objects	Node Signature	Node Prefix	(Deep) Path Signature	(Deep) Path Prefix
S_1	{BurgerKing, MountainView}	{Fastfood, CA}	{Fastfood}	{BurgerKing, MountainView, SanFrancisco, Fastfood}	{BurgerKing, MountainView, SanFrancisco}
S_2	{Pizza, PaloAlto, Brooklyn}	{Pizza, NY, CA}	{Pizza, NY}	{Brooklyn, PaloAlto, Pizza, SanFrancisco, NewYork}	{Brooklyn, PaloAlto, Pizza, SanFrancisco}
S_3	{Fastfood, GoogleHeadquarters}	{Fastfood, CA}	{Fastfood}	{GoogleHeadquarters, MountainView, Fastfood}	{GoogleHeadquarters, MountainView}
S_4	{PizzaHut, KFC, CA}	{Pizza, Fastfood, CA}	{Pizza, Fastfood}	{PizzaHut, CA, KFC, Pizza, Fastfood}	{PizzaHut, CA, KFC, Pizza}
S_5	{Pizza, GoogleHeadquarters}	{Pizza, CA}	{Pizza}	{GoogleHeadquarters, MountainView, Pizza}	{GoogleHeadquarters, MountainView}
S_6	{Fastfood, Manhattan}	{Fastfood, NY}	{Fastfood}	{Manhattan, Fastfood, NewYork}	{Manhattan, Fastfood}
S_7	{Brooklyn, Food}	{Food, NY}	{Food}	{Food, Brooklyn, NewYork}	{Food}
S_8	{Pizza, KFC, Dominos, SanFrancisco, Manhattan, Brooklyn}	{Pizza, Pizza, Fastfood, NY, NY, CA}	{Pizza, Pizza, Fastfood}	{Dominos, Brooklyn, KFC, CA, Manhattan, Pizza, Pizza, SanFrancisco, Fastfood, NewYork, NewYork}	{Dominos, Brooklyn, KFC, CA, Manhattan, Pizza, Pizza, SanFrancisco}
S_9	{Fastfood, PizzaHut, BurgerKing, PaloAlto, MountainView, NewYork}	{Pizza, Fastfood, Fastfood, NY, CA, CA}	{Pizza, Fastfood, Fastfood}	{PaloAlto, NY, BurgerKing, PizzaHut, MountainView, Pizza, SanFrancisco, SanFrancisco, Fastfood, Fastfood, NewYork}	{PaloAlto, NY, BurgerKing, PizzaHut, MountainView, Pizza, SanFrancisco, SanFrancisco}

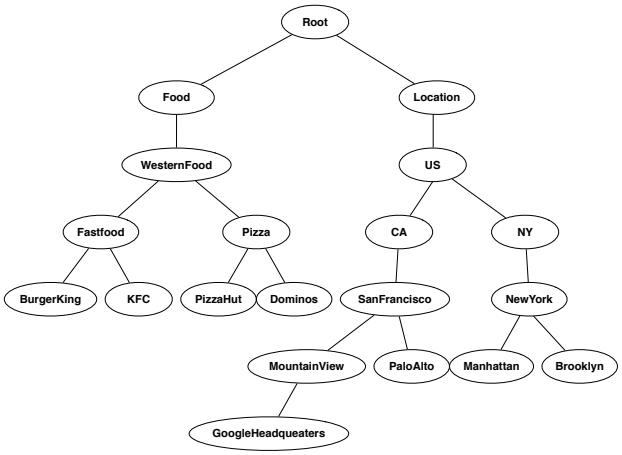


Fig. 1. A Knowledge Hierarchy.

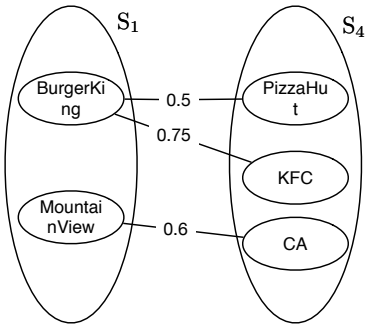


Fig. 2. Bigraph for Two Objects (δ=0.5).

Notation	Description
S	an object (a set)
e	an element
d_e	the depth of e
$\mathtt{LCA}_{e_x,e_y}$	the LCA of e_x and e_y
d_{e_x,e_y}	the depth of $\mathtt{LCA}_{e_x,e_y}$
g_e	node signature of e
G_S	node signature set of S
$\hat{G}_S$	the node prefix of S
p_e	path signature of e
P_S	path signature set of S
$\hat{P}_S$	path prefix of S
τ_{S_x}	$\lceil \tau \lvert S_x \rvert \rceil$
τ_{S_x,S_y}	$\lceil \frac{\tau}{1+\tau}(\lvert S_x \rvert + \lvert S_y \rvert) \rceil$

Fig. 3. Notations.

For example, consider the objects in Table 1. Suppose $\delta = 0.7$ and $\tau = 0.6$. S_1={BurgerKing, MountainView}, and S_3={Fastfood, GoogleHeadquarters}. SIM(BurgerKing, Fastfood)=$\frac{3}{4}$. SIM(MountainView, GoogleHeadquarters)=$\frac{5}{6}$. $||S_1 \widetilde{\cap}_\delta S_3|| = \frac{3}{4} + \frac{5}{6} = \frac{19}{12}$. $\text{SIM}_\delta(S_1, S_3) = \frac{|S_1 \widetilde{\cap}_\delta S_3|}{|S_1|+|S_3|-|S_1 \widetilde{\cap}_\delta S_3|} = \frac{\frac{19}{12}}{2+2-\frac{19}{12}} = \frac{19}{29} > \tau$. Thus $\langle S_1, S_3 \rangle$ is an answer.

Without loss of generality, we first focus on self join ($\mathcal{R} = \mathcal{S}$) and discuss how to join two sets ($\mathcal{R} \neq \mathcal{S}$) in Section 6.1.

2.3 Related Work

Similarity Join. There were many studies on string similarity joins [3], [18], [32], [31], [40], [41], [5], [8], [19], [22], [11], [9]. Given two sets of strings, similarity join finds all similar pairs. Jiang et al. provided an experimental survey in [14], [44] to compare different similarity join algorithms. The widely-adopted technique to support similarity join was prefix filtering [3]. The basic idea of prefix filtering is that, it first generates a prefix for each object and if their prefixes have no overlap, they cannot be similar. Existing studies utilized this property to prune dissimilar pairs. Xiao et al. proposed the position filter [41] and mismatch filter [40] to enhance the prefix filter. Wang et al. [31] proposed a trie-based method to directly compute similar pairs. Li et al. [18] proposed a partition-based method. Two methods were proposed to improve the quality against traditional similarity functions. Wang et al. [32] proposed to tolerate edit-distance errors in set similarity metrics, e.g., allowing approximate matching when computing set similarity. Arasu et al. [2] and Lu et al. [23] proposed transformation-based methods which used synonym rules to improve the quality. Different from them, we propose a new similarity metric based on the knowledge to improve the quality. We also devise a filter-and-verification framework to improve the efficiency.

Similarity Search. There are some similarity search algorithms [17], [33], [24], [45], [6], [42], [13], [20], which, given a set of strings and a query string, finds all similar strings to the query. Existing studies employed a filter-and-verification framework, where the filter step utilizes a lightweight filter to prune large numbers of dissimilar strings, and the verification step verifies the candidates that are not pruned in the filter step. Many effective filters have been devised to prune dissimilar strings. Sarawagi et al. [25] proposed a count filter that pruned dissimilar strings without enough common signatures to the query. Based on the count filter, Li et al. [16] developed several efficient list-merge algorithms. Li et al. [17] used variable-length grams to support string similarity search. Hadjieleftheriou et al. [12] proposed a hash-based method to estimate the number of results. Recently, Wang et al. [33] and Qin et al. [24] extended join techniques to support similarity search. Wang et al. [33] improved prefix filtering and proposed an adaptive framework. Qin et al. [24] devised an asymmetry signature to improve prefix filtering. There are some studies on top-k similarity search [43], [7], [35], [30].

Entity Resolution. Entity resolution is a critical task in data integration and cleaning [38], [36], [37]. It has been extensively studied for decades and Elmagarmid et al. [10] provide an excellent survey. Recently, there are some studies on leveraging crowdsourcing to improve the quality of

entity resolution [29], [27], [28], [34]. Entity resolution is a special case of our problem. If the knolwedge hierarchy is well designed (e.g., the nodes with the same parent refer to the same entity but not similar entities), our method can be used to improve the quality of entity resolution.

Ontological Similarity. There are several studies[1], [4], [26] on linking elements in a record to predefined entities in a knowledgebase. Different from them, we aim to find similar bases using knowledge bases.

3 The K-Join Framework

We propose a filter-and-verification framework. In the filter step, we generate signatures for each object, such that if two objects have no common signatures, they cannot be similar. We take the objects pairs with common signatures as candidates. The verification step verifies the candidates by computing the real similarity. There are two main challenges: (1) devising an effective and lightweight filter that can prune large numbers of dissimilar pairs with a little cost; (2) developing efficient verification algorithms. To address these challenges, we first propose a signature-based filtering technique (Section 3.1) and then present a verification algorithm (Section 3.2). Finally, we devise a filter-and-verification algorithm (Section 3.3).

3.1 Signature-Based Filtering

Given a knowledge hierarchy $\mathcal{T}$ and an element similarity threshold δ, for any two similar elements, we can estimate the minimum depth of their lowest common ancestor (LCA). Suppose e_x and e_y are two *different elements*, and their element similarity is $\frac{d_{e_x,e_y}}{\max(d_{e_x},d_{e_y})} \leq \frac{d_{e_x,e_y}}{d_{e_x,e_y}+1}$. If e_x and e_y are similar, we have $\frac{d_{e_x,e_y}}{d_{e_x,e_y}+1} \geq \delta$, and $d_{e_x,e_y} \geq \frac{\delta}{1-\delta}$. Thus if two different elements are similar, the depth of their LCA is at least $d_\delta = \lceil \frac{\delta}{1-\delta} \rceil$. For example, suppose $\delta = 0.7$. $d_\delta = \lceil \frac{0.7}{1-0.7} \rceil = 3$. In other words, two different elements whose LCA's depth is smaller than $d_\delta = 3$ cannot be similar, because their largest similarity is at most $\frac{2}{2+1} < 0.7$.

Based on this property, we propose a signature scheme.

Signature Scheme for Elements: For any element e with depth d_e, if $d_e < d_\delta$, we select e as its signature, i.e., $g_e = e$. If $d_e \geq d_\delta$, we select the ancestor of e whose depth is d_δ (denoted by e^{d_δ}) as its signature, i.e., $g_e = e^{d_\delta}$. As the signature refers to a tree node, we call g_e a *node signature*, which is defined as bellow.

Definition 4 (Node Signature)**.** *Given an element e, its node signature g_e is defined as*

$$g_e = \begin{cases} e & \text{If } d_e < d_\delta \\ e^{d_\delta} & \text{If } d_e \geq d_\delta \end{cases} \quad (4)$$

We can prove that, given two elements e_x and e_y, if they are similar with threshold δ, their node signatures must be the same, i.e., $g_{e_x} = g_{e_y}$, as formalized in Lemma 1.

Lemma 1. *Given two elements e_x and e_y, if their node signatures are different, they cannot be similar, i.e.,*

$$\text{If } g_{e_x} \neq g_{e_y}, \text{SIM}(e_x, e_y) < \delta.$$

Proof. Consider two similar elements e_x and e_y. If $e_x = e_y$, their node signatures must be the same. Next we consider the case that $e_x \neq e_y$. In this case, we can prove that $d_{e_x} \geq d_\delta$. Because if $d_{e_x} < d_\delta$, for any $e_x \neq e_y$, $\frac{d_{e_x,e_y}}{\max(d_{e_x} e_y)} \leq \frac{d_\delta}{d_\delta+1} < \delta$. This conflicts with that e_x and e_y are similar. Thus $d_{e_x} \geq d_\delta$. Similarly, we can prove that $d_{e_y} \geq d_\delta$. For any elements with depth exceeding d_δ, we generate the node signature on the same depth. Thus their node signatures are $e_x^{d_\delta}$ and $e_y^{d_\delta}$. If the two node signatures are different, we can also prove that e_x and e_y are not similar. Thus $e_x^{d_\delta} = e_y^{d_\delta}$. Hence the Lemma is proved. □

For example, consider elements e_1=BurgerKing, e_2=KFC, e_3=Manhattan. Suppose $\delta = 0.7$. $d_\delta = \lceil \frac{\delta}{1-\delta} \rceil = 3$. As shown in Figure 1, their node signatures are respectively Fastfood, Fastfood and NY. As $g_{e_1} = g_{e_2}$, e_1 and e_2 may be similar ($\text{SIM}(e_1, e_2) = \frac{3}{4} > 0.7$). As $g_{e_1} \neq g_{e_3}$, e_1 and e_3 cannot be similar ($\text{SIM}(e_1, e_3) = 0 < 0.6$).

Signature Scheme for Objects: Given an object S, based on the definition of SIM_δ, if another object S' is similar to S, then S' and S should have at least $\tau_S = \lceil \tau|S| \rceil$ similar elements, as $\frac{|S \widetilde{\cap}_\delta S'|}{|S|+|S'|-|S \widetilde{\cap}_\delta S'|} \geq \tau$ and $|S \widetilde{\cap}_\delta S'| \geq \lceil \tau|S| \rceil$. A filtering strategy aims to find a subset of elements for each object, called a prefix, such that if two objects are similar, their prefixes must have similar elements. An intuitive idea is to remove $\tau_S - 1$ elements and select $|S| - (\tau_S - 1)$ elements as the prefix, because if the two objects have no similar elements in the prefix, they cannot have τ_S similar elements (as the suffix only has $\tau_S - 1$ elements). Based on this idea, we propose a prefix based method.

Formally, given an object S, we generate its node signature set $G_S = \cup_{e \in S} \{g_e\}^\dagger$. Then, we fix a global order for the node signatures of all the elements, e.g., by document frequency (df) in an ascending order. Then let $\hat{G}_S = G_S[1, |S| - (\tau_S - 1)]$, which is the subset of G_S with the first $|S| - (\tau_S - 1)$ node signatures. We call $\hat{G}_S$ the node prefix of node signatures of S, which is defined as below.

Definition 5 (Node Prefix)**.** *Given an object S, its node prefix is $\hat{G}_S = G_S[1, |S| - (\tau_S - 1)]$.*

Then we can prove that if $\hat{G}_{S_x} \cap \hat{G}_{S_y} = \phi$, S_x and S_y cannot be similar as stated in Lemma 2. The basic idea is as follows. As $\hat{G}_{S_x} \cap \hat{G}_{S_y} = \phi$, without loss of generality, assume the last signature in $\hat{G}_{S_x}$ is smaller than the last signature of $\hat{G}_{S_y}$. Then all the signatures in $\hat{G}_{S_x}$ are smaller than the signatures in $G_{S_y} - \hat{G}_{S_y}$. Thus we have $\hat{G}_{S_x} \cap G_{S_y} = \phi$. As $|G_{S_x} - \hat{G}_{S_x}| < \tau_{S_x}$, $|G_{S_x} \cap G_{S_y}| = |\hat{G}_{S_x} \cap G_{S_y}| + |(G_{S_x} - \hat{G}_{S_x}) \cap G_{S_y}| < \tau_{S_x}$. As $|G_{S_x} \cap G_{S_y}| < \tau_{S_x}$, S_x and S_y have less than τ_{S_x} common node signatures. Since elements with different node signatures cannot be similar (Lemma 1), the similarity between S_x and S_y is smaller than τ.

Lemma 2. *Given two objects S_x and S_y, if their node prefixes do not overlap, they cannot be similar, i.e.,*

$$\text{If } \hat{G}_{S_x} \cap \hat{G}_{S_y} = \phi, \ \text{SIM}_\delta(S_x, S_y) < \tau.$$

Proof. As $\hat{G}_{S_x} \cap \hat{G}_{S_y} = \phi$, without loss of generality, assume the last signature in $\hat{G}_{S_x}$ is smaller than the last signature of $\hat{G}_{S_y}$. Then all the signatures in $\hat{G}_{S_x}$ are smaller than the signatures in $G_{S_y} - \hat{G}_{S_y}$. Thus we have $\hat{G}_{S_x} \cap G_{S_y} = \phi$. As $|G_{S_x} - \hat{G}_{S_x}| < \tau_{S_x}$, $|G_{S_x} \cap G_{S_y}| = |\hat{G}_{S_x} \cap G_{S_y}| + |(G_{S_x} - \hat{G}_{S_x}) \cap G_{S_y}| < \tau_{S_x}$. As $|G_{S_x} \cap G_{S_y}| < \tau_{S_x}$, S_x and S_y

†. Note that here we use a multi-set for G_S. That is if $g_{e_x} = g_{e_y}$ for $e_x \in S$ and $e_y \in S$, we keep both of them in the set.

have less than τ_{S_x} common node signatures. As elements with different node signatures cannot be similar, S_x and S_y have less than τ_{S_x} similar elements.The similarity between S_x and S_y must be smaller than τ.

Similarly, if the last signature in $\hat{G}_{S_x}$ is larger than the last signature of $\hat{G}_{S_y}$.Then all the signatures in $\hat{G}_{S_y}$ are smaller than the signatures in $G_{S_x} - \hat{G}_{S_x}$. Thus we have $\hat{G}_{S_y} \cap G_{S_x} = \phi$. As $|G_{S_y} - \hat{G}_{S_y}| < \tau_{S_y}$, $|G_{S_y} \cap G_{S_x}| = |\hat{G}_{S_y} \cap G_{S_x}| + |(G_{S_y} - \hat{G}_{S_y}) \cap G_{S_x}| < \tau_{S_y}$. As $|G_{S_y} \cap G_{S_x}| < \tau_{S_y}$, S_x and S_y have less than τ_{S_y} common node signatures. As elements with different node signatures cannot be similar, S_x and S_y have less than τ_{S_y} similar elements. Thus the similarity between S_x and S_y must be smaller than τ. □

Filtering Strategy. Based on the node prefix, we propose a filtering strategy. We first sort the node signatures for all the elements and fix a global order. Then for each object S, we take its first $|S| - (\tau_S - 1)$ node signatures as its node prefix $\hat{G}_S$. For each signature in $\hat{G}_S$, the objects that also have this signature are candidates of S. To facilitate finding the candidates, for each node signature, we use an inverted list to keep the objects that have this signature in their prefixes and the details will be discussed in Section 3.3.

For example, consider two objects S_1 and S_2 in Table 1. Suppose $\delta = 0.7$ and $\tau = 0.6$. The node signature is at level $\lceil \frac{0.7}{1-0.7} \rceil = 3$. $G_{S_1} = \{$`Fastfood`,`CA`$\}$. $G_{S_2} = \{$`Pizza`,`NY`,`CA`$\}$. $\hat{G}_{S_1} = \{$`Fastfood`$\}$. $\hat{G}_{S_2} = \{$`Pizza`, `NY`$\}$. As $\hat{G}_{S_1} \cap \hat{G}_{S_2} = \phi$, S_1 and S_2 cannot be similar. After the filtering, the number of candidate pairs is 22. As there are 36 pairs, this method can prune 40% pairs.

3.2 Verification

Given a candidate pair S_x and S_y, we check whether they are actually similar. A naive method is to directly compute their similarity, and if the similarity exceeds the threshold τ, the candidate pair is an answer. However it is rather expensive to compute the similarity. To address this issue, we propose an effective method to prune dissimilar pairs.

Based on the similarity definition, if $\frac{|S_x \widetilde{\cap}_\delta S_y|}{|S_x|+|S_y|-|S_x \widetilde{\cap}_\delta S_y|} \geq \tau$, $|S_x \widetilde{\cap}_\delta S_y| \geq \frac{\tau}{1+\tau}(|S_x| + |S_y|)$. We can estimate an upper bound of $|S_x \widetilde{\cap}_\delta S_y|$ and if the upper bound is smaller than $\tau_{S_x,S_y} = \lceil \frac{\tau}{1+\tau}(|S_x| + |S_y|) \rceil$, we prune the pair. To this end, we propose two methods to estimate an upper bound.

We first compute the node signature of each element. Then, based on Lemma 1, two elements with different signatures cannot be similar, and thus we can partition the elements of S_x/S_y into different groups based on the signatures of the elements, such that the elements in the same group may be similar while the elements in different groups cannot be similar. Formally, given two objects S_x and S_y, we first generate the node signatures of each element in S_x and S_y. For each node signature, we generate a group, and the elements having this signature will be in this group. Suppose there are m groups. We generate m groups for S_x (S_y): $S_x^1, S_x^2, \cdots, S_x^m$ ($S_y^1, S_y^2, \cdots, S_y^m$). Since the elements in S_x^i cannot be similar to the elements in S_y^j for $i \neq j$, we only need to consider the elements in S_x^i and S_y^i. Obviously, as there are $|S_x^i|(|S_y^i|)$ elements in $S_x^i(S_y^i)$, the similarity of similar elements in this group is at most $\min(|S_x^i|, |S_y^i|)$. In other words, $\min(|S_x^i|, |S_y^i|)$ is an upper bound of $|S_x^i \widetilde{\cap}_\delta S_y^i|$. Based on these groups, we get an upper bound of $|S_x \widetilde{\cap}_\delta S_y|$, i.e., $\sum_{i=1}^m \min(|S_x^i|, |S_y^i|) \geq |S_x \widetilde{\cap}_\delta S_y|$, proved as below.

Lemma 3. *Given two similar objects S_x and S_y, and S_x^i and S_y^i $(1 \leq i \leq m)$ are groups generated from S_x and S_y respectively based on the node signatures, we have*

$$\sum_{i=1}^{m} \min(|S_x^i|, |S_y^i|) \geq |S_x \widetilde{\cap}_\delta S_y| \geq \tau_{S_x,S_y} = \lceil \frac{\tau}{1+\tau}(|S_x|+|S_y|) \rceil.$$

Proof. As S_x and S_y are similar, we have $\frac{|S_x \widetilde{\cap}_\delta S_y|}{|S_x|+|S_y|-|S_x \widetilde{\cap}_\delta S_y|} \geq \tau$, then $|S_x \widetilde{\cap}_\delta S_y| \geq \frac{\tau}{1+\tau}(|S_x| + |S_y|)$. Suppose there are m groups. We generate m groups for S_x (S_y): $S_x^1, S_x^2, \cdots, S_x^m$ ($S_y^1, S_y^2, \cdots, S_y^m$). Since the elements in S_x^i cannot be similar to the elements in S_y^j for $i \neq j$, we only need to consider the elements in S_x^i and S_y^i. Obviously, as there are $|S_x^i|(|S_y^i|)$ elements in $S_x^i(S_y^i)$, the maximum matching of elements in these two groups is at most $\min(|S_x^i|, |S_y^i|)$. In other words, $\min(|S_x^i|, |S_y^i|)$ is an upper bounds of elements in S_x^i and S_y^i, i.e., $\min(|S_x^i|, |S_y^i|) \geq |S_x^i \widetilde{\cap}_\delta S_y^i|$. Thus we can get an upper bound of S_x and S_y, i.e., $\sum_{i=1}^m \min(|S_x^i|, |S_y^i|)$. Thus $\sum_{i=1}^m \min(|S_x^i|, |S_y^i|) \geq |S_x \widetilde{\cap}_\delta S_y|$. □

Next, we propose a count-based pruning method.

Count Pruning. If $\sum_{i=1}^m \min(|S_x^i|, |S_y^i|)$ is smaller than $\tau_{S_x,S_y} = \lceil \frac{\tau}{1+\tau}(|S_x| + |S_y|) \rceil$, we can prune $\langle S_x, S_y \rangle$.

For example, consider objects S_1 and S_6. Suppose $\delta = 0.7$ and $\tau = 0.6$. G_{S_1} =$\{$`Fastfood`, `CA`$\}$ and G_{S_6} =$\{$`Fastfood`, `NY`$\}$. We partition them to three groups. $G_{S_1}^1 = \{$`Fastfood`$\}$, $G_{S_1}^2 = \{$`CA`$\}$, $G_{S_1}^3 = \phi$, $G_{S_6}^1 = \{$`Fastfood`$\}$, $G_{S_6}^2 = \phi$, $G_{S_6}^3 = \{$`NY`$\}$. Thus $\sum_{i=1}^3 \min(|S_1^i|, |S_6^i|) = 1 < \frac{\tau}{1+\tau}(|S_1| + |S_6|) = \frac{3}{2}$, and we can prune this pair. Using the count pruning, 16 pairs will be pruned among the 22 candidates, and we only need to compute the real similarity of 6 pairs.

If two elements share a common signature, this method estimates their similarity as 1. However, the similarity between two elements that share the same signature may be smaller than 1. Next we discuss how to compute the maximum similarity of two elements. Given a group $S_x^i(S_y^i)$, we partition them into two parts: (1) the exactly matching elements $S_x^i \cap S_y^i$; (2) the approximate matching elements $S_x^i - S_x^i \cap S_y^i$ ($S_y^i - S_x^i \cap S_y^i$). The similarity of elements in $S_x^i \cap S_y^i$ is exactly 1. The maximum similarity of e in $S_x^i - S_x^i \cap S_y^i$ to any element in $S_y^i - S_x^i \cap S_y^i$ is $\frac{d_e}{d_e+1}$. Thus we estimate a tighter upper bound $\sum_{i=1}^m |S_x^i \cap S_y^i| + \min(\sum_{e \in S_x^i - S_x^i \cap S_y^i} \frac{d_e}{d_e+1}, \sum_{e \in S_y^i - S_x^i \cap S_y^i} \frac{d_e}{d_e+1})$. If the bound is smaller than τ_{S_x,S_y}, we prune this pair (Lemma 4).

Lemma 4. *Given two similar objects S_x and S_y, and S_x^i and S_y^i $(1 \leq i \leq m)$ are generated groups from S_x and S_y respectively based on the node signatures, we have*

$$\sum_{i=1}^{m} (|S_x^i \cap S_y^i| + \min(\sum_{e \in S_x^i - S_x^i \cap S_y^i} \frac{d_e}{d_e+1}, \sum_{e \in S_y^i - S_x^i \cap S_y^i} \frac{d_e}{d_e+1}))$$

$$\geq |S_x \widetilde{\cap}_\delta S_y| \geq \tau_{S_x,S_y} = \lceil \frac{\tau}{1+\tau}(|S_x| + |S_y|) \rceil.$$

Proof. Given a group $S_x^i(S_y^i)$, we partition them into two parts: (1) the exactly matching elements $S_x^i \cap S_y^i$; (2) the approximate matching elements $S_x^i - S_x^i \cap S_y^i$ ($S_y^i - S_x^i \cap S_y^i$). The

Algorithm 1: K-Join-FRAMEWORK($\mathcal{S}$, δ, τ)

Input: $\mathcal{S}$: The set of objects
δ: The element similarity threshold
τ: The object similarity threshold
Output: $\mathcal{A}$: The set of similar object pairs in $\mathcal{S}$

1 Get a global order of node signatures;
2 **for** *each object* S_x *in* $\mathcal{S}$ **do**
3 $\quad \hat{G}_{S_x}$ = first $|S_x| - (\tau_{S_x} - 1)$ signatures of S_x;
4 $\quad$ **for** *each signature* $g_e \in \hat{G}_{S_x}$ **do**
5 $\quad\quad$ **for** *each object* $S_y \in \mathcal{L}(g_e)$ **do**
6 $\quad\quad\quad$ **if** VERIFY(S_x, S_y)=*true* **then** $\mathcal{A} = \mathcal{A} \cup (S_x, S_y)$;
7 $\quad\quad$ $\mathcal{L}(g_e) \leftarrow S_x$;

Function K-Join-VERIFY (S_x, S_y)

Input: S_x, S_y: Two Objects
Output: True or False

1 **if** (S_x, S_y) *is not verified, i.e.,* $(S_x, S_y) \notin \mathcal{H}$ **then**
2 $\quad$ Split S_x/S_y into m subsets by node signatures ;
3 $\quad$ ub=$\sum_{i=1}^{m} |S_x^i \cap S_y^i| +$ $\min(\sum_{e \in S_x^i - S_x^i \cap S_y^i} \frac{d_e}{d_e+1}, \sum_{e \in S_y^i - S_x^i \cap S_y^i} \frac{d_e}{d_e+1})$;
4 $\quad$ **if** $ub < \tau_{S_x,S_y} = \frac{\tau}{1+\tau}(|S_x| + |S_y|)$ **then** $\mathcal{H}[(S_x, S_y)] = false$;
5 $\quad$ **else**
6 $\quad\quad$ **if** $\text{SIM}_\delta(S_x, S_y) \geq \tau$ **then** $\mathcal{H}[(S_x, S_y)]$ = true ; **else** $\mathcal{H}[(S_x, S_y)]$ = false;
7 return $\mathcal{H}[(S_x, S_y)]$;

Fig. 4. The K-Join Framework

similarity of elements in $S_x^i \cap S_y^i$ is exactly 1. The maximum similarity of e in $S_x^i - S_x^i \cap S_y^i$ to any element in $S_y^i - S_x^i \cap S_y^i$ is $\frac{d_e}{d_e+1}$. Thus we estimate a tighter upper bound $\sum_{i=1}^{m} |S_x^i \cap S_y^i| + \min(\sum_{e \in S_x^i - S_x^i \cap S_y^i} \frac{d_e}{d_e+1}, \sum_{e \in S_y^i - S_x^i \cap S_y^i} \frac{d_e}{d_e+1})$. According to the proof of Lemma 3, the lemma is proved. □

Weighted Count Pruning. We improve the count pruning technique by considering the weight of each signature. If $\sum_{i=1}^{m} |S_x^i \cap S_y^i| + \min(\sum_{e \in S_x^i - S_x^i \cap S_y^i} \frac{d_e}{d_e+1}, \sum_{e \in S_y^i - S_x^i \cap S_y^i} \frac{d_e}{d_e+1})$ is smaller than $\tau_{S_x,S_y} = \lceil \frac{\tau}{1+\tau}(|S_x| + |S_y|) \rceil$, we prune $\langle S_x, S_y \rangle$.

For example, consider two objects S_1 and S_4. Assume $\delta = 0.7$ and $\tau = 0.6$. G_{S_1} ={Fastfood, CA} and G_{S_4} ={Pizza, Fastfood, CA}. We partition them to three groups. $G_{S_1}^1 = \{\text{Fastfood}\}$, $G_{S_1}^2 = \phi$, $G_{S_1}^3 = \{\text{CA}\}$, $G_{S_4}^1 = \{\text{Fastfood}\}$, $G_{S_4}^2 = \{\text{Pizza}\}$, $G_{S_4}^3 = \{\text{CA}\}$. The count filtering cannot prune it as $\sum_{i=1}^{3} \min(|S_1^i|, |S_4^i|) = 2 \geq \frac{\tau}{1+\tau}(|S_1| + |S_{64}|) = \frac{3}{2}$. However, based on the similarity pruning, $\sum_{i=1}^{3} (|S_1^i \cap S_4^i| + \min(\sum_{e \in S_1^i - S_1^i \cap S_4^i} \frac{d_e}{d_e+1}, \sum_{e \in S_4^i - S_1^i \cap S_4^i} \frac{d_e}{d_e+1})) = \frac{3}{4} + \frac{4}{5} = \frac{31}{20} < \frac{\tau}{1+\tau}(|S_1| + |S_4|) = \frac{15}{8}$, thus we can prune this pair. After this process, the number of candidate pairs is 2.

3.3 The K-Join Algorithm

The pseudo code of the K-Join algorithm is illustrated in Figure 4. It first generates the node signatures of each element of each object and fixes a global order of all the node signatures (line 1). Then for each object S_x, it generates its node prefix $\hat{G}_{S_x}$ with the first $|S_x| - (\tau_{S_x} - 1)$ signatures (line 3). For each signature g_e in $\hat{G}_{S_x}$, the objects that share signature g_e are candidates of S_x. To efficiently get such objects, we utilize an inverted list $\mathcal{L}(g_e)$ to keep them. Thus each object S_y on $\mathcal{L}(g_e)$ is a candidate of S_x. Next we verify (S_x, S_y). If the candidate is actually similar, we add it as a result (line 6). Lastly, we append e on $\mathcal{L}(g_e)$ (line 7).

The K-Join-VERIFY function verifies whether two objects S_x and S_y are similar. If (S_x, S_y) has not been verified, we generate subsets of S_x and S_y based on their node signatures (line 2). Then we estimate an upper bound (line 3). If the upper bound is smaller than the threshold, the pair is not similar (line 4). Otherwise, we compute the real similarity. If the similarity exceeds the threshold, the pair is similar (line 6); the pair is dissimilar otherwise (line 6). Finally, we return the verification result (line 7).

4 THE DEPTH-AWARE FILTERING

The filtering method in the framework generates coarse-grained signatures for small thresholds. For example, if $\delta = 0.6$, the generated node signature is at level $\lceil \frac{\delta}{1-\delta} \rceil = 2$. Obviously for coarse-grained signatures, many dissimilar elements may generate the same signature. Consider two elements e_1= BurgerKing and e_4=Dominos. Suppose $\delta = 0.6$. Their node signatures are both "WesternFood", but their similarity is $\text{SIM}(e_1, e_4) = \frac{2}{4} = 0.5 < 0.6$. Thus, for an element e at level d_e, if d_e is far larger than 2, the node signature is too coarse for the element. To address this issue, we propose a depth-aware filtering method, which utilizes the depth to generate the signature. This method generates fine-grained signatures and can significantly reduce the number of candidates. We first discuss how to generate depth-aware signatures for elements (Section 4.1). Then we discuss how to extend it to support objects (Section 4.2).

4.1 Path Signature for Elements

For each element, we generate a depth-aware signature based on its depth. Consider an element e_y with depth d_{e_y}. If another node e_x with the same depth $d_{e_x} = d_{e_y}$ is similar to e_y, we have $d_{e_x,e_y} \geq \delta d_{e_x}$, as $\text{SIM}(e_x, e_y) = \frac{d_{e_y,e_y}}{\max(d_{e_x}, d_{e_y})} \geq \delta$. Thus we can select $e_y^{\lceil \delta d_{e_x} \rceil}$(the ancestor of e_y at level $\lceil \delta d_{e_x} \rceil$) as the signature of e_y. It is easy to prove that for any two elements with the same depth, if they are similar, they must share a common signature.

However, it is not enough to select a single signature for each element. For example, consider another element e_x with depth $d_{e_x} < d_{e_y}$. e_x selects $e_x^{\lceil \delta d_{e_x} \rceil}$ as its signature and e_y selects $e_y^{\lceil \delta d_{e_y} \rceil}$ as its signature. If $\lceil \delta d_{e_x} \rceil \neq \lceil \delta d_{e_y} \rceil$, the two signatures are not the same. Thus even if e_x and e_y are similar, they do not share a common signature. To address this issue, we can also take $e_y^{\lceil \delta d_{e_x} \rceil}$ as e_y's signature. Thus for any $d_{e_x} < d_{e_y}$, we generate a signature $e_y^{\lceil \delta d_{e_x} \rceil}$ for S_y. We find that if d_{e_x} is too small, e_x cannot be similar to e_y. Thus we want to compute the minimum d_{e_x} of elements that can be similar to e_y. Note that if $d_{e_x} < \delta d_{e_y}$, $\text{SIM}(e_x, e_y) = \frac{d_{e_x,e_y}}{\max(d_{e_x}, d_{e_y})} \leq \frac{d_{e_x}}{d_{e_y}} < \delta$, and e_x cannot be similar to e_y. Thus the minimum depth is δd_{e_y}, and we only consider the elements with depth d_{e_x} s.t. $\lceil \delta d_{e_y} \rceil \leq d_{e_x} \leq d_{e_y}$. Thus for e_y, its signatures include $e_y^{\lceil \delta \lceil \delta d_{e_y} \rceil \rceil}$, $e_y^{\lceil \delta \lceil \delta d_{e_y} \rceil \rceil + 1}$, $\cdots$, $e_y^{\lceil \delta d_{e_y} \rceil}$. We call them *path signatures*.

The above method takes e_y as a reference element. Similarly, we can take e_x as a reference element. Suppose

$d_{e_x} < d_{e_y}$. We generate signature $e_x^{\delta d_{e_y}}$ for e_x. We also want to compute the maximum depth of d_{e_y}. As $\frac{d_{e_x}}{d_{e_y}} \geq \text{SIM}(e_x, e_y) = \frac{d_{e_x,e_y}}{\max(d_{e_x},d_{e_y})} \geq \delta$. The maximum depth is $\frac{d_{e_x}}{\delta}$. Thus for e_x, we generate the following signatures $e_x^{\lceil \delta d_{e_x} \rceil}$, $e_x^{\lceil \delta d_{e_x} \rceil+1}$, $\cdots$, $e_x^{d_{e_x}}$. Since the former signatures have small depth and the latter have large depth, we call the former *shallow path signatures* (or *shallow signatures* for short), and call the latter *deep path signatures* (or *deep signatures* for short). They are two special instances of path signatures. Next we formally define the shallow and deep signatures.

Definition 6 (Shallow Signatures)**.** *For element e with depth d_e, its shallow signatures are $p_e = \{e^{\lceil \delta \lceil \delta d_e \rceil \rceil}, e^{\lceil \delta \lceil \delta d_e \rceil \rceil+1}, \cdots, e^{\lceil \delta d_e \rceil}\}$.*

Definition 7 (Deep Signatures)**.** *For element e with depth d_e, its deep signatures are $p_e = \{e^{\lceil \delta d_e \rceil}, e^{\lceil \delta d_e \rceil+1}, \cdots, e^{d_e}\}$.*

For example, consider e_1=BurgerKing. If $\delta = 0.6$, $\lceil \delta d_{e_1} \rceil$ = 3 and $\lceil \delta \lceil \delta d_{e_1} \rceil \rceil$ = 2. The shallow signatures of e_1 are {Fastfood, WesternFood} and the deep signatures of e_1 are {Fastfood, BurgerKing}. Similarity for e_4=Dominos, its shallow signatures are {Pizza, WesternFood} and deep signatures are {Pizza, Dominos}. Obviously the node and shallow signatures cannot prune $\langle e_1, e_4 \rangle$ as they share a node signature WesternFood and a shallow signature WesternFood, but the deep signature can prune the pai.

Filtering Strategy. Based on the shallow (or deep) signatures, we propose a filtering technique. For each element, we generate its shallow (or deep) signatures. If two elements have no common shallow (or deep) signatures, they cannot be similar as proved in Lemma 5. (Note that we do not need to generate both shallow and deep signatures. Instead, we only need to generate shallow or deep signatures.)

Lemma 5. *Given two elements, if their shallow (or deep) signatures have no overlap, they cannot be similar.*

Proof. Given two elements e_x and e_y, as we use the parent elements of e_x (e_y) whose depths are between $[\lceil \delta d_{e_x} \rceil, d_{e_x}]$ as the deep signatures of e_x (e_y), if their deep signatures have no overlap, the depth of the LCA of e_x and e_y must be smaller than $\max(\lceil \delta d_{e_x} \rceil, \lceil \delta d_{e_y} \rceil)$. Thus we have $\text{SIM}(e_x, e_y) = \frac{d_{e_x,e_y}}{\max(d_{e_x},d_{e_y})} \leq \frac{\max(\lceil \delta d_{e_x} \rceil, \lceil \delta d_{e_y} \rceil)-1}{\max(d_{e_x},d_{e_y})} < \frac{\max(\delta d_{e_x}, \delta d_{e_y})}{\max(d_{e_x},d_{e_y})} \leq \delta$. So e_x and e_y are not similar. The proof for shallow signatures is the same to the proof above. □

Shallow Signatures vs Deep Signatures. We compare the shallow and deep signatures. The number of shallow signatures (i.e., $\delta(1-\delta)d_e + 1$) is smaller than that of deep signatures (i.e., $(1-\delta)d_e + 1$). However, the shallow signature is coarse-grained (with small depth) while the deep signature is fine-grained (with large depth). For fine-grained signatures, elements have low probability to be matched and there are smaller numbers of candidates, and thus the deep signatures have high pruning power. Next we use the deep signature as an example to discuss how to generate the prefix. The techniques can be used for shallow signatures.

4.2 Path Signature for Objects

We discuss how to generate the signatures of objects. Given an object S, based on the definition of SIM_δ, if another object S' is similar to S, then S' and S should have at least $\tau_S = \lceil \tau |S| \rceil$ similar elements, as $\frac{|S \widetilde{\cap}_\delta S'|}{|S|+|S'|-|S \widetilde{\cap}_\delta S'|} \geq \tau$ and $|S \widetilde{\cap}_\delta S'| \geq \lceil \tau |S| \rceil$. It is expensive to directly identify the objects that have $\tau_S = \lceil \tau |S| \rceil$ similar elements with S. Instead, we can utilize the path signatures to efficiently find such objects. To this end, for object S, we generate its path signature set $P_S = \cup_{e \in S} p_e$. Next we want to find a prefix $\hat{P}_S$ of P_S, such that if two objects are similar, their prefixes should have common signatures. An intuitive idea is to remove $\tau_S - 1$ elements and select $|S| - (\tau_S - 1)$ elements as the prefix, because if the two objects have no similar elements in the prefix, they cannot have τ_S similar elements. However this idea relies on two conditions:
(1) In the suffix, there are $\tau_S - 1$ elements. That is even if the suffix of S_x exactly matches that of object S_y, they still only have $\tau_S - 1$ similar elements. Thus if they are similar, they should have at least one similar element in the prefix.
(2) The signatures are sorted based on a global order. If the signatures are not sorted based on a global order, we cannot select a prefix (as the prefix of S_x may have similar elements in the suffix of S_y and vice versa).

If any condition is not true, this prefix based method does not work. Next we propose an efficient method to generate the prefix that satisfies these two conditions.

4.2.1 Path Prefix

Given an object S, we first generate the path signatures of each element. We then sort the path signatures based on their document frequency (df) in an ascending order. Thus the signatures are sorted based on a global order and we can guarantee the second condition.

Next we remove the signatures from the path signature set P_S in a reverse order. When we remove a signature, we count the number of elements who have signatures removed and check the number of removed elements. If the number reaches $\tau_S = \lceil \tau |S| \rceil$, we will not remove the signature any more and terminate the process, because if we remove such signature, the suffix contains signatures of τ_S elements (the suffixes of two objects can have τ_S similar elements). Then the remainder signatures are in the prefix of S. In this way, we can also guarantee the first condition.

Formally, let P_S denote the set of path signatures of S, $P_S[i, |P_S|]$ denote the suffix of P_S which is a subset of P_S from the i-th signature to the last signature, $P_S[1, i]$ denote the prefix of P_S which is a subset of P_S with the first i signatures. Let $\text{DISTELE}(P_S[i, |P_S|])$ denote the number of distinct elements in $P_S[i, |P_S|]$. Then we can define the prefix of path signatures, called path prefix.

Definition 8 (Path Prefix)**.** *Given an object S, the path prefix of S is $\hat{P}_S = P_S[1, i]$, such that*

$$\text{DISTELE}(P_S[i, |P_S|]) = \lceil \tau |S| \rceil \text{ and}$$
$$\text{DISTELE}(P_S[i+1, |P_S|]) = \lceil \tau |S| \rceil - 1$$

If the path prefixes of two objects have no overlap, the two objects cannot be similar as stated in Lemma 6.

Lemma 6. *Given two objects S_x and S_y, if their path prefixes do not overlap, they cannot be similar, i.e.,*

$$\text{If } \hat{P}_{S_x} \cap \hat{P}_{S_y} = \phi, \ \text{SIM}_\delta(S_x, S_y) < \tau.$$

Proof. As $\hat{P}_{S_x} \cap \hat{P}_{S_y} = \phi$, without loss of generality, assume the last signature in $\hat{P}_{S_x}$ is smaller than the last signature

of $\hat{P}_{S_y}$. Then all the signatures in $\hat{P}_{S_x}$ are smaller than the signatures in $P_{S_y} - \hat{P}_{S_y}$. Thus we have $\hat{P}_{S_x} \cap P_{S_y} = \phi$. As $\text{DISTELE}(P_{S_x} - \hat{P}_{S_x}) < \tau_{S_x}$, $\text{DISTELE}(P_{S_x} \cap P_{S_y}) = \text{DISTELE}(\hat{P}_{S_x} \cap P_{S_y}) + \text{DISTELE}((P_{S_x} - \hat{P}_{S_x}) \cap P_{S_y}) < \tau_{S_x}$. As $\text{DISTELE}(P_{S_x} \cap P_{S_y}) < \tau_{S_x}$, S_x and S_y have less than τ_{S_x} similar elements. Thus the similarity of S_x and S_y must be smaller than τ. Similarly, if the last signature in $\hat{P}_{S_x}$ is larger than the last signature of $\hat{P}_{S_y}$. Then all the signatures in $\hat{P}_{S_y}$ are smaller than the signatures in $P_{S_x} - \hat{P}_{S_x}$. Thus $\hat{P}_{S_y} \cap P_{S_x} = \phi$. As $\text{DISTELE}(P_{S_y} - \hat{P}_{S_y}) < \tau_{S_y}$, $\text{DISTELE}(P_{S_y} \cap P_{S_x}) = \text{DISTELE}(\hat{P}_{S_y} \cap P_{S_x}) + \text{DISTELE}((P_{S_y} - \hat{P}_{S_y}) \cap P_{S_x}) < \tau_{S_y}$. As $\text{DISTELE}(P_{S_y} \cap P_{S_x}) < \tau_{S_y}$, S_x and S_y have less than τ_{S_y} similar elements. Thus S_x and S_y cannot be similar. □

For example, consider object S_4. Assume $\delta = 0.7$ and $\tau = 0.6$. $P_{S_4} = \{\texttt{PizzaHut}, \texttt{CA}, \texttt{KFC}, \texttt{Pizza}, \texttt{Fastfood}\}$. $\tau_{S_4} = \lceil 0.6 \cdot 3 \rceil = 2$. We first remove the last signature `Fastfood` and $\text{DISTELE}(P_S[6,6]) = 1$. Next we try to remove the fourth signature `Pizza`. As `Pizza` and `Fastfood` are generated from different elements, $\text{DISTELE}(P_S[5,6]) = 2$. Thus we cannot prune the fourth signature from P_{S_4} and thus $\hat{P}_{S_4} = P_{S_4}[1,4] = \{\texttt{PizzaHut}, \texttt{CA}, \texttt{KFC}, \texttt{Pizza}\}$. Similarly for S_1, $P_{S_1} = \{\texttt{BurgerKing}, \texttt{MountainView}, \texttt{SanFrancisco}, \texttt{Fastfood}\}$. As $\tau_{S_1} = \lceil 0.6 \cdot 2 \rceil = 2$. We prune 1 signature from P_{S_1} and $\hat{P}_{S_1} = P_{S_1}[1,3] = \{\texttt{BurgerKing}, \texttt{MountainView}, \texttt{SanFrancisco}\}$. As $\hat{P}_{S_1} \cap \hat{P}_{S_4} = \phi$, we prune this pair. There are 15 candidate pairs, which is better than the node prefix (22 candidates).

Filtering Strategy. Based on the path prefix, we propose a filtering strategy. We first sort the path signatures for all the elements and fix a global order of path signatures. Then for each object S, we generate its path signature $\hat{P}_S$. Next we can utilize the filtering method in the framework to generate the candidates using path signatures. The only difference is to use path prefix to replace the node prefix.

4.2.2 Weighted Path Prefix

We can further improve the path prefix by considering the maximum similarity of two elements given a matching signature. For example, given an element e, for its path signature e^d, the maximum possible similarity between e to any other element e' is $\frac{d}{d_e} \leq \frac{d}{\max(d_e, d_{e'})}$ given matching the signature e^d. Thus in the path signature, we associate each signature with a maximum similarity. When we remove a signature, we check its weight $\frac{d}{d_e}$ and compute the sum of the weight of removed signatures. Note if two signatures are from the same element, we only keep the larger weight (as any another element cannot have larger similarity to e than this weight). As the weight is smaller than 1, this weighted strategy can prune more signatures than the path signatures. Next, we formally introduce this idea.

Definition 9 (Weighted Path Prefix)**.** *Given an object S, the path prefix of S is $\hat{P}_S = P_S[1,i]$, such that $\text{MSIM}(P_S[i, |P_S|]) \geq \tau|S|$ and $\text{MSIM}(P_S[i+1, |P_S|]) < \tau|S|$* where $\text{MSIM}(P_S[i, |P_S|])$ is the sum of maximum similarity of signatures in set $P_S[i, |P_S|]$.

If the weighted path prefixes of two objects have no overlap, the two objects cannot be similar (Lemma 7).

Algorithm 2: PATHPREFIXFILTER($\mathcal{S}$, δ, τ)

Input: $\mathcal{S}$: The set of objects S
δ: The element similarity threshold
τ: The object similarity threshold
Output: $\mathcal{A}$: The set of similar object pairs in $\mathcal{S}$

1 Get a global order of path signatures;
2 **for** *each object S_x in $\mathcal{S}$* **do**
3 P_{S_x} = path signature set of S_x;
4 $\hat{P}_{S_x}$ = (weighted) path prefix of P_{S_x};
5 **for** *each signature $g_e \in \hat{P}_{S_x}$* **do**
6 **for** *each object $S_y \in \mathcal{L}(g_e)$* **do**
7 **if** VERIFY(S_x, S_y)=*true* **then** $\mathcal{A} \leftarrow (S_x, S_y)$;
8 $\mathcal{L}(g_e) \leftarrow S_x$;

Fig. 5. The Path-Prefix Based Filtering

Lemma 7. *Given two objects S_x and S_y, if their weighted path prefixes do not overlap, they cannot be similar.*

Proof. As $\hat{P}_{S_x} \cap \hat{P}_{S_y} = \phi$, without loss of generality, assume the last signature in $\hat{P}_{S_x}$ is smaller than the last signature of $\hat{P}_{S_y}$. Then all the signatures in $\hat{P}_{S_x}$ are smaller than the signatures in $P_{S_y} - \hat{P}_{S_y}$. Thus we have $\hat{P}_{S_x} \cap P_{S_y} = \phi$. As $\text{MSIM}(P_{S_x} - \hat{P}_{S_x}) < \tau|S_x|$, $\text{MSIM}(P_{S_x} \cap P_{S_y}) = \text{MSIM}(\hat{P}_{S_x} \cap P_{S_y}) + \text{MSIM}((P_{S_x} - \hat{P}_{S_x}) \cap P_{S_y}) < \tau|S_x|$. As $\text{MSIM}(P_{S_x} \cap P_{S_y}) < \tau|S_x|$, the fuzzy overlap of S_x and S_y is less than $\tau|S_x|$. Thus the similarity between S_x and S_y must be smaller than τ. Similarly, if the last signature in $\hat{P}_{S_x}$ is larger than the last signature of $\hat{P}_{S_y}$.Then all the signatures in $\hat{P}_{S_y}$ are smaller than the signatures in $P_{S_x} - \hat{P}_{S_x}$. Thus $\hat{P}_{S_y} \cap P_{S_x} = \phi$. As $\text{MSIM}(P_{S_y} - \hat{P}_{S_y}) < \tau|S_y|$, $\text{MSIM}(P_{S_y} \cap P_{S_x}) = \text{MSIM}(\hat{P}_{S_y} \cap P_{S_x}) + \text{MSIM}((P_{S_y} - \hat{P}_{S_y}) \cap P_{S_x}) < \tau|S_y|$. As $\text{MSIM}(P_{S_y} \cap P_{S_x}) < \tau|S_y|$, the fuzzy overlap of S_x and S_y is less than $\tau|S_y|$. Thus S_x and S_y cannot be similar. □

For example, consider S_4. Assume $\delta = 0.7$ and $\tau = 0.6$. $P_{S_4} = \{\texttt{PizzaHut}, \texttt{CA}, \texttt{KFC}, \texttt{Pizza}, \texttt{Fastfood}\}$. The path prefix of S_4 is $\{\texttt{PizzaHut}, \texttt{CA}, \texttt{KFC}, \texttt{Pizza}\}$. However, if we consider the similarity of each signature, $P_{S_4} = \{\texttt{PizzaHut}{:}\frac{4}{4}, \texttt{CA}{:}\frac{3}{3}, \texttt{KFC}{:}\frac{4}{4}, \texttt{Pizza}{:}\frac{3}{4}, \texttt{Fastfood}{:}\frac{3}{4}\}$. We can prune the last three signatures, because `KFC` and `Fastfood` are from the same element and the weight sum of `KFC` and `Pizza` is smaller than $\tau|S_4| = 1.8$. Thus the weighted path prefix of S_4 is $\{\texttt{PizzaHut}, \texttt{CA}\}$. Similarly, the path prefix of S_2 is $\{\texttt{Brooklyn}, \texttt{PaloAlto}, \texttt{Pizza}, \texttt{SanFrancisco}\}$, and the weighted path prefix of S_2 is $\{\texttt{Brooklyn}, \texttt{PaloAlto}, \texttt{Pizza}\}$. As the path prefixes of S_2 and S_4 share a common signature `Pizza`, we cannot prune this pair. If we use the weighted path prefix, they have no common signature, thus we can prune this pair.

4.2.3 Path Prefix Based Filtering Algorithm

Using (weighted) path prefix, we can devise a filtering algorithm. The pseudo code of the algorithm is shown in Figure 5. The algorithm first gets a global order of the path signatures (line 1), and then computes the prefix by removing signatures in a reverse way (lines 3-4). Then for each signature, it identifies the candidate from the inverted list of this signature (line 7). Finally it needs to append the object onto the inverted list (line 8).

5 ADAPTIVE VERIFICATION

We propose an adaptive verification algorithm to improve the performance of the verification step. We first propose a subgraph matching framework (Section 5.1), and then develop an adaptive verification algorithm (Section 5.2).

5.1 Subgraph Matching

As it is expensive to compute the maximum graph matching in order to compute the knowledge-aware similarity of a candidate pair, we propose a divide-and-conquer algorithm. Given a candidate $\langle S_x, S_y \rangle$, we first partition the two objects into some small sets, and then compute the similarities between small sets, and finally utilize these similarities to compute the similarity of the two objects. Since the subsets have smaller numbers of elements, the complexity of computing the similarity is much lower. Thus this method can significantly improve the verification performance.

Formally, we first group all elements by their node signatures. The elements with the same node signature will fall in the same group. For each group, S_x^i and S_y^i, we first compute the fuzzy overlap $S_x^i \widetilde{\cap}_\delta S_y^i$ on the two small sets and then we can prove that $S_x \widetilde{\cap}_\delta S_y = \sum_{i=1}^m S_x^i \widetilde{\cap}_\delta S_y^i$ as stated in Lemma 8. The basic idea is that the elements from different groups (i.e., with different node signatures) cannot be similar based on Lemma 1. Then we can easily compute the knowledge-aware similarity $\text{SIM}_\delta(S_x, S_y) = \frac{|S_x \widetilde{\cap}_\delta S_y|}{|S_x|+|S_y|-|S_x \widetilde{\cap}_\delta S_y|}$ based on $S_x \widetilde{\cap}_\delta S_y$.

Lemma 8. *Given two objects S_x and S_y, suppose they have m different node signatures. S_x^i (S_y^i) is the subset of S_x (S_y) with the same node signature, we have*

$$S_x \widetilde{\cap}_\delta S_y = \sum_{i=1}^{m} S_x^i \widetilde{\cap}_\delta S_y^i. \tag{5}$$

Proof. Based on Lemma 1, in the bigraph $G = ((S_x, S_y), E)$, all the elements in S_x^i are only connected with elements in S_y^i and all the elements in S_y^i are also only connected with elements in S_x^i. In other words, the subgraphs $G^i = ((S_x^i, S_y^i), E^i)$ and $G^j = ((S_x^j, S_y^j), E^j)$ have no edge. Thus the union of $G^i = ((S_x^i, S_y^i), E^i)$ is exactly $G = ((S_x, S_y), E)$. Hence, $S_x \widetilde{\cap}_\delta S_y = \sum_i S_x^i \widetilde{\cap}_\delta S_y^i$. □

5.2 Adaptive Verification

It is still expensive to compute the maximum matching of the subgraphs w.r.t. the subsets. To alleviate this problem, we propose an adaptive algorithm. Instead of directly computing the maximum matching, we estimate an upper bound and a lower bound of the subgraph matching. If the upper bound is smaller than a threshold, the candidate is not an answer and we can prune the candidate. If the lower bound is larger than a threshold, the candidate must be an answer without needing to compute the real maximum matching. Since there may be many subgraphs, we also discuss how to determine the order of computing the maximum matching of the subgraphs. Based on the order, we first compute the maximum matching of subgraphs that can facilitate the early termination and thus can avoid computing the maximum matching of other subgraphs.

Figure 6 illustrates the pseudo code. It first splits S_x and S_y into subgroups based on the node signatures (line 1). For each group, it estimates the lower bound $\mathcal{B}_i^l$ of fuzzy overlap of elements in the group (line 2) and the upper bound $\mathcal{B}_i^u$ (line 3). Then it computes the overall lower bound $\mathcal{B}^l = \sum \mathcal{B}_i^l$ and upper bound $\mathcal{B}^u = \sum \mathcal{B}_i^u$. If $\frac{\mathcal{B}^l}{|S_x|+|S_y|-\mathcal{B}^l} \geq \tau$, i.e., $\mathcal{B}^l \geq \tau_{S_x,S_y} = \lceil \frac{\tau}{1+\tau}(|S_x| + |S_y|) \rceil$, the candidate is an answer and the algorithm returns true (line 4). If $\frac{\mathcal{B}^u}{|S_x|+|S_y|-\mathcal{B}^u} < \tau$, i.e., $\mathcal{B}^u < \tau_{S_x,S_y} = \lceil \frac{\tau}{1+\tau}(|S_x|+|S_y|) \rceil$, the candidate is not an answer and the algorithm returns false (line 5). Then it sorts the groups based on the two bounds (line 6) and adaptively verifies the subgraphs (lines 8-12). The details on how to estimate the upper and lower bounds are respectively discussed in Section 5.2.1 and Section 5.2.2.

For example, consider S_8={`Pizza`, `KFC`, `Dominos`, `SanFrancisco`, `Manhattan`, `Brooklyn`} and S_9={`Fastfood`, `PizzaHut`, `BurgerKing`, `PaloAlto`, `MountainView`, `NewYork`} in Table 1. Assume $\delta = 0.6$ and $\tau = 0.6$. They can be partitioned into two groups S_8^1 = {`Pizza`, `KFC`, `Dominos`}, S_9^1 = {`Fastfood`, `PizzaHut`, `BurgerKing`} and S_8^2 = {`SanFrancisco`, `Manhattan`, `Brooklyn`}, S_9^2 = {`PaloAlto`, `MountainView`, `NewYork`}. We can compute the lower bounds of the two groups $\mathcal{B}_1^l = \frac{13}{6}$ and $\mathcal{B}_2^l = \frac{8}{5}$, thus the lower bound $\mathcal{B}^l = \frac{113}{30}$. As $\frac{\frac{113}{30}}{6+6-\frac{113}{30}} = \frac{113}{247} < 0.6$, we compute the upper bounds of the two groups $\mathcal{B}_1^u = \frac{9}{4}$ and $\mathcal{B}_2^u = \frac{12}{5}$, thus the upper bound $\mathcal{B}^u = \frac{93}{20}$. As $\frac{\frac{93}{20}}{6+6-\frac{93}{20}} = \frac{93}{147} > 0.6$, we still need to compute the real similarity. If we compute the second group first (we will discuss how to determine the order in Section 5.2.3), $\mathcal{B}^u = \frac{93}{20} - \frac{12}{5} + \frac{8}{5} = \frac{77}{20}$. As $\frac{\frac{77}{20}}{6+6-\frac{77}{20}} = \frac{77}{163} < 0.6$, we return false.

5.2.1 Upper Bound Estimation

We dig into the details on how to compute the maximum weight matching. In the maximum matching, on the one hand, either all the elements in S_x or all the elements in S_y are covered by the edges in the maximum matching. On the other hand, without loss of generality, suppose all the elements in S_x are covered by the edges. For each edge on element e, its weight should not be larger than the maximum weight of edges at e, thus we have the weight of the maximum matching is at most $\sum_{e_x \in S_x} \max \mathtt{w}_{e_x}$. Thus, we can estimate an upper bound $\mathcal{B}^u$ as follows.

For a candidate pair $\langle S_x, S_y \rangle$, we sum up the maximum weight of edges of each element in S_x(or S_y), i.e.,

$$\mathcal{B}^u = \min\Big(\sum_{e_x \in S_x} \max \mathtt{w}_{e_x}, \sum_{e_y \in S_y} \max \mathtt{w}_{e_y} \Big). \tag{6}$$

where $\mathtt{w}_{e_x}$ is the maximum similarity of edges for e_x. It is easy to prove that $\mathcal{B}^u$ is an upper bound of $|S_x \widetilde{\cap}_\delta S_y|$.

Recall the above example. Consider the second group of two objects S_8 and S_9: {`SanFrancisco`, `Manhattan`, `Brooklyn`}, {`PaloAlto`, `MountainView`, `NewYork`}. $\sum_{e_x \in S_{10}} \max \mathtt{w}_{e_x} = \frac{4}{5} + \frac{4}{5} + \frac{4}{5} = \frac{12}{5}$ and $\sum_{e_x \in S_{11}} \max \mathtt{w}_{e_x} = \frac{4}{5} + \frac{4}{5} + \frac{4}{5} = \frac{12}{5}$, thus the upper bound is $\mathcal{B}_2^u = \frac{12}{5}$.

5.2.2 Lower Bound Estimation

We first propose two greedy strategies to calculate the lower bounds and then combine them to give a tighter bound.

Greedy Algorithm: Maximum Weight. We devise a greedy algorithm to select the edge with the maximum weight. To avoid involving an element multiple times, after selecting

Algorithm 3: ADAPTIVEVERIFY($\mathcal{T}$, S_1, S_2, δ, τ)

Input: $\mathcal{T}$: The knowledge hierarchy; S_x, S_y: Two objects; δ: The element similarity threshold; τ: The object similarity threshold

Output: True or False

1 Split S_x and S_y into m subsets by node signatures ;
2 $\mathcal{B}_i^l$ = ESTIMATELOWERBOUND(S_x^i, S_y^i) ;
3 $\mathcal{B}_i^u$ = ESTIMATEUPPERBOUND(S_x^i, S_y^i) ;
4 **if** $\mathcal{B}^l = \sum \mathcal{B}_i^l \geq \tau_{S_x,S_y}$ **then** return True ;
5 **if** $\mathcal{B}^u = \sum \mathcal{B}_i^u < \tau_{S_x,S_y}$ **then** return False ;
6 Sort i by $\mathcal{B}_i^u - \mathcal{B}_i^l$;
7 **for** $i = 1$ *to* m **do**
8 $\quad s = \text{REALSIM}(S_x^i, S_y^i)$;
9 $\quad \mathcal{B}^u = \mathcal{B}^u - \mathcal{B}_i^u + s$;
10 $\quad$ **if** $\mathcal{B}^u < \tau_{S_x,S_y}$ **then** return False ;
11 $\quad \mathcal{B}^l = \mathcal{B}^l - \mathcal{B}_i^l + s$;
12 $\quad$ **if** $\mathcal{B}^l \geq \tau_{S_x,S_y}$ **then** return True ;

Fig. 6. Adaptive Verification Algorithm

an edge, we remove the two elements on the edge. Formally, given a bigraph $G = ((S_x, S_y), E)$, we find the edge with the maximum weight, and remove this edge and the two elements of the edge from the bigraph. We repeat this process until there is no edge. Then we sum up the weight of all removed edges, which is a lower bound of $|S_x \widetilde{\cap}_\delta S_y|$. As each edge will be processed once, we use a min-heap to keep the edge, and the complexity is $\mathcal{O}(|S_x| + |S_y| + |E| \log |E|)$. We denote this lower bound as l_w.

Greedy Algorithm: Maximum Degree. We devise a greedy algorithm to cover as many elements as possible. Formally, given a bigraph $G = ((S_x, S_y), E)$, we find the element e_x with the smallest degree in S_x. Then we find the element e_y with the smallest degree connected to e_x in S_y. We select (e_x, e_y) and delete e_x, e_y from the bigraph. We repeat this process until there is no element in S_x. Then we sum up the weights of the selected edges and this is a lower bound. The time complexity is $\mathcal{O}((|S_x|+|S_y|)\log(|S_x|+|S_y|)+|E|)$. We denote this lower bound as l_e.

We can combine these two lower bounds and get a tighter lower bound $\mathcal{B}^l = \max(l_w, l_e)$.

5.2.3 Determining The Order of Subgraphs

A good order of computing the maximum matching of the subgraphs can early terminate the loop in Lines 9 - 14 in Algorithm 3. Obviously, we want to first check the subgraphs whose estimated upper and lower bounds are rather loose. To this end, we can sort the subgraphs based on $\mathcal{B}^u - \mathcal{B}^l$. Obviously, the larger $\mathcal{B}^u - \mathcal{B}^l$ is, the estimated two bounds are looser. Thus we first compute the maximum matching of the subgraphs with the largest $\mathcal{B}^u - \mathcal{B}^l$.

6 EXTENSIONS

6.1 From Self Join to R-S Join

Given two collections of objects $\mathcal{R}$ and $\mathcal{S}$, we first generate the signatures of all the objects and fix a global order. Then we utilize the inverted lists to index objects in one collection, e.g., $\mathcal{R}$. (We will index the set with larger size because we can search the smaller dataset by utilizing the indexes on the larger dataset.) Next for each object in the other collection, e.g., $\mathcal{S}$, we generate its signature and the objects on the inverted list of each signature is a candidate of this object. Finally we verify the candidates to generate the final answer.

6.2 Supporting Other Element Similarity Metrics

Our method can support other functions to define element similarity (Equation 1) if they depends on d_{e_x}, d_{e_y}, d_{e_x,e_y}. For example, our method can support a famous metrics Wu & Palmer [39] in the AI community, which calculates similarity by considering the depths of the two elements.

$$\text{SIM}(e_x, e_y) = \frac{2 * d_{e_x,e_y}}{d_{e_x} + d_{e_y}}. \quad (7)$$

Suppose e_x and e_y are two *different elements*. If they are similar, we have $\delta \leq \frac{2*d_{e_x,e_y}}{d_{e_x}+d_{e_y}} \leq \frac{2*d_{e_x,e_y}}{2*d_{e_x,e_y}+1}$, thus $d_{e_x,e_y} \geq \frac{\delta}{2(1-\delta)}$. Thus our techniques can support this function.

6.3 Supporting Other Set Similarity Metrics

We can utilize any similarity functions to replace Jaccard to define the knowledge-aware similarity (Equation 3). Our algorithm relies only on τ_S and τ_{S_x,S_y}, thus we discuss how to compute the two values.

Dice Similarity: $\text{SIM}_\delta(S_x, S_y) = \frac{2 \cdot |S_x \widetilde{\cap}_\delta S_y|}{|S_x|+|S_y|} \geq \tau$.
$\tau_{S_x,S_y} = \lceil \frac{\tau}{2} \cdot (|S_x| + |S_y|) \rceil$.
$\tau_{S_x} = \lceil \frac{\tau}{2-\tau} \cdot |S_x| \rceil$ as $\frac{2 \cdot |S_x \widetilde{\cap}_\delta S_y|}{|S_x|+|S_y|} \geq \tau \rightarrow |S_x \widetilde{\cap}_\delta S_y| \geq \frac{\tau}{2-\tau} \cdot |S_x|$.

Cosine Similarity: $\text{SIM}_\delta(S_x, S_y) = \frac{|S_x \widetilde{\cap}_\delta S_y|}{\sqrt{|S_x| \cdot |S_y|}} \geq \tau$.
$\tau_{S_x,S_y} = \lceil \tau \cdot \sqrt{|S_x| \cdot |S_y|} \rceil$.
$\tau_{S_x} = \lceil \tau^2 \cdot |S_x| \rceil$ as $\frac{|S_x \widetilde{\cap}_\delta S_y|}{\sqrt{|S_x| \cdot |S_y|}} \geq \tau \rightarrow |S_x \widetilde{\cap}_\delta S_y| \geq \tau^2 \cdot |S_x|$.

6.4 Supporting One Element Matching Multiple Nodes

We first discuss how to extend node signatures to support the case of one element matching multiple nodes. For each element e, we first find its mapping nodes and generate node signatures for each node. Then we generate the signature set of object S by computing the union of its element's node signatures. We fix a global order of all node signatures. Next we sort the signature set of e and generate the node prefix of S by removing the node signatures in a reverse way, until there are $\tau_S - 1$ elements that have node signatures removed. Thus the node signature based filtering technique can be used. For verification, we still partition the signature sets based on the node signature. Note that if two subsets have common elements, we need to merge them. Then our verification techniques can use used.

Similarly, our (weighted) path prefix, subgraph matching, and adaptive verification algorithms can be used to support the case of one element matching multiple nodes.

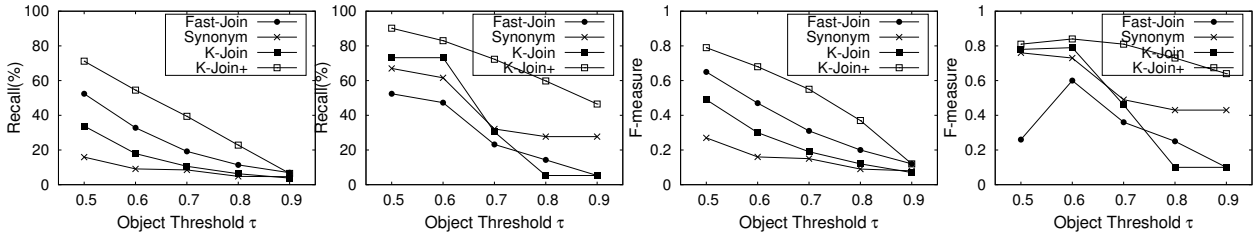
6.5 Supporting DAG

If the knowledge hierarchy is a DAG, we can transform the DAG to a tree. For each node in the DAG with multiple parents, e.g., v parents, we duplicate the node v times and take each of them as a child of these v parents. In this way, we can transform the DAG to a tree. Then an element may map to multiple nodes in the tree and thus we can use the techniques in the above section to support DAG.

7 EXPERIMENTAL STUDY

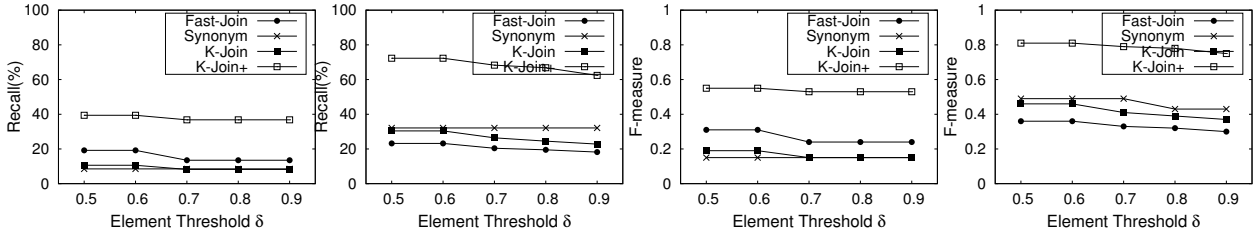
7.1 Experimental Setup

Knowledge Hierarchy. We used a real-world knowledge hierarchy with POI (points of interest) and location categories, e.g., food and addresses, which were crawled from Factual (www.factual.com) as shown in Table 2.



(a) Recall (Pub) (b) Recall (Res) (c) F-measure (Pub) (d) F-measure (Res)

Fig. 7. Effectiveness: Comparison with State-of-the-arts by Varying $\tau(\delta = 0.5)$.



(a) Recall (Pub) (b) Recall (Res) (c) F-measure (Pub) (d) F-measure (Res)

Fig. 8. Effectiveness: Comparison with State-of-the-arts by Varying $\delta(\tau = 0.7)$.

TABLE 2
Knowledge Hierarchy.

# Nodes	Height	Avg Fanout	Max Fanout	Min Fanout
4222	6	7	49	1

TABLE 3
Datasets.

Datasets	Size	AvgLen	MaxLen	MinLen	AvgDep
Paper	1879	6	16	4	3
Restaurant	864	4	4	4	5
POI(small)	100,000	11	21	2	4
POI(large)	1,000,000	11	21	2	4
Tweet (small)	100,000	8	23	2	5
Tweet (large)	1,000,000	7	27	2	5

Datasets. We used four real-world datasets: Pub, Res, POI and Tweet. The Pub dataset contained 1879 papers and each paper was composed of author, title, journal, date, publisher, and institution. The Res contained 864 restaurants and each restaurant was described by name, address, city and food. These two datasets had ground truths [34]. The inconsistencies in the Pub dataset were due to typos or abbreviations; and the errors in the Res dataset were due to synonyms and knowledge hierarchy (e.g., "American food" and "Californian food"). We used them to evaluate the effectiveness. The POI dataset contained 1 million POIs and each POI included address, category and name. The Tweet dataset contained 1 millions crawled tweets which included address and category. As the Pub and Res datasets were too small, we utilized POI and Tweet to compare the efficiency. As the baseline could not support large datasets, we selected 100,000 records from the two datasets and generated two small datasets. The details were shown in Table 3.

Baseline. FastJoin was a state-of-the-art method [32], which extended the set similarity functions to tolerate the edit errors between elements. Synonym was another state-of-the-art method [23], which used synonyms to measure string similarities where an element mapped to any of its synonyms. Crowd was a crowdsourcing based method [29], which utilized human knowledge to improve the quality.

All the algorithms were implemented in C++. The experiments were conducted on a Ubuntu server with two Intel Xeon X5670 CPUs (2.93GHz) and 64GB RAM.

7.2 Evaluation on Effectiveness

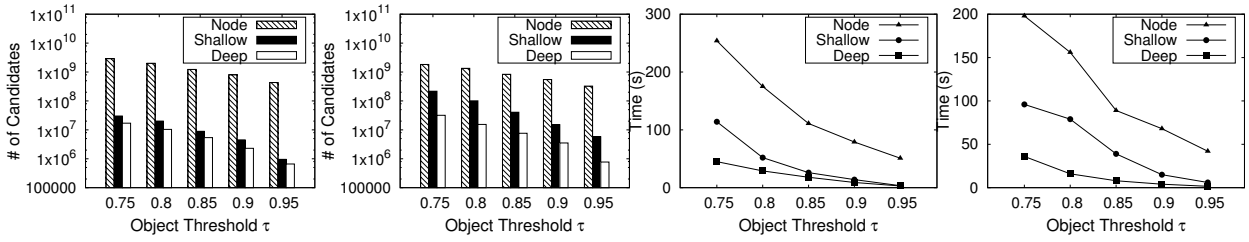
We evaluated the result quality on the Pub and Res datasets. We compared FastJoin, Synonym, K-Join (an element maps one tree node) and K-Join$^+$ (an element maps multiple tree nodes using synonyms and approximating matching), and Crowd. For Pub, we constructed a 3-level hierarchy, e.g., paper, research area, conference. For Res, we used the hierarchy in Table 2. Table 4 shows the results.

TABLE 4
Quality on Pub and Res ($\delta = 0.5, \tau = 0.6$)

	Pub			Res		
	Precision	Recall	F-measure	Precision	Recall	F-measure
FastJoin	87.6	52.4	65.1	81.5	47.3	60.0
K-Join	89.1	33.8	49.2	85.8	73.2	79.2
K-Join$^+$	88.4	71.2	80.1	85.3	83.0	84.0
Synonym	89.1	15.9	27.2	89.5	61.6	76.1
Crowd	68.8	95.0	80.1	81.4	88.8	84.9

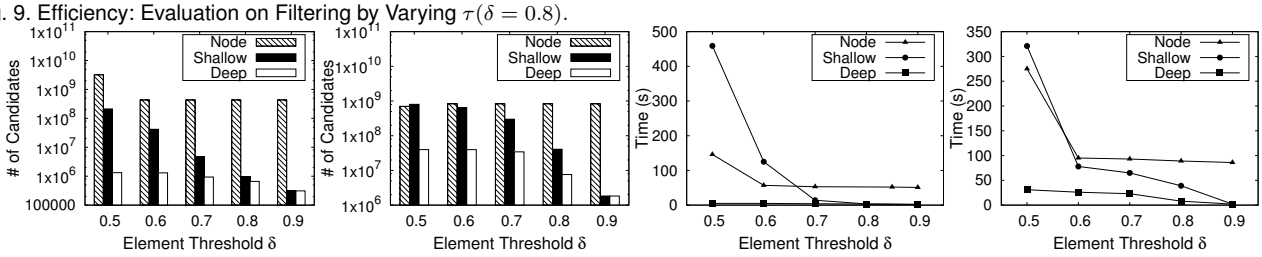
From the results, we had the following observations. Firstly, our method had much higher recall than FastJoin and Synonym, because we could use the knowledge to enrich the data and thus found more similar pairs. For example, given two restaurants with "Californian" food and "American" food that referred to the same restaurant, our method could use the knowledge hierarchy to handle them easily while FastJoin and Synonym cannot. As another example, two journals with names "Artif Intelligence" and "Artificial Intelli", could map to the same category "Artificial Intelligence". FastJoin performed worse on the Res dataset with more synonyms and hierarchy, because although it tried to find more similar pairs by tolerating the edit errors, it cannot tolerate the inconsistencies that an entity had different representations. Synonym performed worse on the Pub dataset with more typos, because while it utilized synonyms to improve the effectiveness, it used a token-based measure which ignored the similarity between tokens with typos. Thus, both of the two competitors cannot use the knowledge hierarchy to improve the result quality, while in contrast our method utilized the knowledge to address this entity-resolution issue. K-Join$^+$ had higher recall than K-Join, as it could match each element to multiple nodes by tolerating typos and synonyms. Due to the flexible matching, K-Join$^+$ combined the strengths of FastJoin, Synonym and K-Join in the same framework magically. Secondly, our method had much higher F-measure than FastJoin and Synonym, because these methods had similar precision rate. Thirdly, our method had nearly the same quality with Crowd, because we can utilize knowledge (similar to human ability) to improve the quality.

Next we varied the thresholds τ and δ and compared different methods. As Crowd did not utilize any threshold,



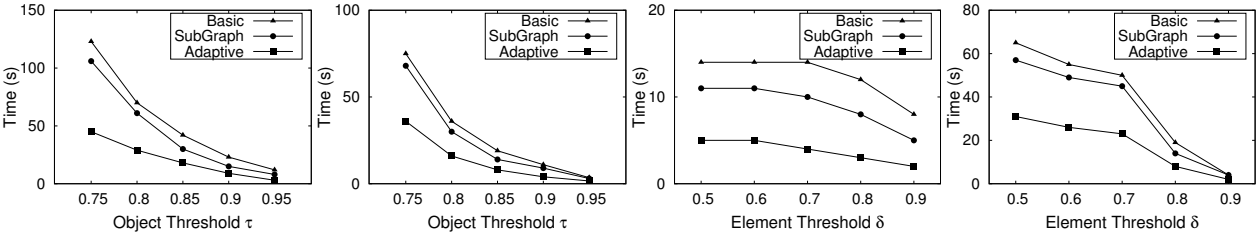
(a) Candidate (POI) (b) Candidate (Tweet) (c) Time (POI) (d) Time (Tweet)

Fig. 9. Efficiency: Evaluation on Filtering by Varying $\tau(\delta = 0.8)$.



(a) Candidate (POI, $\tau = 0.95$) (b) Candidate (Tweet, $\tau = 0.85$) (c) Time (POI, $\tau = 0.95$) (d) Time (Tweet, $\tau = 0.85$)

Fig. 10. Efficiency: Evaluation on Filtering by Varying δ.



(a) Time (POI, $\delta = 0.8$) (b) Time (Tweet, $\delta = 0.8$) (c) Time (POI, $\tau = 0.95$) (d) Time (Tweet, $\tau = 0.85$)

Fig. 11. Efficiency: Evaluation on Verification by Varying τ and δ.

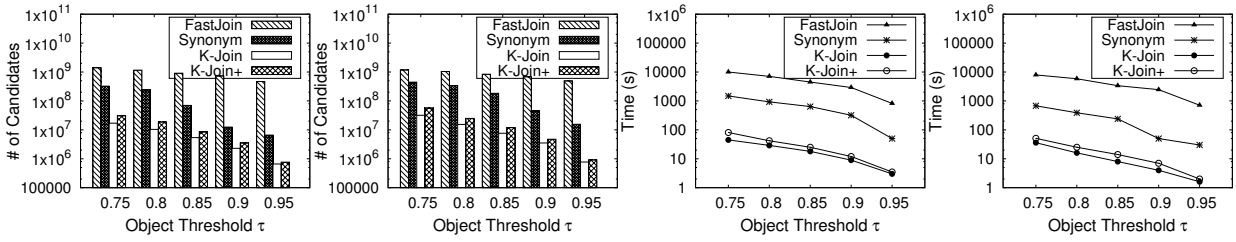
we only showed other four methods. As the precision of these methods was similar, we only showed recall and F-measure, as illustrated in Figures 7 and 8. We had the following observations. Firstly, with the increase of τ, the recall decreased as only a few pairs were returned for a large threshold; the precision slightly increased as the returned pairs had large similarities and thus large possibilities to be true similar pairs; and F-measure also decreased, because the recall significantly reduced while the precision slightly increased. The large increase for F-measure of `FastJoin` from $\tau = 0.5$ to $\tau = 0.6$ was due to that the precision rate increased from 17% to 81.5%. Secondly, with the increase of δ, the recall slightly reduced, as fewer similar entities were found and thus fewer results were reported. The precision would increase as we used more similar entities with the increase of δ, and thus we had similar F-measure.

7.3 Evaluation on Efficiency

7.3.1 Evaluation on Filtering

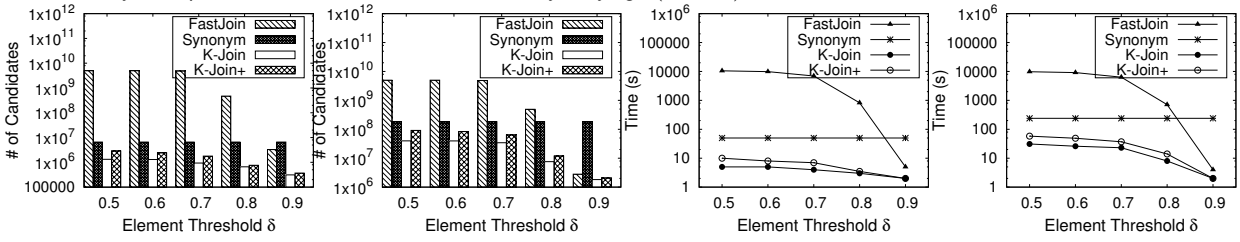
We first evaluated the filtering step and compared three methods: (1) `Node`, using the node signature to generate candidates; (2) `Shallow`, using the shallow signatures to generate candidates; (3) `Deep`, using the deep signatures to generate candidates. Figures 9 and 10 showed the results.

We made the following observations. Firstly, both `Shallow` and `Deep` had smaller numbers of candidates than `Node`. For example, when $\tau = 0.85$ and $\delta = 0.8$ on the POI dataset, the numbers of candidates of `Node`, `Shallow`, and `Deep` were respectively 1.2 billions, 9 millions, 5 millions. This was because `Node` did not utilize the depth of elements and generated coarse-grained signatures while `Shallow` and `Deep` utilized the depth information to generate fine-grained signatures which effectively pruned many dissimilar pairs. Secondly, the number of candidates of `Deep` was smaller than that of `Shallow`, especially on the `Tweet` dataset. For example, for $\tau = 0.85$ and $\delta = 0.8$ on the `Tweet` dataset, the number of candidates of `Deep` was 40 millions and that of `Shallow` was only 7 millions. This was because `Deep` used deeper nodes as the signatures, which generated fine-grained signatures, and thus it could prune many dissimilar pairs and achieved better performance than `Shallow`. In addition, the average depth of the elements on the `Tweet` dataset was larger than that of POI, and thus the performance gap of `Deep` and `Shallow` was more remarkable on `Tweet`. Thirdly, the three methods had similar trends in terms of efficiency, i.e., `Deep` was better than `Shallow`, which in turn was better than `Node`, because if there were more candidates, the verification step took more time to verify them. Fourthly, the number of candidates tended to decrease as the object similarity threshold τ increased, because for a larger threshold, there would be smaller numbers of candidates (and answers). Fifthly, the number of candidates decreased as the element similarity threshold δ increased, as for a larger threshold, there would be more similar elements and thus more candidates. The number of signatures for elements relied on δ and that for objects relied on τ. The two thresholds affected the candidate number. Sixthly, when the element similarity threshold δ was small, the performance of `Shallow` was comparable to that of `Node`, but `Deep` was much better than `Node` and `Shallow`. This was because for a small δ, both `Shallow` and `Node` generated shallow-level signatures while `Deep` generated deep-level signatures. Obviously the deep-level signatures had higher pruning power than the shallow-level signatures. With the increase of δ, the gap between `Deep` and `Shallow` became smaller, because for a large δ, both of them



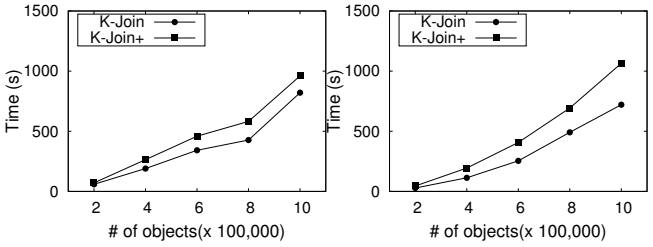
(a) Candidate (POI) (b) Candidate (Tweet) (c) Time (POI) (d) Time (Tweet)

Fig. 12. Efficiency: Comparison with State-of-the-art Method by Varying $\tau(\delta = 0.8)$.



(a) Candidate (POI, $\tau = 0.95$) (b) Candidate (Tweet, $\tau = 0.85$) (c) Time (POI, $\tau = 0.95$) (d) Time (Tweet, $\tau = 0.85$)

Fig. 13. Efficiency: Comparison with State-of-the-art Method by Varying δ.



(a) Time (POI, $\tau = 0.95$) (b) Time (Tweet, $\tau = 0.85$)

Fig. 14. Scalability ($\delta = 0.8$).

could generate deep-level signatures. The performance gap between Deep and Node became larger, as Node cannot generate high-quality signatures.

7.3.2 Evaluation on Verification

We evaluated the verification step. We compared three algorithms: (1) Basic, the basic verification algorithm; (2) SubGraph, the subgraph matching; (3) Adaptive, the adaptive verification algorithm. Figure 11 showed the result.

Firstly, Adaptive was better than SubGraph, which in turn outperformed Basic. For example, for $\tau = 0.8$ and $\delta = 0.8$ on the POI dataset, Basic took 70 seconds, and SubGraph took 61 seconds and Adaptive improved to 29 seconds. This was because SubGraph reduced the verification complexity compared with Basic, and Adaptive used the upper and lower bounds to achieve early termination and avoided many unnecessary computations. Secondly, the performance gap between them became smaller with the increase of threshold τ. For small thresholds τ, as for small thresholds there were larger numbers of candidates, Adaptive and SubGraph had large opportunity to do pruning; while for large thresholds, there were smaller numbers of candidates and there was not enough room to improve verification. Thirdly, with the increase of threshold δ, the three methods achieved better performance, as there were smaller numbers of candidates (as well as answers).

7.3.3 Comparison with the State-of-the-art Method

We compared our method with the state-of-the-art works FastJoin and Synonym. As both FastJoin and Synonym could not support large datasets, we compared the performance on the POI (small) and Tweet (small) datasets. Figures 12 and 13 showed the results. We made the following observations. Firstly, K-Join and K-Join$^+$ significantly outperformed FastJoin and Synonym in terms of both candidate numbers and efficiency, even by 2-3 orders of magnitude. For example, for $\tau = 0.95$ and $\delta = 0.8$ on the Tweet dataset, the numbers of candidates of FastJoin, Synonym, K-Join, K-Join$^+$ were respectively 492 millions, 16 millions, 0.7 millions, 0.9 millions. FastJoin took 711 seconds and Synonym took 30 seconds while our methods only took 2 seconds. The main reasons were: (1) we used efficient node/path prefix filtering to find candidates, which was much better than FastJoin and Synonym (which generated too many signatures), and thus our method generated smaller numbers of candidates than FastJoin and Synonym; (2) we improved the verification step carefully using subgraph matching and adaptive filtering (which were rather efficient to improve the efficiency as demonstrated in the above experiments), while FastJoin and Synonym did not optimize the verification. Secondly, K-Join had slightly better performance than K-Join$^+$, as K-Join$^+$ generated more candidates than K-Join by tolerating more errors. Thirdly, as δ increased, the performance gap between our method and the two competitors tended to decrease, because for a large threshold, there would be smaller numbers of candidates and there was no enough room to improve the efficiency; but for a small threshold, there were large numbers of candidates. Synonym kept the same performance for different δ, as it did not use the element similarity threshold and it used exact token matching to support synonyms.

7.3.4 Evaluation on Scalability

We used the two large datasets, POI (large) and Tweet (large), to evaluate the scalability of our methods. Figure 14 showed the overall time by varying the number of objects. From the results, we can see that our method scaled well and achieved nearly linear scalability. This was attributed to our efficient filtering methods to prune as many dissimilar pairs as possible and adaptive subgraph matching algorithm to avoid verification cost as much as possible. K-Join$^+$ took a little more time than K-Join as it found more results.

8 Conclusion

We study a new problem, knowledge-aware similarity join. We propose a new similarity metric to quantify the knowledge-aware similarity on elements and objects. We propose a filter-and-verification framework to efficiently

identify similar pairs. In the filter step, we devise node/path signatures to prune large numbers of dissimilar pairs. In the verification step, we propose subgraph matching and develop an adaptive verification algorithm. Experimental results show that our method significantly outperforms baseline algorithms in both efficiency and effectiveness.

Acknowledgement. This work was partly supported by the 973 Program of China (2015CB358700), NSF of China (61373024, 61422205, 61472198), Huawei, Shenzhou, Tencent, FDCT/116/2013/A3, MYRG105(Y1-L3)-FST13-GZ, 863 Program(2012AA012600), and Chinese Special Project of Science & Technology(2013zx01039-002-002).

REFERENCES

[1] H. Andrade and J. H. Saltz. Towards a knowledge base management system (kbms): An ontology-aware database management system (dbms). In *SBBD*, pages 27–39, 1999.
[2] A. Arasu, S. Chaudhuri, and R. Kaushik. Transformation-based framework for record matching. In *ICDE*, pages 40–49, 2008.
[3] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *WWW*, pages 131–140, 2007.
[4] P. Ceravolo, E. Damiani, and M. Leida. Semantic-aware, ontology mediated mapping generation system. 2015.
[5] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *ICDE*, page 5, 2006.
[6] D. Deng, G. Li, and J. Feng. A pivotal prefix based filtering algorithm for string similarity search. In *SIGMOD Conference*, pages 673–684, 2014.
[7] D. Deng, G. Li, J. Feng, and W.-S. Li. Top-k string similarity search with edit-distance constraints. In *ICDE*, pages 925–936, 2013.
[8] D. Deng, G. Li, S. Hao, J. Wang, and J. Feng. Massjoin: A mapreduce-based method for scalable string similarity joins. In *ICDE*, pages 340–351, 2014.
[9] D. Deng, G. Li, H. Wen, and J. Feng. An efficient partition based method for exact set similarity joins. *PVLDB*, 9(4):360–371, 2015.
[10] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16, 2007.
[11] J. Feng, J. Wang, and G. Li. Trie-join: a trie-based method for efficient string similarity joins. *VLDB J.*, 21(4):437–461, 2012.
[12] M. Hadjieleftheriou, X. Yu, N. Koudas, and D. Srivastava. Hashed samples: selectivity estimators for set similarity selection queries. *PVLDB*, 1(1):201–212, 2008.
[13] S. Ji, G. Li, C. Li, and J. Feng. Efficient interactive fuzzy keyword search. In *WWW*, pages 371–380, 2009.
[14] Y. Jiang, G. Li, and J. Feng. String similarity joins: An experimental evaluation. *PVLDB*, 2014.
[15] Y. Kim and K. Shim. Efficient top-k algorithms for approximate substring matching. In *SIGMOD Conference*, pages 385–396, 2013.
[16] C. Li, J. Lu, and Y. Lu. Efficient merging and filtering algorithms for approximate string searches. In *ICDE*, pages 257–266, 2008.
[17] C. Li, B. Wang, and X. Yang. Vgram: Improving performance of approximate queries on string collections using variable-length grams. In *VLDB*, pages 303–314, 2007.
[18] G. Li, D. Deng, J. Wang, and J. Feng. Pass-join: A partition-based method for similarity joins. *PVLDB*, 5(3):253–264, 2011.
[19] G. Li, J. He, D. Deng, and J. Li. Efficient similarity join and search on multi-attribute data. In *SIGMOD*, pages 1137–1151, 2015.
[20] G. Li, S. Ji, C. Li, and J. Feng. Efficient type-ahead search on relational data: a TASTIER approach. In *SIGMOD*, pages 695–706, 2009.
[21] G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou. EASE: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *SIGMOD*, pages 903–914, 2008.
[22] S. Liu, G. Li, and J. Feng. A prefix-filter based method for spatio-textual similarity join. *IEEE Trans. Knowl. Data Eng.*, 26(10):2354–2367, 2014.
[23] J. Lu, C. Lin, W. Wang, C. Li, and H. Wang. String similarity measures and joins with synonyms. In *SIGMOD*, pages 373–384, 2013.
[24] J. Qin, W. Wang, Y. Lu, C. Xiao, and X. Lin. Efficient exact edit similarity query processing with the asymmetric signature scheme. In *SIGMOD Conference*, pages 1033–1044, 2011.
[25] S. Sarawagi and A. Kirpal. Efficient set joins on similarity predicates. In *SIGMOD Conference*, pages 743–754, 2004.
[26] K. Slabbekoorn, L. Hollink, and G.-J. Houben. Domain-aware ontology matching. In *The Semantic Web–ISWC 2012*, pages 542–558. Springer, 2012.
[27] V. Verroios and H. Garcia-Molina. Entity resolution with crowd errors. In *ICDE*, pages 219–230, 2015.
[28] N. Vesdapunt, K. Bellare, and N. N. Dalvi. Crowdsourcing algorithms for entity resolution. *PVLDB*, 7(12):1071–1082, 2014.
[29] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. Crowder: Crowdsourcing entity resolution. *PVLDB*, 5(11):1483–1494, 2012.
[30] J. Wang, G. Li, D. Deng, Y. Zhang, and J. Feng. Two birds with one stone: An efficient hierarchical framework for top-k and threshold-based string similarity search. In *ICDE*, 2015.
[31] J. Wang, G. Li, and J. Feng. Trie-join: Efficient trie-based string similarity joins with edit-distance constraints. *PVLDB*, 3(1):1219–1230, 2010.
[32] J. Wang, G. Li, and J. Feng. Fast-join: An efficient method for fuzzy token matching based string similarity join. In *ICDE*, pages 458–469, 2011.
[33] J. Wang, G. Li, and J. Feng. Can we beat the prefix filtering?: an adaptive framework for similarity join and search. In *SIGMOD Conference*, pages 85–96, 2012.
[34] J. Wang, G. Li, T. Kraska, M. J. Franklin, and J. Feng. Leveraging transitive relations for crowdsourced joins. In *SIGMOD*, pages 229–240, 2013.
[35] X. Wang, X. Ding, A. K. H. Tung, and Z. Zhang. Efficient and effective knn sequence search with approximate n-grams. In *PVLDB*, volume 7, pages 1–12, 2014.
[36] S. E. Whang and H. Garcia-Molina. Joint entity resolution. In *ICDE*, pages 294–305, 2012.
[37] S. E. Whang and H. Garcia-Molina. Incremental entity resolution on rules and data. *VLDB J.*, 23(1):77–102, 2014.
[38] S. E. Whang, D. Menestrina, G. Koutrika, M. Theobald, and H. Garcia-Molina. Entity resolution with iterative blocking. In *SIGMOD*, pages 219–232, 2009.
[39] Z. Wu and M. Palmer. Verbs semantics and lexical selection. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, pages 133–138. Association for Computational Linguistics, 1994.
[40] C. Xiao, W. Wang, and X. Lin. Ed-join: an efficient algorithm for similarity joins with edit distance constraints. *PVLDB*, 1(1):933–944, 2008.
[41] C. Xiao, W. Wang, X. Lin, and J. X. Yu. Efficient similarity joins for near duplicate detection. In *WWW*, pages 131–140, 2008.
[42] X. Yang, Y. Wang, B. Wang, and W. Wang. Local filtering: Improving the performance of approximate queries on string collections. In *SIGMOD*, pages 377–392, 2015.
[43] Z. Yang, J. Yu, and M. Kitsuregawa. Fast algorithms for top-k approximate string matching. In *AAAI*, 2010.
[44] M. Yu, G. Li, D. Deng, and J. Feng. String similarity search and join: a survey. *Frontiers of Computer Science*, 10(3):399–417, 2016.
[45] Z. Zhang, M. Hadjieleftheriou, B. C. Ooi, and D. Srivastava. Bed-tree: an all-purpose index structure for string similarity search based on edit distance. In *SIGMOD*, pages 915–926, 2010.

Zeyuan Shang is a student at Tsinghua University. He received his Bachelor Degree in Computer Science from Tsinghua University, Beijing, China in 2015. His research interests mainly include data cleaning and integration.

Yaxiao Liu is a PhD student at Tsinghua University. He got Bachelor Master degree in Computer Science from Tsinghua University, Beijing, China. His research interests mainly include large-scale stream data management, and data cleaning and integration.

Guoliang Li is currently working as an associate professor in the Department of Computer Science, Tsinghua University, Beijing, China. He received his PhD degree in Computer Science from Tsinghua University, Beijing, China in 2009. His research interests mainly include data cleaning and integration, spatial databases, and crowdsourcing.

Jianhua Feng received his B.S., M.S. and PhD degrees in Computer Science from Tsinghua University. He is currently working as a professor of Department Computer Science in Tsinghua University. His main research interests include large-scale data management and analysis.