Extending String Similarity Join to Tolerant Fuzzy Token Matching

JIANNAN WANG, GUOLIANG LI, and JIANHUA FENG, Tsinghua University

String similarity join that finds similar string pairs between two string sets is an essential operation in many applications and has attracted significant attention recently in the database community. A significant challenge in similarity join is to implement an effective fuzzy match operation to find all similar string pairs which may not match exactly. In this article, we propose a new similarity function, called *fuzzy-token-matching-based similarity* which extends token-based similarity functions (e.g., jaccard similarity and cosine similarity) by allowing fuzzy match between two tokens. We study the problem of similarity join using this new similarity function and present a signature-based method to address this problem. We propose new signature schemes and develop effective pruning techniques to improve the performance. We also extend our techniques to support weighted tokens. Experimental results show that our method achieves high efficiency and result quality and significantly outperforms state-of-the-art approaches.

Categories and Subject Descriptors: H.2.8 [**Database Management**]: Database Applications; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Search process, Clustering*

General Terms: Algorithms, Performance, Experiment

Additional Key Words and Phrases: String similarity join, similarity function, signature scheme, fuzzy token matching-based similarity, weighted tokens

ACM Reference Format:
Jiannan Wang, Guoliang Li, and Jianhua Feng. 2014. Extending string similarity join to tolerant fuzzy token matching. ACM Trans. Datab. Syst. 39, 1, Article 7 (January 2014), 45 pages.
DOI: http://dx.doi.org/10.1145/2535628

1. INTRODUCTION

Similarity join has become a fundamental operation in many applications, such as data integration and cleaning, near duplicate object detection and elimination, and collaborative filtering [Xiao et al. 2008a]. In this article, we study string similarity join which, given two sets of strings, finds all *similar* string pairs from each set. Existing similarity-join approaches [Sarawagi and Kirpal 2004; Chaudhuri et al. 2006; Arasu et al. 2006; Bayardo et al. 2007, Xiao et al. 2008a, 2009; Wang et al. 2010] mainly use the following functions to quantify similarity of two strings.

Token-Based Similarity Functions. They first tokenize strings as token sets (i.e., bag of words), and then quantify the similarity based on the token sets, such as jaccard similarity and cosine similarity. Usually if two strings are similar, their token sets

This work was partly supported by the National Natural Science Foundation of China under Grant Nos. 61003004, 61272090, and 61373024, the National Grand Fundamental Research 973 Program of China under Grant No. 2011CB302206, a project of Tsinghua University under Grant No. 20111081073, Tsinghua-Tencent Joint Laboratory for Internet Innovation Technology, and the "NExT Research Center" funded by MDA, Singapore, under Grant No. WBS:R-252-300-001-490.
Authors' addresses: J. Wang (corresponding author), G. Li, and J. Feng, Department of Computer Science and Technology, Tsinghua National Laboratory for Information Science and Technology (TNList), Tsinghua University, Beijing 100084, China; email: {wjn08, liguoliang, fengjh}@tsinghua.edu.cn.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2014 ACM 0362-5915/2014/01-ART7 $15.00
DOI: http://dx.doi.org/10.1145/2535628

should have a large overlap. Token-Based Similarity Functions have a limitation that they only consider exact match of two tokens and neglect fuzzy match of two tokens. Note that many datasets contain typos and inconsistences in their tokens and may have many mismatched token pairs that refer to the same token. For example, consider two strings nba mcgrady and macgrady nba. Their token sets are, respectively, {nba, mcgrady} and {macgrady, nba}. The two token sets contain a mismatched token pair (mcgrady, macgrady). As an example, the jaccard similarity between the two strings is 1/3 (i.e., the ratio of the number of tokens in their intersection to that in their union). Although the two strings are very similar, their jaccard similarity is very low.

In order to make token-based similarity cope with typos, we can tokenize a string into a q-gram set rather than bags of words. We adopt the most common way to generate q-gram sets [Chandel et al. 2007; Hassanzadeh and Miller 2009]. Given a string, the method first extends each word in the string to a new word by prefixing and suffixing $q - 1$ special symbols (e.g., $), and then generates all substrings with q characters for the new word. The q-gram set for the string is obtained by merging the generated substrings for all new words. For example, consider two strings nba mcgrady and macgrady nba, and suppose $q = 3$. The first string can be tokenized as {$$n, nb, nba, ba, a$$, $$m, $mc, mcg, cgr, gra, rad, ady, dy$, y$$}, and the second string can be tokenized as {$$m, $ma, mac, acg, cgr, gra, rad, ady, dy$, d$$, $$n, nb, nba, ba, a$$}. Since their gram sets share twelve common grams, their jaccard similarity based on 3-gram tokenization is $\frac{12}{17}$, which is a reasonable similarity value for the two strings. Typically, q-gram based approaches [Chandel et al. 2007; Hassanzadeh and Miller 2009] tend to select a smaller gram size q in order to capture typos, but on the other hand, a smaller q may make the strings match a lot of grams that are generated from dissimilar words. Therefore, even with the q-gram tokenization, token-based similarity still has some limitations.

Character-Based Similarity Functions. They use characters in the two strings to quantify the similarity, such as edit distance which is the minimum number of single-character edit operations (i.e., insertion, deletion, and substitution) needed to transform one to another. In comparison with token-based similarity, edit distance is sensitive to the positions of the tokens in a string. For example, recall the two strings nba mcgrady and macgrady nba. Their edit distance is 9. Although the two strings are very similar, their edit-distance-based similarity is very low.

These two classes of similarity functions have limitations in evaluating the similarity of two strings. These problems seem trivial but are very serious for many datasets, such as Web query log and person names. To address these problems, we propose a new similarity function, fuzzy-token-matching-based similarity (hereinafter referred to as fuzzy-token similarity), by combining token-based similarity and character-based similarity. Differently from token-based similarity that only considers exact matches between two tokens, we also incorporate character-based similarity of mismatched token pairs into the fuzzy-token similarity. For example, recall the two strings nba mcgrady and macgrady nba. They contain one exactly matched token nba and one approximately matched token pair (mcgrady, macgrady). We consider both of the two cases in the fuzzy-token similarity. We give the formal definition of the fuzzy-token similarity and prove that many well-known similarity functions (e.g., jaccard similarity) are special cases of fuzzy-token similarity (Section 2).

There are several challenges to addressing the similarity-join problem using fuzzy-token similarity. First, fuzzy-token similarity is more complicated than token-based similarity and character-based similarity, and it is rather expensive to compute fuzzy-token similarity of two strings (Section 2.2). Second, for exact matches of token pairs, we can sort the tokens and use prefix filtering to prune large numbers of dissimilar

string pairs [Chaudhuri et al. 2006]. However, as we consider fuzzy match of two tokens, it is nontrivial to sort the tokens and use prefix filtering. Thus it calls for new effective techniques and efficient algorithms. Third, in real-world applications, as different tokens may have different weights, it is essential for our techniques to support weighted tokens. In this article, we propose *f*uzzy-token-m*a*tching-based *s*tring similari*t*y *join* (called Fast-Join) to address these problems. To summarize, we make the following contributions.

—We propose a new similarity function, fuzzy-token similarity, and prove that many existing token-based similarity functions and character-based similarity functions are special cases of fuzzy-token similarity.
—We formulate the similarity-join problem using fuzzy-token similarity. We propose a signature-based framework to address this problem.
—We propose a new signature scheme for token sets and prove it is superior to the state-of-the-art method. We present a new signature scheme for tokens and develop effective pruning techniques to improve the performance.
—We extend fuzzy-token similarity to support weighted tokens and propose novel signature schemes for weighted token sets.
—We have implemented our method on real datasets. The experimental results show that our method achieves high performance and result quality and outperforms state-of-the-art methods.

The rest of this article is organized as follows. Section 2 proposes the fuzzy-token similarity. Section 3 formalizes the similarity-join problem using fuzzy-token similarity and presents a signature-based method. We propose new signature schemes for token sets and tokens, respectively, in Section 4 and Section 5. Section 6 extends our techniques to support weighted tokens. Experimental results are provided in Section 7. We review related works in Section 8 and conclude in Section 9.

2. FUZZY-TOKEN SIMILARITY

We first review existing similarity functions and then formalize the fuzzy-token similarity. Finally we prove that existing similarities are special cases of fuzzy-token similarity.

2.1. Existing Similarity Functions

String similarity functions are used to quantify the similarity between two strings, which can be roughly divided into three groups: token-based similarity, character-based similarity, and hybrid similarity.

Token-Based Similarity. It tokenizes strings into token sets (e.g., using white space) and quantifies the similarity based on the token sets. For example, given a string `nba mcgrady`, its token set is {`nba`, `mcgrady`}. We give some representative token-based similarity: *dice similarity*, *cosine similarity*, and *jaccard similarity*, defined as follows. Given two strings s and s' with token sets T and T'.

—*Dice Similarity.* $\text{DICE}(\mathsf{s}, \mathsf{s}') = \frac{2 \cdot |T \cap T'|}{|T| + |T'|}$.
—*Cosine Similarity.* $\text{COSINE}(\mathsf{s}, \mathsf{s}') = \frac{|T \cap T'|}{\sqrt{|T| \cdot |T'|}}$.
—*Jaccard Similarity.* $\text{JACCARD}(\mathsf{s}, \mathsf{s}') = \frac{|T \cap T'|}{|T| + |T'| - |T \cap T'|}$.

Note that there may exist duplicate tokens in token sets; to avoid multiset intersection, we append each token with an ordinal number to distinguish duplicate tokens [Chaudhuri et al. 2006; Wang et al. 2012].

Token-Based Similarity functions use the overlap of two token sets to quantify the similarity. However, they only consider exactly matched token pairs to compute the overlap and neglect the approximately matched pairs which refer to the same token. For example, consider two strings s_1 = nba trace mcgrady and s_2 = trac macgrady nba. Their token sets share one token nba, and their jaccard similarity is 1/5. Consider another string s_3 = nba trace video. For s_1 and s_3, their token sets share two tokens {nba,trace}, and their jaccard similarity is 2/4. Although $\text{JACCARD}(s_1, s_2) < \text{JACCARD}(s_1, s_3)$, actually s_2 should be much more similar to s_1 than s_3, since all of the three tokens in s_2 are similar to those in s_1.

Character-Based Similarity. It considers characters in strings to quantify the similarity. As an example, edit distance is the minimum number of single-character edit operations (i.e., insertion, deletion, substitution) to transform one into another. For example, the edit distance between macgrady and mcgrady is 1. We normalize the edit distance to interval [0,1] and use *edit similarity* to quantify the similarity of two strings, where edit similarity between two strings s and s′ is $\text{NED}(s, s') = 1 - \frac{\text{ED}(s,s')}{\max(|s|,|s'|)}$ in which $|s|$ ($|s'|$) denotes the length of s (s′).

Note that edit similarity is sensitive to the position information of each token. For example, the edit similarity between strings nba trace mcgrady and trace mcgrady nba is very small, although they are actually very similar.

Hybrid Similarity. Chaudhuri et al. [2003] proposed generalized edit similarity (GES), which extends the character-level edit operator to the token-level edit operator. For example, consider two strings nba mvp mcgrady and mvp macgrady. We can use two token-level edit operators to transform the first one to the second one (e.g., deleting the token nba and substituting mcgrady for macgrady). Note that we can consider the token weight in the transformation. For example, nba is less important than macgrady, and we can assign a lower weight for nba. However the generalized edit similarity is sensitive to token positions.

Chaudhuri et al. [2003] also derived an approximation of generalized edit similarity (AGES). This similarity ignores the positions of tokens and requires each token in one string to match the "closest" token (i.e., the most similar one) in another string. For example, consider two strings s = wnba nba and s′ = nba. For the tokens in s wnba and nba, their "closest" tokens in s′ are both nba. We, respectively, compute the similarity between wnba and nba and that between nba and nba. The AGES between s and s′ is the average value of these two similarity values, that is, $\text{AGES}(s,s') = \frac{0.75+1}{2} = 0.875$. However, as shown in the experiment, AGES did not have good performance, since the closest tokens chosen by AGES may not be real fuzzy-matching tokens.

To address this problem, SoftTFIDF [Cohen et al. 2003] used a threshold to remove the closest tokens with lower similarity. For example, consider the two strings in the preceding example. If we specify a threshold of 0.8, for the token wnba in s, although its closest token in s′ is nba, their similarity will not be considered, since it is smaller than the specified threshold (i.e., $\text{NED}(\text{wnba}, \text{nba}) = 0.75 < 0.8$). With the help of the threshold, SoftTFIDF can achieve much better performance than AGES, but to the best of our knowledge, there does not exist any efficient similarity-join algorithm for SoftTFIDF.

Considering the limitations of existing similarity functions, in the following sections, we propose a new similarity function along with an efficient similarity-join algorithm to support fuzzy-token matching.

2.2. Fuzzy-Token Similarity

We propose a powerful similarity function, fuzzy-token similarity, by combining token-based similarity and character-based similarity. Different from token-based similarity

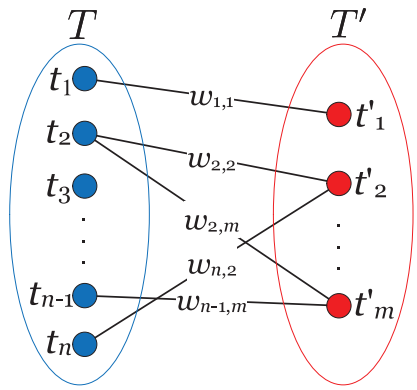


Fig. 1. Weighted bigraph.

which computes the exact overlap of two token sets (i.e., the number of exactly matched token pairs), we compute *fuzzy overlap* in considering fuzzy match between tokens as follows.

Given two token sets, we use character-based similarity to quantify the similarity of token pairs from the two sets. As an example, in this article, we focus on edit similarity, which is a good measurement in capturing typographical errors for text documents [Xiao et al. 2008a]. We first compute the edit similarity of each token pair from the two sets, then use maximum weight matching in bipartite graphs (bigraphs) for computing fuzzy overlap as follows.

We construct a weighted bigraph $G = ((X, Y), E)$ for token sets T and T' as follows, where X and Y are two disjoint sets of vertices, and E is a set of weighted edges that connect a vertex from X to a vertex in Y. In our problem, as illustrated in Figure 1, the vertices in X and Y are, respectively, tokens in T and T', and an edge from a token $t \in T$ to a token $t' \in T'$ is their edit similarity. For example, in the figure, the edge with the weight $w_{1,1}$ means that the edit similarity between t_1 and t'_1 is $w_{1,1}$. Intuitively, if two tokens are not similar, their similarity value does not make any sense, thus the corresponding edge should not be considered. To achieve this goal, we specify an edit-similarity threshold δ and only keep the edges whose weight is no smaller than δ. For example, consider $t = \texttt{boeing}$ and $t' = \texttt{boxing}$. Obviously, they have different meanings, and their similarity should not be considered. But without using δ, we still need to add an edge between t and t' with the weight of $\text{NED}(\texttt{boeing}, \texttt{boxing}) = 0.83$. However, if we specify an edit-similarity threshold $\delta = 0.9$, this issue can be solved, since $\text{NED}(\texttt{wnba}, \texttt{nba}) = 0.83 < 0.9$, and the edge will be removed.

The *maximum weight matching* of G is a set of edges $M \subseteq E$ satisfying the following conditions. (1) Matching: any two edges in M have no a common vertex; (2) maximum: the sum of weights of edges in M is maximal. We use G's maximum weight matching as the fuzzy overlap of T and T', denoted by $T \widetilde{\cap}_\delta T'$. Note that the time complexity for finding maximum weight matching is $\mathcal{O}(|V|^2 * |E|)$ [Bertsekas 1993], where $|V|$ is the number of vertices and $|E|$ is the number of edges in bigraph G. We give an example to show how to compute the fuzzy overlap.

Example 2.1. Consider two strings $\texttt{s} = \texttt{nba mcgrady}$ and $\texttt{s}' = \texttt{macgrady nba}$. We first compute the edit similarity of each token pair: $\text{NED}(\texttt{nba}, \texttt{macgrady}) = \frac{1}{8}$, $\text{NED}(\texttt{nba}, \texttt{nba}) = 1$, $\text{NED}(\texttt{mcgrady}, \texttt{macgrady}) = \frac{7}{8}$, $\text{NED}(\texttt{mcgrady}, \texttt{nba}) = \frac{1}{7}$. For an edit-similarity threshold $\delta = 0.8$, we construct a weighted bigraph with two weighted edges: one edge e_1 with weight 1 for token pair (nba, nba) and the other edge e_2 with

weight $\frac{7}{8}$ for token pair (mcgrady, macgrady). The maximum weight matching of this bigraph is the edge set $\{e_1, e_2\}$ which meets two conditions: matching and maximum. Thus the fuzzy overlap $T \widetilde{\cap}_{0.8} T'$ is $\{e_1, e_2\}$ and its weight is $|T \widetilde{\cap}_{0.8} T'| = \frac{15}{8}$.

Using fuzzy overlap, we define fuzzy-token similarity.

Definition 2.2 (*Fuzzy-Token Similarity*). Given two strings s and s' and an edit-similarity threshold δ, let T and T' be the token sets of s and s', respectively.

—*Fuzzy-Dice Similarity.* $\text{FDICE}_\delta(\mathsf{s}, \mathsf{s}') = \frac{2 \cdot |T \widetilde{\cap}_\delta T'|}{|T|+|T'|}$.

—*Fuzzy-Cosine Similarity.* $\text{FCOSINE}_\delta(\mathsf{s}, \mathsf{s}') = \frac{|T \widetilde{\cap}_\delta T'|}{\sqrt{|T| \cdot |T'|}}$.

—*Fuzzy-Jaccard Similarity.* $\text{FJACCARD}_\delta(\mathsf{s}, \mathsf{s}') = \frac{|T \widetilde{\cap}_\delta T'|}{|T|+|T'|-|T \widetilde{\cap}_\delta T'|}$.

For example, consider s and s' in Example 2.1. Their fuzzy-jaccard is $\text{FJACCARD}_\delta(\mathsf{s}, \mathsf{s}') = \frac{1+7/8}{4-1-7/8} = 15/17$.

Discussion. Fuzzy-token similarity is able to capture two types of errors: (1) *token swap error* means that tokens may match in different positions (e.g., nba mcgrady and mcgrady nba); (2) *edit error* means that tokens may match approximately (e.g., macgrady and mcgrady). Typically, these errors are introduced when the data is acquired manually or from Optical Character Recognition (OCR), and changing the order of tokens has little effect on the meaning of the data. In practice, there are many real application domains satisfying the characteristics, such as Web query log and scanned person names.

In addition, the use of the threshold δ will introduce a bias against errors in short tokens (e.g., the token whose length is smaller than $\frac{\delta}{1-\delta}$ needs to match exactly). This bias is reasonable, since a short token contains fewer characters, potentially resulting in fewer number of edit errors.

2.3. Comparison with Existing Similarities

In this section, we compare fuzzy-token similarity with existing similarities. Existing token-based similarity, such as jaccard similarity, obeys the triangle inequality: however, fuzzy-token similarity does not obey the triangle inequality. We give an example to prove this property. Consider three strings with only one token, $\mathsf{s}_1 = \texttt{abc}$, $\mathsf{s}_2 = \texttt{abcd}$ and $\mathsf{s}_3 = \texttt{bcd}$. We have $\text{NED}(\mathsf{s}_1, \mathsf{s}_2) = \text{NED}(\mathsf{s}_2, \mathsf{s}_3) = 0.75$, and $\text{NED}(\mathsf{s}_1, \mathsf{s}_3) = \frac{1}{3}$. Let edit-similarity threshold $\delta = 0.5$. We have $|\mathsf{s}_1 \widetilde{\cap}_{0.5} \mathsf{s}_2| = |\mathsf{s}_2 \widetilde{\cap}_{0.5} \mathsf{s}_3| = 0.75$ and $|\mathsf{s}_1 \widetilde{\cap}_{0.5} \mathsf{s}_3| = 0$ (as $\frac{1}{3} < 0.5$). Thus $\text{FJACCARD}_\delta(\mathsf{s}_1, \mathsf{s}_2) = \text{FJACCARD}_\delta(\mathsf{s}_2, \mathsf{s}_3) = \frac{0.75}{2-0.75} = 0.6$ and $\text{FJACCARD}_\delta(\mathsf{s}_1, \mathsf{s}_3) = 0$. Usually, one minus the similarity denotes the corresponding distance. We have $(1 - 0.6) + (1 - 0.6) < (1 - 0)$. Thus fuzzy-jaccard does not obey the triangle inequality. Similarly, the example can also show fuzzy-dice and fuzzy-cosine do not obey the triangle inequality. Thus our similarities are not metric-space similarities and cannot use existing studies [Jacox and Samet 2008] to support our similarities.[1]

Next we investigate the relationship between fuzzy-token similarity and existing similarities. We first compare it with token-based similarity. If $\delta = 1$ for fuzzy-token similarity, then a fuzzy overlap will be equal to the overlap (Lemma 2.3), and the corresponding fuzzy-token similarity will turn to token-based similarity. Thus token-based similarity is only a special case of the fuzzy-token similarity when $\delta = 1$.

[1]Our similarities will become metric-space similarities if we do not use δ. But in this case, the precision of our similarities is very low (see the experiment in Section 7.1.1).

LEMMA 2.3. *For token sets T and T', $|T \widetilde{\cap}_1 T'| = |T \cap T'|$.*

PROOF. When $\delta = 1$, consider the weighted bigraph between T and T'. For each edge, it connects two same tokens. Since there's no common vertex among the edges, $|T \widetilde{\cap}_1 T'|$ is the sum of the weights of all the edges. As the weight of each edge is 1, we have $|T \widetilde{\cap}_1 T'|$ is equal to $|T \cap T'|$. □

For the general case ($\delta \in [0, 1]$), a fuzzy overlap never have a smaller value than the corresponding overlap (Lemma 2.4).

LEMMA 2.4. *For token sets T and T', $|T \widetilde{\cap}_\delta T'| \geq |T \cap T'|$.*

PROOF. When $\delta = 1$, consider the weighted bigraph between T and T'. With δ decreasing, it may have additional edges. So, $|T \widetilde{\cap}_\delta T'| \geq |T \widetilde{\cap}_1 T'|$. Applying Lemma 2.3 to the inequality, we have the Lemma 2.4. □

Based on this lemma, we can deduce that fuzzy-token similarity will never have a smaller value than the corresponding token-based similarity. One advantage of this property is that for a string pair, if they are similar evaluated by token-based similarity, then they are still similar for fuzzy-token similarity.

Next we compare fuzzy-token similarity with edit similarity. We find that edit similarity is also a special case of the fuzzy-token similarity, as stated in Lemma 2.5.

LEMMA 2.5. *Given two strings s and s', let token sets $T = \{s\}$ and $T' = \{s'\}$, we have $|T \widetilde{\cap}_{\delta=0} T'| = \text{NED}(s, s')$.*

PROOF. When $\delta = 0$, consider the weighted bigraph between $T = \{\mathsf{s}\}$ and $T' = \{\mathsf{s}'\}$. It has only one edge with the weight $\text{NED}(\mathsf{s}, \mathsf{s}')$. Its maximum weight matching consists of this edge, so we have $|T \widetilde{\cap}_{\delta=0} T'| = \text{NED}(\mathsf{s}, \mathsf{s}')$. □

Based on the preceding analysis, fuzzy-token similarity is a generalization of many existing similarities, such as jaccard similarity and edit similarity, and also has some different properties from existing similarities which pose new challenges when using it to quantify the similarity.

3. STRING SIMILARITY JOIN USING FUZZY-TOKEN SIMILARITY

In this section, we study the similarity-join problem using fuzzy-token similarity to compute similar string pairs.

3.1. Problem Formulation

Let S and S' be two collections of strings, and R and R' be the corresponding collections of token sets. For $T \in R$ and $T' \in R'$, let $\mathcal{F}_\delta(T, T')$ denote the fuzzy-token similarity of T and T', where $\mathcal{F}_\delta$ could be FJACCARD_δ, FCOSINE_δ, and FDICE_δ. We define the similarity-join problem as follows.

Definition 3.1. *(Fuzzy-Token-Matching-Based String Similarity Join).* Given two collections of strings S and S', and a threshold τ, a fuzzy token matching based string similarity join is to find all the pairs $(\mathsf{s}, \mathsf{s}') \in S \times S'$ such that $\mathcal{F}_\delta(T, T') \geq \tau$, where $T(T')$ is the token set of $s(s')$.

A straightforward method to address this problem is to enumerate each pair $(T, T') \in R \times R'$ and compute their fuzzy-token similarity. However this method is rather expensive, and we propose an efficient method, called Fast-Join.

3.2. A Signature-Based Method

We adopt a signature-based method [Sarawagi and Kirpal 2004]. For ease of presentation, we introduce this method using self-join, that is, $R = R'$. First, we generate

signatures for each token set, which have a property that: given two token sets T and T' with signature sets $Sig(T)$ and $Sig(T')$, respectively, T and T' are similar only if $Sig(T) \cap Sig(T') \neq \phi$. Based on this property, we can filter large numbers of dissimilar pairs and obtain a small set of candidate pairs. Finally, we verify the candidate pairs to generate the final results. We call our method as Fast-Join.

Signature Schemes. It is very important to devise a high-quality scheme in this framework, as such a signature could prune large numbers of dissimilar pairs. Section 4 and Section 5 study how to generate high-quality signatures.

The Filter Step. This step generates candidates of similar pairs based on signatures. We use an *inverted index* to generate candidates [Sarawagi and Kirpal 2004] as follows. Each signature in the signature sets has an inverted list of those token sets whose signature sets contain the signature. In this way, two token sets in the same inverted lists are candidates as their signature sets have overlaps. For example, given token sets T_1, T_2, T_3, T_4, with $Sig(T_1) = \{\texttt{ad}, \texttt{ac}, \texttt{dc}\}$, $Sig(T_2) = \{\texttt{be}, \texttt{cf}, \texttt{em}\}$, $Sig(T_3) = \{\texttt{ad}, \texttt{ab}, \texttt{dc}\}$, and $Sig(T_4) = \{\texttt{bm}, \texttt{cf}, \texttt{be}\}$. The inverted list of ad is $\{T_1, T_3\}$. Thus (T_1, T_3) is a candidate. As there is no signature whose inverted list contains both T_1 and T_2, they are dissimilar and can be pruned. To find similar pairs among the four token sets, we generate two candidates (T_1, T_3) and (T_2, T_4) and prune the other four pairs.

We can optimize this framework using the all-pair-based algorithm [Bayardo et al. 2007]. In this article, we focus on how to generate effective signatures and use this framework as an example. Our method can be easily extended to other frameworks.

The Refine Step. This step verifies the candidates to generate the final results. Given two token sets T_1 and T_2, we construct a weighted bigraph as described in Section 2.2. As it is expensive to compute the maximum weight matching, we propose an improved method. We compute an upper bound of the maximal weight by relaxing the *matching* condition, that is, we allow that the edges in M can share a common vertex. We can compute this upper bound by summing up the maximum weight of edges of every token in T (or T'). If this upper bound makes $\mathcal{F}_\delta(T, T')$ smaller than τ, we can prune the pair (T, T'), since $\mathcal{F}_\delta(T, T')$ is no larger than its upper bound and thus will also be smaller than τ.

4. SIGNATURE SCHEME OF TOKEN SETS

In the signature-based method, it is very important to define a high-quality signature scheme, since a better signature scheme can prune many more dissimilar pairs and generate smaller numbers of candidates. In this section, we propose a high-quality signature scheme for token sets.

4.1. Existing Signature Schemes

Let us first review existing signature schemes for exact search, that is, $\delta = 1$. Consider two token sets $T = \{t_1, t_2, \ldots, t_n\}$ and $T' = \{t'_1, t'_2, \ldots, t'_m\}$, where t_i denotes a token in T and t'_j denotes a token in T'. Suppose T and T' are similar if $|T \cap T'| \geq c$, where c is a constant. A simple signature scheme is $Sig(T) = T$ and $Sig(T') = T'$. Obviously if T and T' are similar, their overlap is not empty, that is, $Sig(T)$ and $Sig(T')$ have common signatures. A well-known improved method is to use prefix filtering [Chaudhuri et al. 2006], which selects a subset of tokens as signatures. To use prefix filtering, we first fix a global order on all signatures (i.e., tokens). We then remove the $\lceil c - 1 \rceil$ signatures with largest order from $Sig(T)$ and $Sig(T')$ to obtain the new signature set $Sig_p(T)$ and $Sig_p(T')$. Note that if T and T' are similar, $|Sig_p(T) \cap Sig_p(T')| \neq \phi$ [Chaudhuri et al. 2006].

For example, consider two token sets $T = \{\texttt{nba}, \texttt{kobe}, \texttt{bryant}\}$, $T' = \{\texttt{nba}, \texttt{tracy}, \texttt{mcgrady}\}$ and a threshold $c = 2$. They cannot be filtered by the simple signature scheme, as $Sig(T) = T$ and $Sig(T') = T'$ have overlaps. Using alphabetical order, we can remove nba from $Sig(T)$ and tracy from $Sig(T')$, and get $Sig_p(T) = \{\texttt{bryant}, \texttt{kobe}\}$ and $Sig_p(T') = \{\texttt{nba}, \texttt{mcgrady}\}$. As they have no overlap, we can prune them.

However, it is not straightforward to extend this method to support $\delta \neq 1$, as we consider fuzzy token matching. For example, consider the token sets $\{\texttt{hoston}, \texttt{mcgrady}\}$ and $\{\texttt{houston}, \texttt{macgrady}\}$. Clearly they have large fuzzy overlap but have no overlap. To address this problem, we propose an effective signature scheme for fuzzy overlap.

4.2. Token-Sensitive Signature

As the similarity function $\mathcal{F}_\delta$ is rather complicated and it is hard to devise an effective signature scheme for this similarity, we simplify it and deduce an equation that if $\mathcal{F}_\delta(T, T') \geq \tau$, then there exists a constant c such that $|T \widetilde{\cap}_\delta T'| \geq c$. Then we propose a signature scheme $Sig^\delta(\cdot)$ satisfying if $|T \widetilde{\cap}_\delta T'| \geq c$, then $Sig^\delta(T) \cap Sig^\delta(T') \neq \phi$. We can devise a pruning technique: if $Sig^\delta(T) \cap Sig^\delta(T') = \phi$, we have $|T \widetilde{\cap}_\delta T'| < c$ and $\mathcal{F}_\delta(T, T') < \tau$, thus we can prune (T, T'). Section 4.3 gives how to deduce c for different similarity functions. Here we discuss how to devise effective signature schemes for $|T \widetilde{\cap}_\delta T'| \geq c$.

Signature Scheme for $|T \widetilde{\cap}_\delta T'| \geq c$. Given two token sets $T = \{t_1, t_2, \ldots, t_n\}$ and $T' = \{t'_1, t'_2, \ldots, t'_m\}$, we study how to generate the signature sets $Sig^\delta(T)$ and $Sig^\delta(T')$ for the condition $\delta \neq 1$ such that if $|T \widetilde{\cap}_\delta T'| \geq c$, then $Sig^\delta(T) \cap Sig^\delta(T') \neq \phi$.

Intuitively, $|T \widetilde{\cap}_\delta T'| \geq c$ means that there are at least c similar token pairs between T and T', where two tokens $t_i \in T$ and $t'_j \in T'$ are similar if and only if $\text{NED}(t_i, t'_j) \geq \delta$ holds. To generate $Sig^\delta(T)$ and $Sig^\delta(T')$, we first generate the signatures of tokens t_i and t'_j, denoted as $sig^\delta(t_i)$ and $sig^\delta(t'_j)$, respectively, such that if $\text{NED}(t_i, t'_j) \geq \delta$ holds, $sig^\delta(t_i) \cap sig^\delta(t'_j) \neq \phi$. (We will discuss the signature scheme for tokens in Section 5.) Let $Sig^\delta(T) = \biguplus_{i=1}^n sig^\delta(t_i)$ and $Sig^\delta(T') = \biguplus_{i=1}^m sig^\delta(t'_i)$ be the signatures of T and T' respectively, where $\biguplus$ denotes the union operation for multisets.[2] Then we have if $|T \widetilde{\cap}_\delta T'| \geq c$, there will be at least c token pairs between T and T' whose signature sets have overlap, that is, $|Sig^\delta(T) \cap Sig^\delta(T')| \geq c$. The following Lemma shows the correctness of the signature scheme for token sets.

LEMMA 4.1. *For two token sets* $T = \{t_1, t_2, \ldots, t_n\}$ *and* $T' = \{t'_1, t'_2, \ldots, t'_m\}$, *if* $|T \widetilde{\cap}_\delta T'| \geq c$, *then* $|Sig^\delta(T) \cap Sig^\delta(T')| \geq c$, *where* $Sig^\delta(T) = \biguplus_{i=1}^n sig^\delta(t_i)$ *and* $Sig^\delta(T') = \biguplus_{i=1}^m sig^\delta(t'_j)$.

PROOF. Recall that $T \widetilde{\cap}_\delta T'$ denotes the maximum weight matching of their corresponding weighted bigraph G. Each edge in G for vertices $t_i \in T$ and $t'_j \in T'$ is $\text{NED}(t_i, t'_j) \geq \delta$. We construct another bigraph G' with the same vertices and edges as G except that the edge weights are assigned as follows. Then for each edge of vertices t_i and t'_j in G', we assign its weight to $|sig^\delta(t_i) \cap sig^\delta(t'_j)|$. As there exists an edge in G between t_i and t'_j, we have $\text{NED}(t_i, t'_j) \geq \delta$, thus $sig^\delta(t_i) \cap sig^\delta(t'_j) \neq \phi$. Since $|sig^\delta(t_i) \cap sig^\delta(t'_j)| \geq 1 \geq \text{NED}(t_i, t'_j)$, for any edge in G', its weight is no smaller than that of the corresponding edge in G. Therefore, the maximum matching weight in G' is no smaller than that in G. Without loss of generality, let $M = \{(t_1, t'_1), (t_2, t'_2), \ldots, (t_k, t'_k)\}$ be the maximum weight matching of G', where each element (t_i, t'_i) in M denotes an edge

[2]In this article, we use multiset which is a generalization of a set. A multiset can have more than one membership, that is, there may be multiple instances of a member in a multiset.

of G' with the edge weight of $|sig^{\delta}(t_i) \cap sig^{\delta}(t'_i)|$. Thus the maximal matching weight of G' is $\sum_{i=1}^{k} |sig^{\delta}(t_i) \cap sig^{\delta}(t'_i)|$. Based on the definition of matching, no two edges in M have a common vertex. Hence,

$$\begin{aligned}\sum_{i=1}^{k} |sig^{\delta}(t_i) \cap sig^{\delta}(t'_i)| &\leq \left| \left(\biguplus_{i=1}^{k} sig^{\delta}(t_i) \right) \cap \left(\biguplus_{i=1}^{k} sig^{\delta}(t'_i) \right) \right| \\ &\leq \left| \left(\biguplus_{i=1}^{n} sig^{\delta}(t_i) \right) \cap \left(\biguplus_{i=1}^{m} sig^{\delta}(t'_i) \right) \right|.\end{aligned}$$

Since the maximal matching weight in G' is no smaller than that in G, then we have

$$\sum_{i=1}^{k} |sig^{\delta}(t_i) \cap sig^{\delta}(t'_i)| \geq |T \widetilde{\cap}_{\delta} T'|.$$

Based on these two equations, we can deduce that

$$|Sig^{\delta}(T) \cap Sig^{\delta}(T')| = \left| \left(\biguplus_{i=1}^{n} sig^{\delta}(t_i) \right) \cap \left(\biguplus_{i=1}^{m} sig^{\delta}(t'_i) \right) \right| \geq |T \widetilde{\cap}_{\delta} T'| \geq c.$$

Therefore, the lemma is proved. □

Obviously we can use prefix filtering to improve this signature scheme. We fix a global order and then generate $Sig_p^{\delta}(T)$ from $Sig^{\delta}(T)$ by removing the last $\lceil c-1 \rceil$ signatures with largest order. Example 4.2 gives an example.

Example 4.2. Consider the collection of token sets $\mathcal{R}$ in Figure 2. Given $\delta = 0.8$ and $c = 2.4$, we aim to generate a signature set for each token set in $\mathcal{R}$ such that if two token sets are similar (i.e., $|T_i \widetilde{\cap}_{0.8} T_j| \geq 2.4$), then their corresponding signature sets have overlaps (i.e., $Sig^{\delta}(T_i) \cap Sig^{\delta}(T_j) \neq \phi$). We assume the global order is the alphabetical order in this example.

At the first step, as shown in "Token Signatures", we collect all the tokens in $\mathcal{R}$ and generate a signature set for each token. Here we choose some q-grams (substrings of the token that consists of q consecutive characters) as token's signatures [Xiao et al. 2008a], which will be explained in Section 5. For instance, the signature set of `macgrady` is {`ac`, `cg`, `ad`}. We find that if two tokens are similar (e.g., NED(`macgrady`, `mcgrady`) $\geq$ 0.8), they at least share one signature (e.g., `ad`).

At the second step, we generate signatures $Sig^{\delta}(T_i)$ as the union of its tokens' signatures. For example, consider the token set T_2 = {`trcy`, `macgrady`, `mvp`}, we have $Sig^{\delta}(T_2) = \{\mathtt{tr}^1, \mathtt{rc}^1, \mathtt{cy}^1, \mathtt{ac}^2, \mathtt{cg}^2, \mathtt{ad}^2, \mathtt{mv}^3\}$. Each signature has a superscript that denotes which token generates this signature. For instance, $\mathtt{ac}^2$ denotes that the signature `ac` is generated from the second token `macgrady`. Note that $Sig^{\delta}(T_i)$ is a multiset. For example, $Sig^{\delta}(T_1)$ contains two `an` from the second and the third tokens, respectively.

At the third step, to generate signatures using prefix filtering, we delete $\lceil c-1 \rceil = 2$ largest signatures from $Sig^{\delta}(T_j)$ and generate $Sig_p^{\delta}(T_j)$. For instance, we can get $Sig_p^{\delta}(T_2)$ by removing `rc` and `tr` from $Sig^{\delta}(T_2)$ since they are the two largest signatures based on alphabetical order. Using this signature scheme, $Sig_p^{\delta}(T_2)$ have no overlap with $Sig_p^{\delta}(T_3)$, so we can filter (T_2, T_3). For other token-set pairs, such as (T_2, T_1), because $Sig_p^{\delta}(T_2)$ and $Sig_p^{\delta}(T_1)$ have a common signature (i.e., `cy`), they will be considered as the candidate pair for further verification.

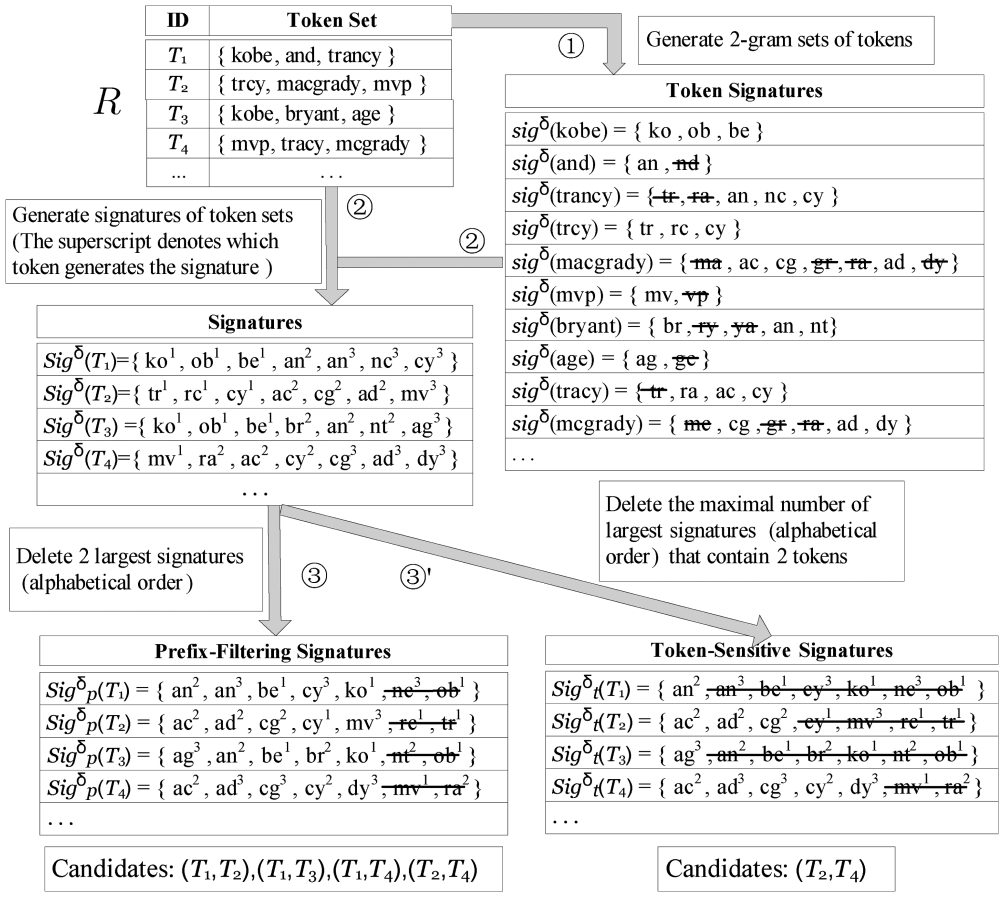


Fig. 2. Prefix-filtering signatures and token-sensitive signatures of the token sets in $\mathcal{R}$ ($\delta = 0.8$, $c = 2.4$).

Token-Sensitive Signature. We propose a novel signature scheme that can remove many more signatures than prefix filtering. As an example, consider the token sets T_1 and T_3 in Figure 2. $Sig^{\delta}(T_1)$ and $Sig^{\delta}(T_3)$ have a large overlap {an, be, ko, ob}. Thus based on prefix filtering, when $c = 2.4$ they will not be filtered. Here we have an observation that these signatures are only generated from two tokens. For example, the overlap $\{\mathtt{an}^2, \mathtt{be}^1, \mathtt{ko}^1, \mathtt{ob}^1\}$ in T_3 is generated from two tokens kobe and bryant. That is, T_3 at most has two similar tokens with T_1. However, if $|T_1 \widetilde{\cap}_{0.8} T_3| \geq 2.4$, T_3 has at least $\lceil c \rceil = 3$ tokens similar to T_1. Therefore, T_1 and T_3 should be filtered. Based on this observation, we devise a new signature scheme called token-sensitive signature scheme. Given a token set T, we generate its token-sensitive signature set $Sig_t^{\delta}(T)$ as follows. Different from a prefix filtering signature scheme which removes the last $\lceil c - 1 \rceil$ signatures, the token-sensitive signature scheme removes the maximal number of largest signatures (in the global order on signatures) that are generated from at most $\lceil c - 1 \rceil$ distinct tokens. That is, if we remove one more signatures, then the removed signatures are generated from $\lceil c \rceil$ tokens. The following lemma shows the correctness of the token-sensitive scheme.

LEMMA 4.3. *Given two token sets T and T' and a threshold c, if $Sig_t^{\delta}(T) \cap Sig_t^{\delta}(T') = \phi$, the token pair (T, T') can be filtered, that is, $|T \widetilde{\cap}_{\delta} T'| < c$.*

PROOF. Let Ω be a set of tokens in T whose signatures have been removed when generating $Sig_t^\delta(T)$. Based on the definition of $Sig_t^\delta(T)$, we have $|\Omega| = \lceil c-1 \rceil$. Let Δ be the rest of the tokens in T, that is, $\Delta = T - \Omega$. We prove it by contradiction. Assume the lemma does not hold. That is, there exists T and T' such that if $Sig_t^\delta(T) \cap Sig_t^\delta(T') = \phi$, then $|T \widetilde{\cap}_\delta T'| \geq c$.

As there are $\lceil c-1 \rceil$ tokens in Ω, the size of the fuzzy overlap between Ω and T' is

$$|\Omega \widetilde{\cap}_\delta T'| \leq \lceil c-1 \rceil < c. \tag{1}$$

And since $T = \Omega + \Delta$, the size of the fuzzy overlap between T and T' is no larger than the size of the fuzzy overlap between Ω and T' plus the size of the fuzzy overlap between Δ and T', that is,

$$|T \widetilde{\cap}_\delta T'| \leq |\Omega \widetilde{\cap}_\delta T'| + |\Delta \widetilde{\cap}_\delta T'|. \tag{2}$$

As $|T \widetilde{\cap}_\delta T'| \geq c$, according to Equations (1) and (2), we have

$$|\Delta \widetilde{\cap}_\delta T'| \geq 1. \tag{3}$$

Similar to Equation (2), as $T' = \Omega' + \Delta'$, we have

$$|\Delta \widetilde{\cap}_\delta T'| \leq |\Delta \widetilde{\cap}_\delta \Omega'| + |\Delta \widetilde{\cap}_\delta \Delta'|. \tag{4}$$

Based on the given condition $Sig_t^\delta(T) \cap Sig_t^\delta(T') = \phi$, there is no similar token pairs between Δ and Δ', that is, $\Delta \cap \Delta' = \phi$, thus according to Equations (3) and (4), we have

$$|\Delta \widetilde{\cap}_\delta \Omega'| \geq 1. \tag{5}$$

Similarly, we can also deduce

$$|\Omega \widetilde{\cap}_\delta \Delta'| \geq 1. \tag{6}$$

Next we prove that Equation (5) and Equation (6) cannot hold simultaneously.

From Equation (5), we can derive that there exist two tokens $t \in \Delta_1$, and $t' \in \Omega'$ such that $sig^\delta(t) \cap sig^\delta(t') \neq \phi$. Consider a signature $s_1 \in sig^\delta(t) \cap sig^\delta(t')$. Based on the definition of Δ', for any token $t \in \Delta_1$, we can easily deduce $sig^\delta(t) \in Sig_t^\delta(T)$, thus

$$s_1 \in Sig_t^\delta(T). \tag{7}$$

Since $Sig_t^\delta(T) \cap Sig_t^\delta(T') = \phi$, we can derive from Equation (7) that $s_1 \notin Sig_t^\delta(T')$. As $s_1 \in sig^\delta(t') \subseteq Sig^\delta(T')$, we have

$$s_1 \in Sig^\delta(T') - Sig_t^\delta(T'). \tag{8}$$

Similarly, we can also deduce from Equation (6) that there exists a signature s_2 such that

$$s_2 \in Sig_t^\delta(T'), \tag{9}$$

$$s_2 \in Sig^\delta(T) - Sig_t^\delta(T). \tag{10}$$

Since the signatures in $Sig^\delta(T)$ ($Sig^\delta(T')$) are sorted according to a global order, for any signature in $Sig_t^\delta(T)$ ($Sig_t^\delta(T')$), it must rank before all the signatures in $Sig^\delta(T) - Sig_t^\delta(T)$ ($Sig^\delta(T') - Sig_t^\delta(T')$). On one hand, based on Equations (7) and (10), s_1 ranks before s_2 in the global order. However, on the other hand, based on Equations (8) and (9), we can also deduce that s_2 ranks before s_1 in the same global order. Hence, $s_1 = s_2$. But this contradicts with $Sig_t^\delta(T) \cap Sig_t^\delta(T') = \phi$. Therefore, the assumption does not hold, and the lemma is proved. □

We give the pseudocode of the token-sensitive signature scheme in Figure 3. First, $Sig_t^\delta(T)$ is initialized as the union of the signature sets of T's tokens. Then we scan

ALGORITHM 1: TokenSensitiveSignature(T, c)

Input: T is a token set
c is a fuzzy-overlap threshold
Output: $Sig_t^\delta(T)$ is the token-sensitive signature set of T

```
1  begin
2      Sig_t^δ(T) = ⊎_{t∈T} sig^δ(t);
3      Let H be a hash table storing token ids;
4      for each s^tid ∈ Sig_t^δ(T) in decreasing global order on signatures do
5          if tid ∉ H then
6              Add tid into H;
7              if H.size() ≥ c then
8                  break;
9          Remove s^tid from Sig_t^δ(T);
10     return Sig_t^δ(T);
```

Fig. 3. Algorithm of generating token-sensitive signatures for a token set.

the signatures in $Sig_t^\delta(T)$ based on the predefined global order decreasingly. For each signature s^{tid}, we check whether the token tid has occurred before. We use a hash table $\mathcal{H}$ to store the occurred tokens. If tid has occurred (i.e., $tid \in \mathcal{H}$), we remove s^{tid} from $Sig_t^\delta(T)$. If tid has not occurred (i.e., $tid \notin \mathcal{H}$), we add tid into $\mathcal{H}$ and if $\mathcal{H}$.size() $\geq c$, we stop scanning the following signatures and return the signature set $Sig_t^\delta(T)$; otherwise, we remove s^{tid} from $Sig_t^\delta(T)$ and scan the next signature. Example 4.4 shows how this algorithm works.

Example 4.4. Consider the token set T_1 in Figure 2. Given $\delta = 0.8$ and $c = 2.4$, we first initialize $Sig^\delta(T_1) = \{\texttt{an}^2, \texttt{an}^3, \texttt{be}^1, \texttt{cy}^3, \texttt{ko}^1, \texttt{nc}^3, \texttt{ob}^1\}$ with signatures sorted in alphabetical order. We scan the signatures in $Sig^\delta(T_1)$ from back to front. Initially, $\mathcal{H} = \{\}$. For the last signature $\texttt{ob}^1$, it comes from the first token kobe in T_1, since $1 \notin \mathcal{H}$, we add 1 into $\mathcal{H}$. As the size of $\mathcal{H} = \{1\}$ is smaller than 2.4, we remove $\texttt{ob}^1$ from $Sig^\delta(T_1)$ and scan the next signature $\texttt{nc}^3$. Since $\texttt{nc}^3$ comes from the third token and $3 \notin \mathcal{H}$, we add 3 into $\mathcal{H}$. As the size of $\mathcal{H} = \{1, 3\}$ is smaller than 2.4, we remove $\texttt{nc}^1$ from $Sig^\delta(T_1)$. Note that the prefix filtering signature scheme will stop here, but the token-sensitive signature scheme will scan the next signature $\texttt{ko}^1$. Since $\texttt{ko}^1$ comes from the first token and $1 \in \mathcal{H}$, we can directly remove $\texttt{ko}^1$ from $Sig^\delta(T_1)$ and scan the following signatures. We can also remove $\texttt{cy}^3$, $\texttt{be}^1$, $\texttt{an}^3$ as they come from the first or the third tokens which have already been added into $\mathcal{H}$. Finally, we stop at the signature $\texttt{an}^2$. Since $\texttt{an}^2$ comes from the second token and $2 \notin \mathcal{H}$, we add 2 into $\mathcal{H}$. As the size of $\mathcal{H} = \{1, 2, 3\}$ is no smaller than 2.4, we stop removing signatures and return the final signature set $Sig_t^\delta(T_1) = \{\texttt{an}^2\}$.

Figure 2 shows the token-sensitive signatures of the token sets in $\mathcal{R}$. Compared with prefix-filtering signature scheme, it significantly reduces the size of a signature set and filters more token-set pairs. In Example 4.2, a prefix-filtering signature scheme can only prune (T_2, T_3) and (T_3, T_4), but since $Sig_t^\delta(T_1) \cap Sig_t^\delta(T_2) = \phi$ and $Sig_t^\delta(T_1) \cap Sig_t^\delta(T_3) = \phi$ and $Sig_t^\delta(T_1) \cap Sig_t^\delta(T_4) = \phi$, a token-sensitive signature scheme can further filter the token-set pairs (T_1, T_2) and (T_1, T_3) and (T_1, T_4).

Lemma 4.5 proves that the token-sensitive signature scheme is superior to the prefix-filtering signature scheme, since for any token set, it generates no more signatures than the prefix-filtering signature scheme.

LEMMA 4.5. *Given the same global order and the same signature scheme for tokens, for any token set T, the token-sensitive signature scheme generates no more signatures than the prefix filtering signature scheme, that is, $Sig_t^\delta(T) \subseteq Sig_p^\delta(T)$.*

PROOF. Consider the union of signatures of the tokens in T, that is, $Sig^\delta(T)$. If the last $\lceil c \rceil$ signatures in $Sig^\delta(T)$ come from $\lceil c \rceil$ different tokens, then both of signature schemes will remove the last $\lceil c-1 \rceil$ signatures; otherwise, the token-sensitive signature scheme will continue to remove signatures, but the prefix-filtering signature scheme only removes the last $\lceil c-1 \rceil$ signatures. □

4.3. Deducing Constant *c*

In this section, we introduce how to compute the constant c for fuzzy-token similarity, that is, if $\mathcal{F}_\delta(T, T') \geq \tau$, then there exists a constant c such that $|T \widetilde{\cap}_\delta T'| \geq c$. We utilize similar ideas for existing token-based similarity to deduce such constants [Bayardo et al. 2007].

Fuzzy-Dice Similarity.

$$\begin{aligned} \frac{2 \cdot |T \widetilde{\cap}_\delta T'|}{|T| + |T'|} \geq \tau \quad &\Longrightarrow \quad \frac{2 \cdot |T \widetilde{\cap}_\delta T'|}{|T| + |T \widetilde{\cap}_\delta T'|} \geq \tau \\ &\Longrightarrow \quad |T \widetilde{\cap}_\delta T'| \geq \frac{\tau}{2-\tau} \cdot |T|. \end{aligned}$$

Fuzzy-Cosine Similarity.

$$\begin{aligned} \frac{|T \widetilde{\cap}_\delta T'|}{\sqrt{|T| \cdot |T'|}} \geq \tau \quad &\Longrightarrow \quad \frac{|T \widetilde{\cap}_\delta T'|}{\sqrt{|T| \cdot |T \widetilde{\cap}_\delta T'|}} \geq \tau \\ &\Longrightarrow \quad |T \widetilde{\cap}_\delta T'| \geq \tau^2 |T|. \end{aligned}$$

Fuzzy-Jaccard Similarity.

$$\begin{aligned} \frac{|T \widetilde{\cap}_\delta T'|}{|T| + |T'| - |T \widetilde{\cap}_\delta T'|} \geq \tau \quad &\Longrightarrow \quad \frac{|T \widetilde{\cap}_\delta T'|}{|T| - |T \widetilde{\cap}_\delta T'| + |T \widetilde{\cap}_\delta T'|} \geq \tau \\ &\Longrightarrow \quad |T \widetilde{\cap}_\delta T'| \geq \tau \cdot |T|. \end{aligned}$$

Thus given a token set T, we can deduce that $c = \frac{\tau}{2-\tau} \cdot |T|$ for fuzzy-dice, $c = \tau^2 |T|$ for fuzzy-cosine, and $c = \tau \cdot |T|$ for fuzzy-jaccard.

We can prove that if $\mathcal{F}_\delta(T, T') \geq \tau$, then $Sig_t^\delta(T) \cap Sig_t^\delta(T') \neq \phi$. We only show the proof of fuzzy-jaccard. fuzzy-dice and fuzzy-cosine can be proved similarly. If $\text{FJACCARD}_\delta(T, T') \geq \tau$, then $|T \widetilde{\cap}_\delta T'| \geq \max(c_1, c_2)$, where $c_1 = \tau \cdot |T|$ and $c_2 = \tau \cdot |T'|$. Let $Sig_t^\delta(T)$ and $Sig_t^\delta(T)'$ be the signature set of T when the fuzzy-overlap threshold is c_1 and $\max(c_1, c_2)$, respectively. Let $Sig_t^\delta(T')$ and $Sig_t^\delta(T')'$ be the signature set of T' when the fuzzy-overlap threshold is c_2 and $\max(c_1, c_2)$, respectively. As $|T \widetilde{\cap}_\delta T'| \geq \max(c_1, c_2)$, $Sig_t^\delta(T)' \cap Sig_t^\delta(T')' \neq \phi$. As $\max(c_1, c_2)$ is no smaller than c_1 and c_2, $Sig_t^\delta(T)' \subseteq Sig_t^\delta(T)$ and $Sig_t^\delta(T')' \subseteq Sig_t^\delta(T')$, thus $Sig_t^\delta(T) \cap Sig_t^\delta(T') \neq \phi$.

5. SIGNATURE SCHEMES FOR TOKENS

As we need to use the signatures of tokens for generating the signatures of token sets, in this section, we study effective signature schemes for tokens.

5.1. Extending Existing Signature Schemes to Support Edit Similarity

Many signature schemes [Gravano et al. 2001; Xiao et al. 2008a; Bocek et al. 2007; Arasu et al. 2006; Wang et al. 2009] are proposed for evaluating edit distance. They

generate signature sets for tokens t and t' such that if $\text{ED}(t, t')$ is no larger than an edit-distance threshold λ, then their signature sets have overlaps. But for edit similarity, tokens with different lengths might have different edit-distance thresholds. In order to use existing signature schemes, given an edit-similarity threshold δ, for a token t we can compute its maximal edit-distance threshold λ such that for any token t' if $\text{NED}(t, t') \geq \delta$, then $\text{ED}(t, t') \leq \lambda$. As $\text{NED}(t, t') = 1 - \frac{\text{ED}(t,t')}{\max(|t|,|t'|)} \geq \delta$, we have $1 - \frac{\text{ED}(t,t')}{|t|+\text{ED}(t,t')} \geq \delta$, that is $\text{ED}(t, t') \leq \frac{1-\delta}{\delta} \cdot |t|$. Thus we can set $\lambda = \frac{1-\delta}{\delta} \cdot |t|$. For example, consider the token `tracy` and $\delta = 0.8$. For any token t' such that $\text{NED}(\texttt{tracy}, t') \geq 0.8$, the edit distance between t' and `tracy` is no larger than $\frac{1-0.8}{0.8} \cdot |5| = 1.25$. Next we review existing signature schemes for tokens. Note that they are designed for edit distance instead of edit similarity; we extend them to support edit similarity. Indeed, the techniques can be easily applied to solve edit-similarity join problem.

q-Gram-Based. signature scheme [Gravano et al. 2001; Xiao et al. 2008a] utilizes the idea that if two tokens are similar, they will have enough common q-grams, where a q-gram is a substring with length q. To extend q-gram-based signature scheme to support edit similarity, for a token t, we compute its maximal edit-distance threshold $\lambda = \frac{1-\delta}{\delta} \cdot |t|$ based on the given edit-similarity threshold δ. We generate t's signature set using the edit-distance threshold λ. However the q-gram-based signature scheme is ineffective for short tokens as it will result in a large number of candidates which need to be further verified.

Deletion-Based Neighborhood Generation [Bocek et al. 2007]. We can use the same idea as the q-gram-based signature scheme to extend the deletion-based neighborhood generation to support edit similarity. However this scheme will generate a large number of signatures for long tokens, even for a large edit-similarity threshold.

Part-Enum [Arasu et al. 2006]. Uses the pigeon-hole principle to generate signatures. For a token t, it first obtains the q-gram set represented as a feature vector. For two tokens, if their edit distance is within λ, then the hamming distance between their feature vectors is no larger than $q \cdot \lambda$. Based on this property, to generate the signatures of the token t with the edit-distance threshold λ, Part-Enum only needs to generate the signatures of the feature vector of t with the hamming-distance threshold $q \cdot \lambda$. It divides the feature vector into $\lceil \frac{q \cdot \lambda + 1}{2} \rceil$ partitions, and based on the pigeon-hole principle, there exists at least one partition whose hamming distance is no larger than 1. For each partition, it further divides the partition into multiple subpartitions. All of the subpartitions compose the signatures of t. To extend Part-Enum to support edit similarity, we cannot simply generate signatures with the maximal edit-distance threshold. This is because the edit distance will affect the number of partitions. For example, given the edit-similarity threshold $\delta = 0.8$ and $q = 1$, for `macgrady`, the maximal edit-distance threshold $\lambda = \frac{1-0.8}{0.8} \cdot |8| = 2$, and Part-Enum needs to divide its feature vector into $\lceil \frac{1 \cdot 2+1}{2} \rceil = 2$ partitions. But for `mcgrady`, the maximal edit-distance threshold $\lambda = \frac{1-0.8}{0.8} \cdot |7| = 1$, and Part-Enum needs to divide its feature vector into $\lceil \frac{1 \cdot 1+1}{2} \rceil = 1$ partition. Although $\text{NED}(\texttt{mcgrady},\texttt{macgrady}) \geq 0.8$, their signature sets have no overlap. To solve this problem, for a token t we compute the minimum length $\delta \cdot |t|$ of a token t' such that $\text{NED}(t, t') \geq \delta$. When generating the signatures for t, we consider the maximal edit-distance threshold $\lfloor \frac{1-\delta}{\delta} \cdot l \rfloor$ for each possible length l of the token t', that is, $l \in [\delta \cdot |t|, |t|]$. For example, consider the token `macgrady`. The length range is $[0.8 \cdot 8, 8]$. Two lengths 7 and 8 satisfy this range. For them, we, respectively, compute the maximal edit-distance thresholds for $l = 7$, $\lfloor \frac{1-0.8}{0.8} \cdot |7| \rfloor = 1$ and for $l = 8$, $\lfloor \frac{1-0.8}{0.8} \cdot |8| \rfloor = 2$. The signature set of `macgrady` for $\delta = 0.8$ is the union of its signature set with the edit-distance thresholds

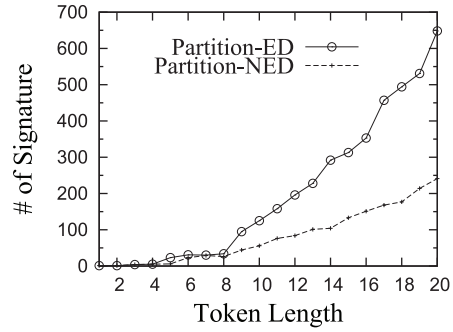


Fig. 4. Comparison of the number of signatures between Partition-ED and Partition-NED for different length of tokens ($\delta = 0.75$).

1 and 2. However Part-Enum needs to tune many parameters to generate signatures, and it generates larger numbers of candidates as it ignores the position information.

Partition-ED [Wang et al. 2009]. A partition-based signature scheme for solving the approximate-entity-extraction problem. It also uses the pigeon-hole principle to generate signatures. Different from Part-Enum, it directly partitions a token instead of the feature vector of a token. Each token t will generate two signature sets, one is called query signature set $sig_q^\delta(t)$ and the other is called data signature set $sig_d^\delta(t)$. For two tokens t and t', if $\text{ED}(t, t') \leq \lambda$, then $sig_q^\delta(t) \cap sig_d^\delta(t') \neq \phi$. Given an edit-distance threshold λ, to obtain $sig_q^\delta(t)$ it divides t into $\lceil \frac{\lambda+1}{2} \rceil$ partitions, and based on the pigeon-hole principle, there exists at least one partition whose edit distance is no larger than 1. It adds 0- and 1-deletion neighborhoods of each partition into $sig_q^\delta(t)$ [Bocek et al. 2007]. To obtain $sig_d^\delta(t)$, it still divides t into $\lceil \frac{\lambda+1}{2} \rceil$ partitions. But for each partition, it also needs to shift and scale it to generate more partitions [Wang et al. 2009]. For all generated partitions, it adds their 0- and 1-deletion neighborhoods into $sig_d^\delta(t)$.

To extend Partition-ED to support edit similarity, for the query signature set $sig_q^\delta(t)$, we only need to generate $sig_q^\delta(t)$ with the edit-distance threshold $\frac{1-\delta}{\delta} \cdot |t|$. For the data signature set, as the same reason as Part-Enum, since the edit distance can affect the number of partitions, we compute the minimum length $\delta \cdot |t|$ and the maximum length $\frac{|t|}{\delta}$ of a token t' such that $\text{NED}(t, t') \geq \delta$. We generate $sig_q^\delta(t)$ with the edit-distance threshold $\lfloor \frac{1-\delta}{\delta} \cdot l \rfloor$ for each possible length l of t', that is, $l \in [\delta \cdot |t|, |t|]$. For example, consider the token `macgrady` and $\delta = 0.8$. The length range is $[0.8 \cdot 8, \frac{8}{0.8}]$. Four lengths 7,8,9,10 satisfy this range. We generate $sig_d^\delta(t)$ with the edit-distance thresholds $\lfloor \frac{1-0.8}{0.8} \cdot |7| \rfloor = 1$, $\lfloor \frac{1-0.8}{0.8} \cdot |8| \rfloor = 2$, $\lfloor \frac{1-0.8}{0.8} \cdot |9| \rfloor = 2$ and $\lfloor \frac{1-0.8}{0.8} \cdot |10| \rfloor = 2$. However, Partition-ED will generate many redundant signatures. For example, for the strings with lengths 9 as their edit-distance threshold with `macgrady` should be no larger than $(1-\delta) * \max(9, |\texttt{macgrady}|) = 1.8$, thus we do not need to generate signatures with the edit-distance threshold 2. Similarly, for strings with lengths 7 and 8, we only need to generate signatures with the edit-distance threshold 1. To address this problem, we propose a new signature scheme Partition-NED in Section 5.2. Figure 4 compares the number of signatures generated by Partition-ED and Partition-NED for different lengths of tokens ($\delta = 0.75$). We can see when the length of token is larger than 8, Partition-ED will generate many more signatures than Partition-NED. For example, Partition-ED generates 125 signatures for the tokens whose length is 10, and Partition-NED only generates 56 signatures.

Experimental result in Section 7 shows our algorithm achieves the best performance when using the Partition-NED signature scheme for generating signatures of tokens.

5.2. Partition-NED Signature Scheme

As discussed in Section 5.1, when extending existing signature schemes to support edit similarity, they have some limitations. To address these limitations, in this section we propose a new signature scheme for edit similarity called Partition-NED.

Overview of Partition-NED. For each token, we generate the same query signature set $sig_q^\delta(t)$ as Partition-ED. Next we discuss how to generate the data signature set $sig_d^\delta(t)$ such that for any token t', if t' is similar to t within δ (i.e., $\text{NED}(t, t') \geq \delta$), then $sig_d^\delta(t) \cap sig_q^\delta(t') \neq \phi$ holds. We first compute the possible lengths of the tokens that can be similar to t, that is, $[\delta \cdot |t|, \frac{|t|}{\delta}]$. Then, we enumerate each possible length and add the corresponding signatures into $sig_d^\delta(t)$. Consider one of possible lengths, $n \in [\delta \cdot |t|, \frac{|t|}{\delta}]$. When generating query signature sets, all the tokens with the length n are divided into $d = \lceil \frac{\lambda+1}{2} \rceil$ partitions, where $\lambda = \lfloor \frac{1-\delta}{\delta} \cdot n \rfloor$ is the maximal edit-distance threshold. For any token (whose length is n) that is similar to t, based on the pigeon-hole principle, there exists at least one partition whose edit distance with a substring of t is within 1. If we can find the corresponding substrings in t for each partition, we only need to add 0- and 1-deletion neighborhoods of them into $sig_d^\delta(t)$.

For example, consider a token $t = c_1c_2 \ldots c_9$. Given $\delta = 0.75$, to generate t's data signature set, we first compute the possible lengths of the tokens that can be similar to t, that is, $[0.75 \cdot 9 = 6.75, \frac{9}{0.75} = 12]$. There are six possible lengths 7,8,9,10,11,12 satisfying the range. For each possible length (e.g., $n = 12$), we compute the maximal edit-distance threshold $\lambda = \lfloor \frac{1-0.75}{0.75} \cdot 12 \rfloor = 4$, and get $d = \lceil \frac{4+1}{2} \rceil = 3$ partitions. Since $\lambda = 4$ and $d = 3$, based on the pigeon-hole principle, for any token (whose length is 12) that is similar to t, there at least exists one partition whose edit distance with a substring of t is within 1. Therefore, the problem is how to find such substrings of t. In the following, we give the algorithm to solve this problem and propose two effective pruning techniques for reducing the number of substrings.

Algorithm Description. Consider a token $t = c_1c_2 \ldots c_m$. To find the corresponding substrings of t with respect to the length n, let t' be an *arbitrary* token whose length is n, that is, $t' = c'_1c'_2 \ldots c'_n$. Note that we introduce the notation t' only for ease of presentation. Suppose t' is divided into d partitions: $t'[1 : \ell] = c_1 \ldots c_\ell; t'[\ell+1 : 2\ell+1] = c_{\ell+1} \ldots c_{2*\ell}; \ldots; t'[(d-1) * \ell + 1 : n] = c_{(d-1)*\ell+1} \ldots c_n$, where $\ell = \lfloor \frac{n}{d} \rfloor$. For example, in Figure 5, the token t' is divided into $d = 3$ partitions $t'[1 : 4]$, $t'[5 : 8]$ and $t'[9 : 12]$, where $\ell = \lfloor \frac{12}{3} \rfloor = 4$. Let $t[p_i : q_i] = c_{p_i}c_{p_i+1} \ldots c_{q_i}$ denote the ith partition of t. Let $\lambda = (1 - \delta) \cdot max(|t|, |t'|)$ be the edit-distance threshold between t and t'. For example, in Figure 5, if $\text{NED}(t, t') \geq 0.75$, then $\text{ED}(t, t') \leq (1 - 0.75) \cdot max(9, 12) = 3$, thus the edit-distance threshold is $\lambda = 3$. For the partitions of t', we consider three cases to find corresponding substrings in t.

Case 1 - The First Partition. Suppose the first partition $t'[p_1 = 1 : q_1]$ has zero or one edit error. For the case that $t'[1 : q_1]$ has zero edit error, $t'[1 : q_1]$ is exactly the same as the substring $t[1 : q_1]$ of t. For the other case that $t'[1 : q_1]$ has one edit error, we, respectively, consider three operations, that is, replacement, deletion, or insertion. If the edit operation is replacement, $t'[1 : q_1]$ can be transformed to $t[1 : q_1]$ by one replacement operation; if the edit operation is deletion, $t'[1 : q_1]$ can be transformed to $t[1 : q_1 - 1]$ by one deletion operation; if the edit operation is insertion, $t'[1 : q_1]$ can be transformed to $t[1 : q_1 + 1]$ by one insertion operation. Therefore, for the first partition

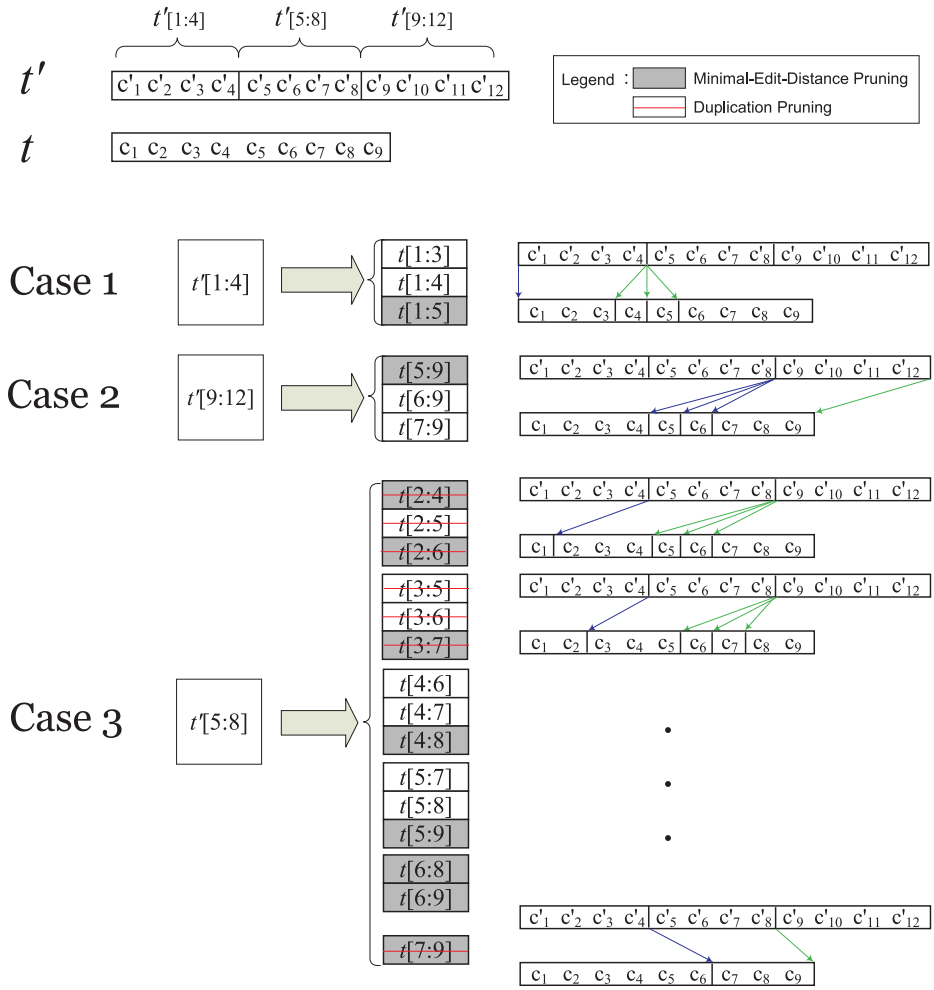


Fig. 5. For the partitions of t', we find eight corresponding substrings t[1:3], t[1:4], t[6:9], t[7:9], t[4:6], t[4:7], t[5:7] and t[5:8] of t ($\delta = 0.75$).

$t'[p_1 = 1 : q_1]$, we select the substrings $t[1 : q_1 - 1]$, $t[1 : q_1]$, and $t[1 : q_1 + 1]$. For example, in Figure 5, the first partition is $t'[1 : 4]$, thus we select the corresponding substrings $t[1 : 3]$, $t[1 : 4]$ and $t[1 : 5]$ from t.

Case 2 - The Last Partition. Suppose the last partition $t'[p_d : n]$ has one or zero edit errors. For the case that $t'[p_d : n]$ has zero edit error, $t'[p_d : n]$ is exactly the same as the substring $t[m - n + p_d : m]$ of t. For the other case that $t'[p_d : n]$ has one edit error, we, respectively, consider three operations, that is, replacement, deletion, or insertion. If the edit operation is replacement, $t'[p_d : n]$ can be transformed to $t[m-n+p_d : m]$ by one replacement operation; if the edit operation is deletion, $t'[p_d : n]$ can be transformed to $t[m - n + p_d : m]$ by one deletion operation; if the edit operation is insertion, $t'[1 : q_1]$ can be transformed to $t[1 : q_1 + 1]$ by one insertion operation. Therefore, for the first partition $t'[p_1 = 1 : q_1]$, we select the substrings $t[1 : q_1 - 1]$, $t[1 : q_1]$, and $t[1 : q_1 + 1]$. For example, in Figure 5, the last partition is $t'[9 : 12]$, thus we select $t[5 : 9]$, $t[6 : 9]$ and $t[7 : 9]$ from t.

Case 3 - The Middle Partitions. Suppose a middle partition $t'[p_i : q_i]$ $(i \neq 1, d)$ has zero or one edit operation. To find its corresponding substrings in t, we can easily derive that their lengths cannot differ from the length of $t'[p_i : q_i]$ by more than 1. That is, the lengths of the corresponding substrings should be within $[q_i - p_i, q_i - p_i + 2]$. Next we need to determine starting positions of the corresponding substrings in t. Wang et al. [2009] presented that there are at most λ insertions or deletions before $t'[p_i : q_i]$, thus the starting positions of the corresponding substrings must be within $[p_i - \lambda, p_i + \lambda]$. Therefore, for a middle partition $t[p_i : q_i]$, we select the substrings of t whose starting positions are within $[p_i - \lambda, p_i + \lambda]$ and lengths are within $[q_i - p_i, q_i - p_i + 2]$. For example, consider the middle partition $t'[5 : 8]$ in Figure 5. Since $\lambda = 3$, the starting positions are within $[2, 8]$. For each starting position in $[2, 8]$, we select three substrings whose lengths are within $[3, 5]$. For example, we select $t[2 : 4]$, $t[2 : 5]$ and $t[2 : 6]$ for the starting position 2. We only select $t[7 : 9]$ for the starting postilion 7 since $t[7 : 10]$ and $t[7 : 11]$ exceeds the length of t.

In Figure 5, for all the partitions of t', we find a total of 21 corresponding substrings of t. Next, we propose two pruning techniques to reduce unnecessary substrings.

Minimal-Edit-Distance Pruning. Suppose $t[p_i : q_i]$ is the corresponding substring of the partition $t'[p'_i : q'_i]$. When computing the edit distance between t and t', $t[p_i : q_i]$ and $t'[p'_i : q'_i]$ should be aligned, their prefix strings $t[1 : p_i - 1]$ and $t'[1 : p'_i - 1]$ should be aligned, and their suffix strings $t[q_i + 1 : m]$ and $t'[q'_i + 1 : n]$ should be aligned. So the edit distance $\text{ED}(t, t')$ is the sum of $\text{ED}(t[p_i : q_i], t'[p'_i : q'_i])$, $\text{ED}(t[1 : p_i - 1], t'[1 : p'_i - 1])$ and $\text{ED}(t[q_i + 1 : m], t'[q'_i + 1 : n])$. We know that the edit distance between two strings is no smaller than their length difference. Thus we can compute the minimum of the edit distance,

$$\text{ED}(t, t') \geq |\xi| + |p_i - p'_i| + |(m - q_i) - (n - q'_i)|, \tag{11}$$

where $|\xi| = |(q_i - p_i) - (q'_i - p'_i)|$ is the length difference between $t[p_i : q_i]$ and $t'[p'_i : q'_i]$.

If the right side of Equation (11) is larger than λ, then we can prune the substrings $t[p_i : q_i]$. For example, in Figure 5, we can prune the corresponding substring $t[3 : 7]$ for the partition $t'[5 : 8]$, since the minimum of $\text{ED}(t, t')$ is $|1| + |3 - 5| + |(9 - 7) - (12 - 8)| = 5$ and 5 is larger than $\lambda = 3$.

Duplication Pruning. Recall three cases of selecting the corresponding substrings, we consider each partition independently, and thus some conditions may be repeatedly considered. For example, consider the substring $t[3 : 5]$ for the partition $t'[5 : 8]$ in Figure 5. On the left $t[1 : 2]$ of $t[3 : 5]$, it needs at least two edit operations to align $t[1 : 2]$ and $t'[1 : 4]$. Therefore, there exists at most one edit error on the right $t[6 : 9]$ of $t[3 : 5]$ due to the total edit distance $\lambda = 3$. Note that the condition that $t[6 : 9]$ has one or zero edit error has been considered in Case 2, and thus we can prune the substring $t[3 : 5]$.

Formally, to find the substrings of t, we first consider the first partition and the last partition. Then we consider the middle partitions from right to left. For the partition, $t'[p'_i : q'_i]$ and let k denote the number of partitions behind $t'[p'_i : q'_i]$. We can prune the substrings in t' with starting positions larger than $p'_i + \lambda - 2k$ (or smaller than $p'_i - (\lambda - 2k)$). This is because for each of such substrings, for example, $t[p_i : q_i]$, the edit operations before $t[p_i : q_i]$ will be larger than $\lambda - 2k$ and correspondingly, the edit operations after $t[p_i : q_i]$ will be smaller than $2k$ (otherwise the total edit distance is larger than λ). As there are k partitions behind $t[p_i : q_i]$, there at least exists one partition with zero or one edit error. As this partition has been considered, we can prune the substring $t[p_i : q_i]$.

In Figure 5, using minimal-edit-distance pruning we can prune ten substrings and using duplication pruning we can prune eight substrings. Using both of them, we can reduce the number of substrings from 21 to 8.

6. EXTENSION TO WEIGHTED TOKEN SETS

In real-world applications, different tokens may have different weights. For example, consider $T = \{\texttt{kobe}, \texttt{and}; \texttt{tracy}\}$. As and is a frequent token, matching the other tokens kobe and tracy are more important than matching and; thus kobe and tracy should have higher weights than and. We call a token set with weighted tokens as a weighted token set. In this section, we extend Fast-Join to support weighted token sets. We first define weighted fuzzy-token similarity in Section 6.1 and then propose two novel signature schemes for weighted token sets in Section 6.2 and Section 6.3, respectively.

6.1. Definition of Weighted Fuzzy-Token Similarity

We use *weighted overlap* to quantify the similarity of weighted token sets. The weighted overlap between T and T', denoted by $\mathcal{W}(T \cap T')$, is defined as the sum of the weights of the tokens in their intersection. For example, consider $T = \{(\texttt{kobe}, 8), (\texttt{and}, 1), (\texttt{tracy}, 10)\}$ and $T' = \{(\texttt{kobe}, 8), (\texttt{trany}, 14)\}$ (The number in each round bracket is the token's weight). Based on the definition, their intersection is $T \cap T' = \{(\texttt{kobe}, 8)\}$, so their weighted overlap is $\mathcal{W}(T \cap T') = 8$. Note that the weighted overlap neglects fuzzy matching tokens (e.g., (tracy,10) and (trany,14)). To address this problem, we extend weighted overlap to *weighted fuzzy overlap* as follows.

Given two weighted token sets, T and T', we construct a weighted bigraph $G = ((X, Y), E)$, where each vertex in $X(Y)$ is a token in T (T') and E is an edge set for pair $(t, t') \in T \times T'$. The weight of edge (t, t') quantifies the importance of (fuzzy) matching between t and t' which relies on three values: the edit similarity $\text{NED}(t, t')$ between t and t', the weight $\mathcal{W}(t)$ of t, and the weight $\mathcal{W}(t')$ of t'. There is an edge between t and t' if their edge weight $\mathcal{W}(t, t')$ is not smaller than a threshold δ. For example, consider $t = (\texttt{tracy}, 10)$ and $t' = (\texttt{trany}, 14)$. The weight of their edge is a combination of $\text{NED}(\texttt{tracy}, \texttt{trany}) = 0.8$, $\mathcal{W}(\texttt{tracy}) = 10$ and $\mathcal{W}(\texttt{trany}) = 14$. Instead of using a specific function to combine the three values, we define a general aggregation function

$$\mathcal{W}(t, t') = \mathcal{A}\big(\text{NED}(t, t'), \mathcal{W}(t), \mathcal{W}(t')\big), \tag{12}$$

where the aggregation function $\mathcal{A}(\cdot, \cdot, \cdot)$ satisfies two requirements.

(1) *Monotonicity.* Since a larger value of $\text{NED}(t, t')$, $\mathcal{W}(t)$, or $\mathcal{W}(t')$ should lead to a larger weight of an edge, the function is monotonically nondecreasing.
(2) *Symmetry.* Since the weight of an edge between t and t' should be the same as that between t' and t, the function is symmetric.

We generate tokens' weights using the well-known inverse document frequency (IDF) from the IR community [Baeza-Yates and Ribeiro-Neto 1999]. Intuitively, if a token (e.g., (and, 1)) is very frequent and occurs in a lot of token sets, it should be assigned a much smaller weight than other infrequent tokens (e.g., (kobe, 8)). The IDF method may also assign a high weight to typo tokens, since they are usually less frequent than the correctly spelled tokens. This problem would lead Equation (12) to a higher value than what it should be and outputs some dissimilar pairs of token sets as a result. As it is hard to predicate typo tokens, one heuristic solution is to assume the token with a higher weight as a typo and decrease the effect of its weight to the final value. We defer the detailed study of this problem to future work. In the remainder of the article, for ease of presentation, we use the following function as an example of Equation (12).

$$\mathcal{W}(t, t') = \text{NED}(t, t') \times \frac{\mathcal{W}(t) + \mathcal{W}(t')}{2}. \tag{13}$$

After constructing the weighted bigraph, we compute the maximum weight matching of the graph and define it as the weighted fuzzy overlap $T \,\widetilde{\cap}_\delta\, T'$. The corresponding

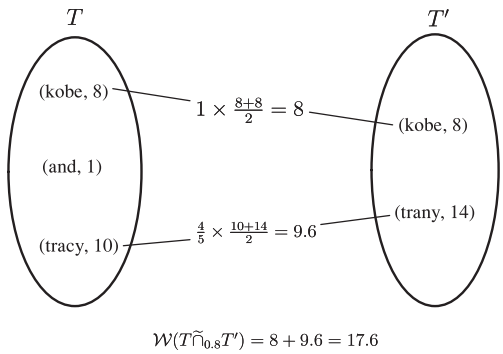


Fig. 6. Computing the weighted fuzzy overlap between two weighted token sets, T and T' ($\delta = 0.8$).

maximum weight is $\mathcal{W}(T \widetilde{\cap}_\delta T')$, which is the sum of the weights of the edges in $T \widetilde{\cap}_\delta T'$. Next we give an example to explain how to compute the weighted fuzzy overlap between two weighted token sets.

Example 6.1. Consider two weighted token sets, T = {(kobe,8), (and,1), (tracy,10)} and T' = {(kobe,8), (trany,14)}. To compute the weighted fuzzy overlap between T and T', we construct a weighted bigraph as shown in Figure 6. We first compute the edit similarity of each pair of tokens in $T \times T'$: $\text{NED}(\texttt{kobe}, \texttt{kobe}) = 1$, $\text{NED}(\texttt{kobe}, \texttt{trany}) = 0$, $\text{NED}(\texttt{and}, \texttt{kobe}) = 0$, $\text{NED}(\texttt{and}, \texttt{trany}) = 0.4$, $\text{NED}(\texttt{tracy}, \texttt{kobe}) = 0$, $\text{NED}(\texttt{tracy}, \texttt{trany}) = 0.8$. Suppose the given edit-similarity threshold is $\delta = 0.8$. Since only $\text{NED}(\texttt{kobe}, \texttt{kobe}) = 1 \geq 0.8$ and $\text{NED}(\texttt{tracy}, \texttt{trancy}) = 0.8$ are no smaller than $\delta = 0.8$, based on our definition, there are two (fuzzy) matching token pairs, thus we add two edges to the weighted bigraph. The weight of each edge is computed by Equation (13). For example, for the edge between tracy and trany, we have $\mathcal{W}(\texttt{tracy}) = 10$, $\mathcal{W}(\texttt{trany}) = 14$, so its weight is $\frac{4}{5} \times \frac{10+14}{2} = 9.6$. As the two edges do not share any common vertex, the maximum weight matching of the weighted bigraph consists of the two edges. The corresponding maximum weight is the sum of their weights, that is, $\mathcal{W}(T \widetilde{\cap}_{0.8} T') = 8 + 9.6 = 17.6$.

Using weighted fuzzy overlap, we define weighted fuzzy-token similarity.

Definition 6.2 (*Weighted Fuzzy-Token Similarity*). Given two strings s and s′ and an edit-similarity threshold δ, let T and T' be the weighted token sets of s and s′, respectively, we have the following.

—*Weighted Fuzzy-Dice Similarity*. $\text{WFDICE}_\delta(\mathsf{s}, \mathsf{s}') = \frac{2 \cdot \mathcal{W}(T \widetilde{\cap}_\delta T')}{\mathcal{W}(T) + \mathcal{W}(T')}$.

—*Weighted Fuzzy-Cosine Similarity*. $\text{WFCOSINE}_\delta(\mathsf{s}, \mathsf{s}') = \frac{\mathcal{W}(T \widetilde{\cap}_\delta T')}{\sqrt{\mathcal{W}(T) \cdot \mathcal{W}(T')}}$.

—*Weighted Fuzzy-Jaccard Similarity*. $\text{WFJACCARD}_\delta(\mathsf{s}, \mathsf{s}') = \frac{\mathcal{W}(T \widetilde{\cap}_\delta T')}{\mathcal{W}(T) + \mathcal{W}(T') - \mathcal{W}(T \widetilde{\cap}_\delta T')}$.

6.2. Weighted-Prefix-Filtering Signature Scheme

To make Fast-Join support weighted fuzzy-token similarity, we need to study signature schemes for weighted token sets. That is, given two weighted token sets, T and T', we generate signatures $Sig^\delta(T)$ and $Sig^\delta(T')$ such that if $\mathcal{W}(T \widetilde{\cap}_\delta T') \geq c$, then $Sig^\delta(T) \cap Sig^\delta(T') \neq \phi$. One simple signature scheme is to merge the signatures of all tokens, that is, $Sig^\delta(T) = \biguplus_{t \in T} sig^\delta(t)$ and $Sig^\delta(T') = \biguplus_{t' \in T'} sig^\delta(t')$. This method is feasible because if $\mathcal{W}(T \widetilde{\cap}_\delta T') \geq c$, then there at least exists a pair of tokens $(t, t') \in T \times T'$ such that $\text{NED}(t, t') \geq \delta$ (otherwise, $\mathcal{W}(T \widetilde{\cap}_\delta T') = 0$). Based on the definition of the signature

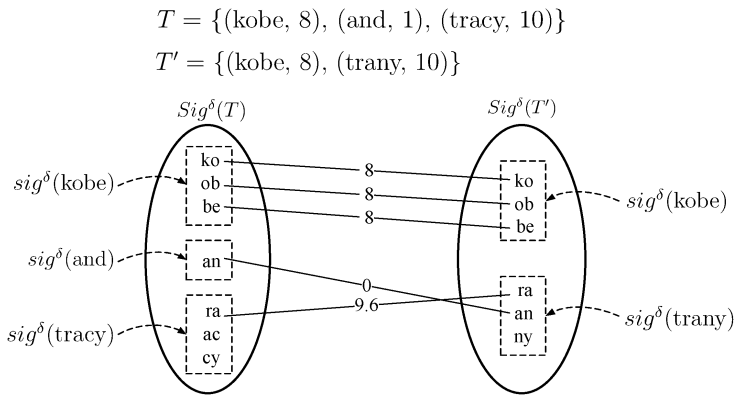


Fig. 7. A weighted bigraph G^δ built for $Sig^\delta(T)$ and $Sig^\delta(T')$ ($\delta = 0.8$).

scheme for tokens, we have $sig^\delta(t) \cap sig^\delta(t') \neq \phi$, thus $Sig^\delta(T) \cap Sig^\delta(T') \neq \phi$. To improve the simple signature scheme, we find some signatures in $Sig^\delta(T)$ and $Sig^\delta(T')$ can be removed. In this section, we propose a prefix-filtering-based signature scheme to reduce signatures. We develop a more sophisticated signature scheme to remove more signatures in next section.

We first consider the problem: "in which case we can remove the signatures $s_1, s_2, \ldots, s_k$ from $Sig^\delta(T)$ such that for any $T' \in R$, if $\mathcal{W}(T \widetilde{\cap}_\delta T') \geq c$, then $Sig^\delta(T) - \{s_1, s_2, \ldots, s_k\}$ still has overlap with $Sig^\delta(T')$?"

To explore the problem, we build a weighted bigraph G^δ for $Sig^\delta(T)$ and $Sig^\delta(T')$. Each signature in $Sig^\delta(T)$ and $Sig^\delta(T')$ is taken as a vertex. $Sig^\delta(T)$ and $Sig^\delta(T')$ represents two disjoint sets of vertices. For each pair of signatures in $Sig^\delta(T') \times Sig^\delta(T')$, if the two signatures in the pair are the same, we add a weighted edge between them. The weight of the edge is equal to the weight between the two tokens that generate the signatures of the pair. For example, consider two weighted token sets $T = \{(\texttt{kobe}, 8), (\texttt{and}, 1), (\texttt{tracy}, 10)\}$ and $T' = \{(\texttt{kobe}, 8), (\texttt{trany}, 14)\}$. Figure 7 shows the weighted bigraph G^δ built for $Sig^\delta(T)$ and $Sig^\delta(T')$. We first generate the signatures of three tokens `kobe`, `and`, `tracy` in T to obtain the set of vertices $Sig^\delta(T)$, and generate the signatures of two tokens `kobe`, `trany` in T' to obtain the other disjoint set of vertices $Sig^\delta(T')$. Then we add edges between $Sig^\delta(T)$ and $Sig^\delta(T')$. Since `ra` is a vertex in both $Sig^\delta(T)$ and $Sig^\delta(T')$, we add an edge for `ra`. As `ra` in $Sig^\delta(T)$ comes from the token `tracy`, and `ra` in $Sig^\delta(T')$ comes from the token `trany`, the weight of the edge for `ra` is equal to $\mathcal{W}(\texttt{tracy}, \texttt{trany}) = 9.6$. Similarly, for the vertices `ko`, `ob`, `be` and `an`, we add four edges to G^δ whose weights are $\mathcal{W}(\texttt{kobe}, \texttt{kobe}) = 8$, $\mathcal{W}(\texttt{kobe}, \texttt{kobe}) = 8$, $\mathcal{W}(\texttt{kobe}, \texttt{kobe}) = 8$, $\mathcal{W}(\texttt{and}, \texttt{trany}) = 0$, respectively.

By comparing G^δ with the weighted bigraph G built for T and T', we find that for each edge in G, there at least exists a corresponding edge in G^δ that has the same weight. Consider an edge (t, t') in G whose weight is $\mathcal{W}(t, t')$. Since t and t' are similar, we have $sig^\delta(t) \cap sig^\delta(t') \neq \phi$. For any signature in $sig^\delta(t) \cap sig^\delta(t')$, based on the definition of G^δ, there is an edge in G^δ that connects with it whose weight is $\mathcal{W}(t, t')$. Therefore, the weight of the maximum weight matching of G^δ must be no smaller than that of G. Since the weight of the maximum weight matching of G is no smaller than c, that is, $\mathcal{W}(T \widetilde{\cap}_\delta T') \geq c$, we have the weight of the maximum weight matching of G^δ is no smaller than c.

Next we study how the removal of the vertices $s_1, s_2, \ldots, s_k$ from $Sig^\delta(T)$ affects the weight of the maximum weight matching of G^δ. After removing a vertex s_i, we need to remove all the edges that connect with s_i from G^δ. Let $\mathcal{W}_{\max}(s_i)$ denote the maximum

weight of the edges that connect with s_i (We will discuss how to compute $\mathcal{W}_{\max}(s_i)$ in the later text). So the weight of the maximum weight matching of G^δ is at most reduced by $\mathcal{W}_{\max}(s_i)$. Similarly, after removing all the vertices $s_1, s_2, \ldots, s_k$ from $Sig^\delta(T)$, the weight of the maximum weight matching of G^δ is at most reduced by $\sum_{i=1}^{k} \mathcal{W}_{\max}(s_i)$. Since the weight of the maximum weight matching of G^δ is no smaller than c, if the total reduced weight is smaller than c, that is,

$$\sum_{i=1}^{k} \mathcal{W}_{\max}(s_i) < c, \tag{14}$$

then after removing the vertices $s_1, s_2, \ldots, s_k$ from $Sig^\delta(T)$, the weight of the maximum weight matching of G^δ is still larger than zero. That is, there is at least an edge between $Sig^\delta(T) - \{s_1, s_2, \ldots, s_k\}$ and $Sig^\delta(T')$, thus $Sig^\delta(T) - \{s_1, s_2, \ldots, s_k\}$ still has overlap with $Sig^\delta(T')$.

Now we discuss how to compute $\mathcal{W}_{\max}(s_i)$. Recall the definition of G^δ, the weight of an edge that connects with s_i in G^δ is equal to the weight between two tokens that, respectively, generate s_i in T and T'. Since T is given, the token that generates s_i in T can be easily obtained, denoted by t_{s_i}. However, for the token that generates s_i in T', since T' could be any token set in R, we need consider all possible tokens that generate s_i. Let R_{s_i} denote all the tokens in R whose signature sets contain s_i, and we have

$$\mathcal{W}_{\max}(s_i) = \max_{t' \in R_{s_i}} \mathcal{W}(t_{s_i}, t'). \tag{15}$$

Since $\mathcal{W}(t_{s_i}, t') = \mathcal{A}(\text{NED}(t_{s_i}, t'), \mathcal{W}(t_{s_i}), \mathcal{W}(t'))$ (Equation (12)), then we have

$$\mathcal{W}_{\max}(s_i) = \max_{t' \in R_{s_i}} \mathcal{A}(\text{NED}(t_{s_i}, t'), \mathcal{W}(t_{s_i}), \mathcal{W}(t')). \tag{16}$$

Note that it is expensive to compute $\mathcal{W}_{\max}(s_i)$ based on Equation (16), since we need to enumerate every $t' \in R_{s_i}$. To avoid such expensive computation, we compute an upper bound for $\mathcal{W}_{\max}(s_i)$, which can be computed efficiently. As $\mathcal{A}(\cdot, \cdot, \cdot)$ is a monotonic function, and $\text{NED}(t_{s_i}, t') \leq 1$, we have

$$\mathcal{W}_{\max}(s_i) \leq \mathcal{A}\left(1, \mathcal{W}(t_{s_i}), \max_{t' \in R_{s_i}} \mathcal{W}(t')\right). \tag{17}$$

The equation $\max_{t' \in R_{s_i}} \mathcal{W}(t')$ denotes the maximum weight of the tokens whose signature sets signature s_i. For ease of presentation, we use a simplified notation $\mathcal{W}_{\max}(t_{s_i})$ to replace $\max_{t' \in R_{s_i}} \mathcal{W}(t')$. Therefore, the upper bound of $\mathcal{W}_{\max}(s_i)$ is

$$\mathcal{W}^{\text{u}}_{\max}(s_i) = \mathcal{A}(1, \mathcal{W}(t_{s_i}), \mathcal{W}_{\max}(t_{s_i})). \tag{18}$$

Note that we do not need to compute $\mathcal{W}_{\max}(t_{s_i})$ on the fly. Instead, we precompute and store $\mathcal{W}_{\max}(t_s)$ for all signatures as follows. We enumerate each signature s of each token $t \in R$, and compare $\mathcal{W}_{\max}(t_s)$ with $\mathcal{W}(t)$. If $\mathcal{W}(t)$ is larger, we update $\mathcal{W}_{\max}(t_s)$ to $\mathcal{W}(t)$. After the enumeration, we obtain $\mathcal{W}_{\max}(t_{s_i})$ for all signatures. For example, consider $R = \{T_1, T_2, T_3, T_4\}$ in Figure 8. We first generate the signatures of each token in R, that is, $sig^\delta(\texttt{and}) = \{\texttt{an}\}$, $sig^\delta(\texttt{trany}) = \{\texttt{ra}, \texttt{an}, \texttt{ny}\}$, $sig^\delta(\texttt{kobe}) = \{\texttt{ko}, \texttt{ob}, \texttt{be}\}$, $sig^\delta(\texttt{tracy}) = \{\texttt{ra}, \texttt{ac}, \texttt{cy}\}$, $sig^\delta(\texttt{age}) = \{\texttt{ag}\}$, and $sig^\delta(\texttt{trcy}) = \{\texttt{tr}, \texttt{rc}, \texttt{cy}\}$. Then we enumerate the signatures of the six tokens. For the first token and, it has one signature an. We compare $\mathcal{W}_{\max}(t_{\texttt{an}}) = 0$ with $\mathcal{W}(\texttt{and}) = 1$. (The weight of each token can be found in Figure 8). Since $\mathcal{W}(\texttt{and})$ is larger, we set $\mathcal{W}_{\max}(t_{\texttt{an}}) = 1$. For the second token trany, it has three signatures ra, an, and ny. We, respectively, compare $\mathcal{W}_{\max}(t_{\text{“}\texttt{ra}\text{”}}) = 0$, $\mathcal{W}_{\max}(t_{\text{“}\texttt{an}\text{”}}) = 1$

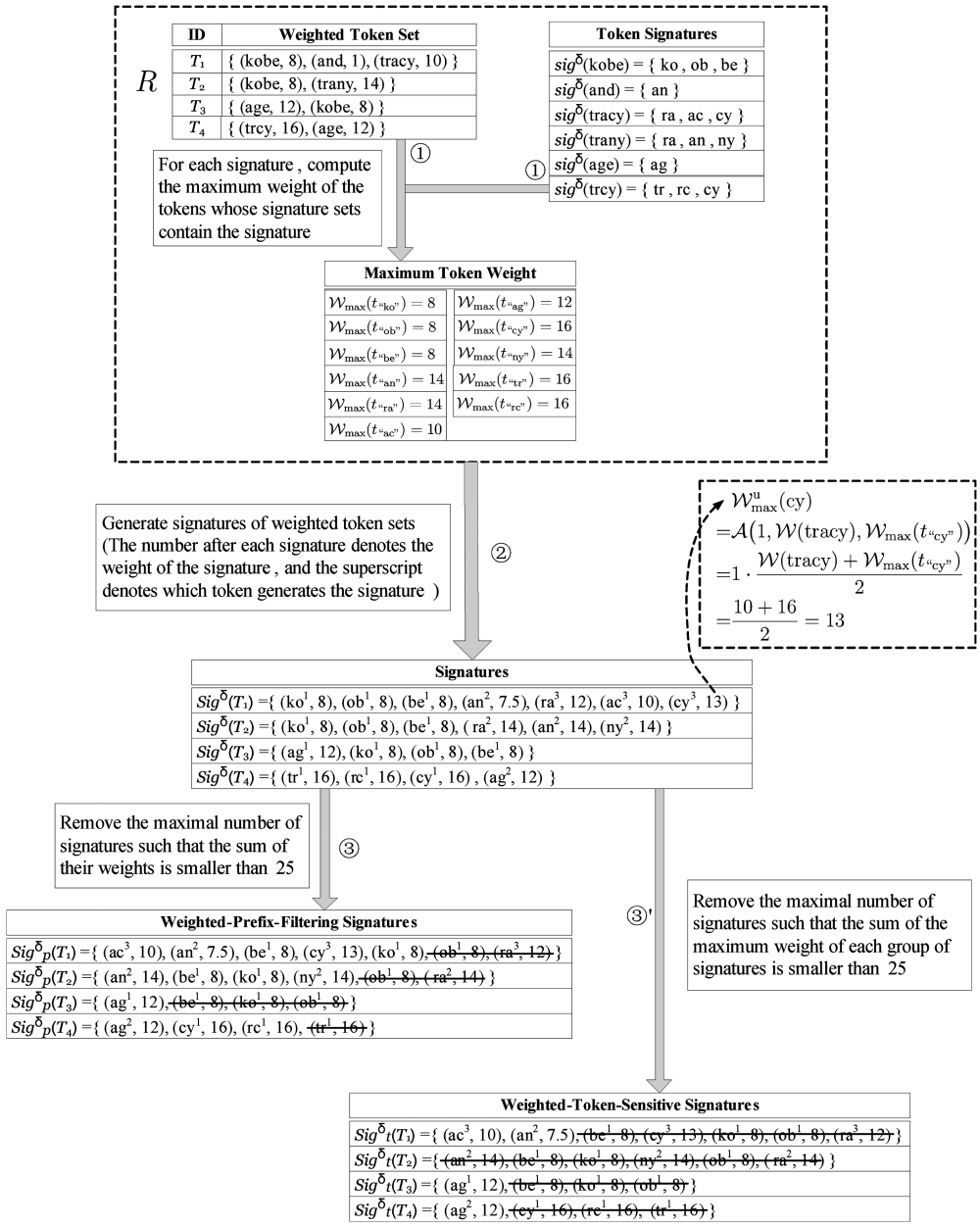


Fig. 8. Weighted-prefix-filtering signatures and weighted-token-sensitive signatures of the weighted token sets in $\mathcal{R}$ ($\delta = 0.8, c = 25$).

and $\mathcal{W}_{\max}(t_{\text{“ny”}}) = 0$ with $\mathcal{W}(\texttt{trany}) = 14$, and set $\mathcal{W}_{\max}(t_{\text{“ra”}}) = 14$, $\mathcal{W}_{\max}(t_{\text{“an”}}) = 14$ and $\mathcal{W}_{\max}(t_{\text{“ny”}}) = 14$. Similarly, after enumerating the other four tokens, we obtain $\mathcal{W}_{\max}(t_s)$ for all signatures in Figure 8.

When deciding whether $s_1, s_2, \ldots, s_k$ can be removed from $Sig^{\delta}(T)$, we first compute $\mathcal{W}^{\text{u}}_{\max}(s_i)$ for each signature s_i ($i \in [1, k]$), and call it as the weight of signature s_i, and

ALGORITHM 2: WeightedPrefixFilteringSignature(T, c)

Input: T is a weighted token set
c is a weighted-fuzzy-overlap threshold
Output: $Sig_p^\delta(T)$ is the weighted-prefix-filtering signature set of T

```
1  begin
2      Sig_p^δ(T) = ⊎_{t∈T} sig^δ(t);
3      sum = 0;
4      for each s ∈ Sig_p^δ(T) in decreasing global order on signatures do
5          W^u_max(s) = A(1, W(t_s), W_max(t_s));
6          sum = sum + W^u_max(s);
7          if sum ≥ c then
8              break;
9          Remove s from Sig_p^δ(T);
10     return Sig_p^δ(T);
```

Fig. 9. Algorithm of generating weighted prefix-filtering signatures for a weighted token set.

then compare the sum of signatures' weights with the threshold c, that is,

$$\sum_{i=1}^{k} \mathcal{W}^{\mathrm{u}}_{\max}(s_i) < c. \tag{19}$$

If the equation holds, as $\mathcal{W}^{\mathrm{u}}_{\max}(s_i)$ is the upper bound of $\mathcal{W}_{\max}(s_i)$ ($i \in [1, k]$), Equation (14) must hold, thus we can remove $s_1, s_2, \ldots, s_k$ from $Sig^\delta(T)$. For example, consider $Sig^\delta(T_1) = \{\mathtt{ko}^1, \mathtt{ob}^1, \mathtt{be}^1, \mathtt{an}^2, \mathtt{ra}^3, \mathtt{ac}^3, \mathtt{cy}^3\}$ in Figure 8. Suppose $c = 25$. To decide whether ac and cy can be removed from $Sig^\delta(T_1)$, we need to compute the weights of ac and cy, that is, $\mathcal{W}^{\mathrm{u}}_{\max}(\mathtt{ac})$ and $\mathcal{W}^{\mathrm{u}}_{\max}(\mathtt{cy})$. Figure 8 shows how to compute $\mathcal{W}^{\mathrm{u}}_{\max}(\mathtt{cy})$. Since cy comes from tracy in T_1, we have $t_{\text{“cy”}} = \mathtt{tracy}$, thus $\mathcal{W}(t_{\text{“cy”}}) = \mathcal{W}(\mathtt{tracy}) = 10$. As already computed, $\mathcal{W}_{\max}(t_{\mathrm{cy}}) = 16$. Based on Equation (18), we obtain $\mathcal{W}^{\mathrm{u}}_{\max}(\mathtt{cy}) = \mathcal{A}(1, \mathcal{W}(t_{\text{“cy”}}), \mathcal{W}_{\max}(t_{\mathrm{cy}})) = 1 \cdot \frac{10+16}{2} = 13$. Similarly, we can also obtain $\mathcal{W}^{\mathrm{u}}_{\max}(\mathtt{ac}) = 10$ for the signature ac. Since $c = 25$ and $\mathcal{W}^{\mathrm{u}}_{\max}(\mathtt{cy}) + \mathcal{W}^{\mathrm{u}}_{\max}(\mathtt{ac}) = 23 < 25$, based on Equation (19), the signatures cy and ac can be removed from $Sig^\delta(T_1)$.

Now we have shown that the signatures $s_1, s_2, \ldots, s_k$ can be removed from $Sig^\delta(T)$ if $\sum_{i=1}^{k} \mathcal{W}^{\mathrm{u}}_{\max}(s_i) < c$. The next question is can we also remove the signatures from $Sig^\delta(T')$. Inspired by the prefix-filtering signature scheme, we fix a global order on all signatures and remove the signatures based on the global order. In this case, the signatures can be removed from both $Sig^\delta(T)$ and $Sig^\delta(T')$. We call such signature scheme as weighted-prefix-filtering signature scheme. Figure 9 gives the pseudocode of the signature scheme. Let $Sig_p^\delta(T)$ denote the signature set of T generated by the weighted-prefix-filtering signature scheme. Initially, $Sig_p^\delta(T) = Sig^\delta(T)$. Then we remove signatures from $Sig_p^\delta(T)$ based on the decreasing global order on signatures. Let *sum* be the sum of the removed signatures' weights. For each signature s, we compute s's weight $\mathcal{W}^{\mathrm{u}}_{\max}(s)$ and add it to *sum*. If $sum \geq c$, that is, Equation (19) does not hold, we stop removing signatures and return $Sig_p^\delta(T)$; otherwise, we remove the signature s from $Sig_p^\delta(T)$, and repeat the previous process. For example, Figure 8 shows the weighted-prefix-filtering signatures for four weighted token sets in R. Consider $T_1 = \{(\mathtt{kobe}, 8), (\mathtt{and}, 1), (\mathtt{tracy}, 10)\}$. We first obtain $Sig^\delta(T_1) = \{(\mathtt{ac}^3, 10), (\mathtt{an}^2, 7.5), (\mathtt{be}^1, 8), (\mathtt{cy}^3, 13), (\mathtt{ko}^1, 8), (\mathtt{ob}^1, 8), (\mathtt{ra}^3, 12)\}$ (The signatures are sorted based on alphabetical order, and the number in the round bracket

is signature's weight). Next we remove the signatures in $Sig^{\delta}(T_1)$ from back to front. The last two signatures ($\texttt{ob}^1$, 8) and ($\texttt{ra}^3$,12) can be removed, since $c = 25$ and the sum of their weights is $\mathcal{W}^{\mathrm{u}}_{\max}(\texttt{ra}) + \mathcal{W}^{\mathrm{u}}_{\max}(\texttt{ob}) = 20 < 25$. Note that we are unable to continue to remove ($\texttt{ko}^1$, 8), since $\mathcal{W}^{\mathrm{u}}_{\max}(\texttt{ra}) + \mathcal{W}^{\mathrm{u}}_{\max}(\texttt{ob}) + \mathcal{W}^{\mathrm{u}}_{\max}(\texttt{ko}) = 28 > 25$. Therefore, $Sig^{\delta}_p(T_1) = \{(\texttt{ac}^3, 10), (\texttt{an}^2, 7.5), (\texttt{be}^1, 8), (\texttt{cy}^3, 13), (\texttt{ko}^1, 8)\}$. Lemma 6.3 shows the correctness of the algorithm.

LEMMA 6.3. *Given two weighted token sets T and T' and a threshold c, if $\mathcal{W}(T \widetilde{\cap}_{\delta} T') \geq c$, then $Sig^{\delta}_p(T) \cap Sig^{\delta}_p(T') \neq \phi$.*

PROOF. Let $\Omega = Sig^{\delta}(T) - Sig^{\delta}_p(T)$ denote the set of removed signatures from $Sig^{\delta}(T)$. Let $\Omega' = Sig^{\delta}(T') - Sig^{\delta}_p(T')$ denote the set of removed signatures from $Sig^{\delta}(T')$. We prove it by contradiction. Assume the lemma does not hold. That is, there exists T and T' such that if $\mathcal{W}(T \widetilde{\cap}_{\delta} T') \geq c$, then $Sig^{\delta}_p(T) \cap Sig^{\delta}_p(T') = \phi$.

Based on the definition of $Sig^{\delta}_p(T)$, we have

$$Sig^{\delta}_p(T) \cap Sig^{\delta}(T') \neq \phi. \tag{20}$$

And since $Sig^{\delta}(T') = Sig^{\delta}_p(T') + \Omega'$, and $Sig^{\delta}_p(T) \cap Sig^{\delta}_p(T') = \phi$, we have

$$Sig^{\delta}_p(T) \cap \Omega' \neq \phi. \tag{21}$$

Similarly, we can also deduce that

$$Sig^{\delta}_p(T') \cap \Omega \neq \phi. \tag{22}$$

Since the signatures in $Sig^{\delta}(T)(Sig^{\delta}(T'))$ are sorted based on a global order, for any signature in $Sig^{\delta}_p(T)\,(Sig^{\delta}_p(T'))$, it must rank before all the signatures in $\Omega\,(\Omega')$. Suppose $s_1 \in Sig^{\delta}_p(T) \cap \Omega'$ and $s_2 \in Sig^{\delta}_p(T') \cap \Omega$. On one hand, since $s_1 \in Sig^{\delta}_p(T)$ and $s_2 \in \Omega$, s_1 ranks before s_2 in the global order. However, on the other hand, since $s_2 \in Sig^{\delta}_p(T')$ and $s_1 \in \Omega'$, we can also deduce that s_2 ranks before s_1 in the same global order. Hence, $s_1 = s_2$. However, as $s_1 \in Sig^{\delta}_p(T)$ and $s_2 \in Sig^{\delta}_p(T')$, $Sig^{\delta}_p(T)$ and $Sig^{\delta}_p(T')$ have the common signature s_1 $(= s_2)$, which contradicts with $Sig^{\delta}_p(T) \cap Sig^{\delta}_p(T') = \phi$. Therefore, the assumption does not hold, and the lemma is proved. □

6.3. Weighted-Token-Sensitive Signature Scheme

When deciding how many signatures can be removed, weighted-prefix-filtering signature scheme does not consider which token each removed signature comes from. Based on this observation, in this section, we propose a more sophisticated signature scheme, called weighted-token-sensitive signature scheme, which takes into account which token each removed signature comes from. This method is able to remove more signatures than the weighted-prefix-filtering signature scheme.

Revisit the problem "in which case we can remove the signatures $s_1, s_2, \ldots, s_k$ from $Sig^{\delta}(T)$ such that for any $T' \in R$, if $\mathcal{W}(T \widetilde{\cap}_{\delta} T') \geq c$, then $Sig^{\delta}(T) - \{s_1, s_2, \ldots, s_k\}$ still has overlap with $Sig^{\delta}(T')$?"

As discussed in the previous section, if $\mathcal{W}(T \widetilde{\cap}_{\delta} T') \geq c$, the weight of the maximum weight matching of G^{δ} built for $Sig^{\delta}(T)$ and $Sig^{\delta}(T')$ is no smaller than c. Next we seek to derive a stronger condition about G^{δ} by taking into account which token each signature comes from. We find there exists a set of edges $\mathbb{E}$ in G^{δ} such that (1) the sum of the weights of these edges, denoted by $\mathcal{W}(\mathbb{E})$, is no smaller than c; (2) the vertices (i.e., signatures) connected by these edges come from different tokens. To prove that, we construct the edge set $\mathbb{E}$ as follows. For each token pair (t, t') in $T \widetilde{\cap}_{\delta} T'$, since t and t' are similar, we have $sig^{\delta}(t) \cap sig^{\delta}(t') \neq \phi$. For any signature in $sig^{\delta}(t) \cap sig^{\delta}(t')$, based

on the definition of G^δ, there is an edge in G^δ that connects with the signature whose weight is $\mathcal{W}(t, t')$. We choose any one of these edges and add it into $\mathbb{E}$. Obviously, the sum of the weights of the edges in $\mathbb{E}$ is $\mathcal{W}(\mathbb{E}) = \sum_{(t,t')\in T\widetilde{\cap}_\delta T'} \mathcal{W}(t, t') = \mathcal{W}(T\widetilde{\cap}_\delta T') \geq c$. Since $T\widetilde{\cap}_\delta T'$ is the maximum weight matching of G^δ, there is no common token among the token pairs in $T\widetilde{\cap}_\delta T'$, thus we have the signatures connected by the edges in $\mathbb{E}$ coming from different tokens. For example, consider the weighted bigraph G^δ in Figure 7. As shown in Figure 6, $T\widetilde{\cap}_\delta T'$ contains two token pairs, one is (kobe, kobe), and the other is (tracy, trany). So we need to add two edges into $\mathbb{E}$. For the first token pair, since $sig^\delta(\texttt{kobe}) \cap sig^\delta(\texttt{kobe}) = \{\texttt{ko}, \texttt{ob}, \texttt{be}\}$, there are three edges that, respectively, connect with ko, ob, and be, and we add one of the three edges into $\mathbb{E}$, for example, the edge connecting with ko, and its weight is $\mathcal{W}(\texttt{kobe}, \texttt{kobe})$. For the second edge, since $sig^\delta(\texttt{tracy}) \cap sig^\delta(\texttt{trany}) = \{\texttt{ra}\}$, there is only one edge that connects with ra, and we add the edge into $\mathbb{E}$, and its weight is $\mathcal{W}(\texttt{tracy}, \texttt{trany})$. After adding the two edges, we have $\mathcal{W}(\mathbb{E}) = \mathcal{W}(\texttt{kobe}, \texttt{kobe}) + \mathcal{W}(\texttt{tracy}, \texttt{trany})$ that is equal to $\mathcal{W}(T\widetilde{\cap}_\delta T') \geq c$. In addition, the signatures ko and ra connected by these two edges come from different tokens, that is, kobe and tracy in T, and kobe and trany in T'.

Next we study how the removal of the signatures $s_1, s_2, \ldots, s_k$ from $Sig^\delta(T)$ affects the edges in $\mathbb{E}$. We divide $s_1, s_2, \ldots, s_k$ into $|T|$ groups, $\mathcal{G}^1, \mathcal{G}^2, \ldots, \mathcal{G}^{|T|}$, where $\mathcal{G}^j$ ($j \in [1, |T|]$) consists of the signatures that come from the jth token of T. For each group of signatures (e.g., $\mathcal{G}^j$), if we remove them from $Sig^\delta(T)$, since they come from the same token, that is, jth token, based on the definition of $\mathbb{E}$, there is at most one edge connecting with them, thus at most one edge will be removed, whose weight is no larger than the maximum weight of the signatures in $\mathcal{G}^j$, that is,

$$\max_{s\in\mathcal{G}^j} \left\{\mathcal{W}^{\mathrm{u}}_{\max}(s)\right\}. \tag{23}$$

Therefore, after removing all groups of signatures from $Sig^\delta(T)$, the sum of the weights of the removed edges in $\mathbb{E}$ is no larger than

$$\sum_{1\leq j\leq |T|} \max_{s\in\mathcal{G}^j} \left\{\mathcal{W}^{\mathrm{u}}_{\max}(s)\right\}. \tag{24}$$

Since the sum of the weights of all edges in $\mathbb{E}$ is $\mathcal{W}(\mathbb{E}) \geq c$, if the following equation holds

$$\sum_{1\leq j\leq |T|} \max_{s\in\mathcal{G}^j} \left\{\mathcal{W}^{\mathrm{u}}_{\max}(s)\right\} < c, \tag{25}$$

then after removing all groups of signatures, that is, $s_1, s_2, \ldots, s_k$, from $Sig^\delta(T)$, the sum of the weights of the rest edges in $\mathbb{E}$ is still larger than zero. That is, there is at least an edge between $Sig^\delta(T) - \{s_1, s_2, \ldots, s_k\}$ and $Sig^\delta(T')$, thus $Sig^\delta(T) - \{s_1, s_2, \ldots, s_k\}$ still has overlap with $Sig^\delta(T')$. Recall that weighted-prefix-filtering signature scheme uses Equation (19) to decide whether $s_1, s_2, \ldots, s_k$ can be removed. It is worth noting that if Equation (19) holds, Equation (25) must hold. Therefore, weighted-token-sensitive signature scheme can remove no smaller number of signatures than weighted-prefix-filtering signature scheme.

For example, consider $Sig^\delta(T_1) = \{(\texttt{ko}^1,8), (\texttt{ob}^1,8), (\texttt{be}^1,8), (\texttt{an}^2,7.5), (\texttt{ra}^3,12), (\texttt{ac}^3,10), (\texttt{cy}^3,13)\}$ in Figure 8, where the superscript of each signature denotes which token generates the signature. Suppose $c = 25$. To decide whether the last four signatures $(\texttt{an}^2,7.5)$, $(\texttt{ra}^3,12)$, $(\texttt{ac}^3,10)$, $(\texttt{cy}^3,13)$ can be removed from $Sig^\delta(T_1)$, we divide them into $|T_1| = 3$ groups according to their superscripts, $\mathcal{G}^1 = \{\}$, $\mathcal{G}^2 = \{(\texttt{an}^2, 7.5)\}$, and $\mathcal{G}^3 = \{(\texttt{ra}^3, 12), (\texttt{ac}^3, 10), (\texttt{cy}^3, 13)\}$. For $\mathcal{G}^1 = \{\}$, as there is no signature in it, the maximum weight of the signatures in $\mathcal{G}^1$ is 0; For $\mathcal{G}^2$, as there is only one signature, and

ALGORITHM 3: WeightedTokenSensitiveSignature(T, c)

Input: T is a weighted token set
c is a weighted-fuzzy-overlap threshold
Output: $Sig_t^\delta(T)$ is the weighted-token-sensitive signature set of T

```
1  begin
2      Sig_t^δ(T) = ⊎_{t∈T} sig^δ(t);
3      Let H be a hash map from group id to maximum weight;
4      Initialize H[j] = 0 for j ∈ [1, |T|];
5      for each s^tid ∈ Sig_t^δ(T) in decreasing global order on signatures do
6          w = W^u_max(s^tid);
7          if H[tid] < w then
8              H[tid] = w;
9          if Σ_j H[j] ≥ c then
10             break;
11         Remove s^tid from Sig_t^δ(T);
12     return Sig_t^δ(T);
```

Fig. 10. Algorithm of generating weighted token-sensitive signatures for a weighted token set.

its weight is $\mathcal{W}^{\mathrm{u}}_{\max}(\mathtt{an}^2) = 7.5$, the maximum weight of the signatures in $\mathcal{G}^2$ is 7.5; For $\mathcal{G}^3$, as there are three signatures, and their weights are $\mathcal{W}^{\mathrm{u}}_{\max}(\mathtt{ra}^3) = 12$, $\mathcal{W}^{\mathrm{u}}_{\max}(\mathtt{ac}^3) = 10$, $\mathcal{W}^{\mathrm{u}}_{\max}(\mathtt{cy}^3) = 13$, the maximum weight of the signatures in $\mathcal{G}^3$ is 13. We add up the maximum weights of $\mathcal{G}^1$, $\mathcal{G}^2$ and $\mathcal{G}^3$ and obtain $0 + 7.5 + 13 = 20.5$. Since their sum is smaller than $c = 25$, based on Equation (25), weighted-token-sensitive signature scheme can remove the four signatures. But note that the weighted-prefix-filtering signature scheme cannot remove the four signatures, since the sum of the weights of the four signatures is $7.5+12+10+13 = 42.5 \not< c = 25$ (i.e., Equation (19) does not hold).

Now we have shown that the signatures $s_1, s_2, \ldots, s_k$ can be removed from $Sig^\delta(T)$ if Equation (25) holds. The next question is that can we also remove the signatures from $Sig^\delta(T')$. We find if fixing a global order on all signatures and removing the signatures based on the global order, then we can remove the signatures from both $Sig^\delta(T)$ and $Sig^\delta(T')$. Figure 10 illustrates the pseudocode of the weighted-token-sensitive signature scheme. Let $Sig_t^\delta(T)$ denote the signature set of T generated by the weighted-token-sensitive signature scheme. Initially, $Sig_t^\delta(T) = Sig^\delta(T)$. Then we remove signatures from $Sig_t^\delta(T)$ based on the decreasing global order on signatures. We maintain a hash map $\mathcal{H}$ to store the maximum weight of each group of signatures. Initially, the maximum weight of each group is set to zero. For each signature s^{tid}, we compare its weight with the maximum weight $\mathcal{H}[tid]$ of its group: if s^{tid}'s weight is larger, we update $\mathcal{H}[tid]$ to s^{tid}'s weight. If the sum of the maximum weights of all groups $\sum_j \mathcal{H}[j]$ is larger than or equal to the threshold c, we stop scanning the following signatures and return the signature set $Sig_t^\delta(T)$; otherwise, we remove s^{tid} from $Sig_t^\delta(T)$ and scan the next signature. Lemma 6.4 shows the correctness of the algorithm.

For example, Figure 8 shows the weighted-token-sensitive signatures for four weighted token sets in R. Consider $T_1 = \{(\mathtt{kobe}, 8), (\mathtt{and}, 1), (\mathtt{tracy}, 10)\}$. We first obtain $Sig^\delta(T_1) = \{(\mathtt{ac}^3, 10), (\mathtt{an}^2, 7.5), (\mathtt{be}^1, 8), (\mathtt{cy}^3, 13), (\mathtt{ko}^1, 8), (\mathtt{ob}^1, 8), (\mathtt{ra}^3, 12)\}$ (The signatures are sorted based on alphabetical order), and initialize $Sig_t^\delta(T_1)$ as $Sig^\delta(T_1)$. Next we remove the signatures in $Sig_t^\delta(T_1)$ from back to front. For the last signature $(\mathtt{ra}^3, 12)$, we compare its weight 12 with $\mathcal{H}[3] = 0$, as $\mathtt{ra}^3$'s weight is larger, we set $\mathcal{H}[3] = 12$. As $\mathcal{H}[1] + \mathcal{H}[2] + \mathcal{H}[3] = 12$ is smaller than $c = 25$, we remove $(\mathtt{ra}^3, 12)$ from $Sig_t^\delta(T_1)$ and scan the following signatures. Similarly, we can also remove the following signatures

Table I. Dataset Statistics

Data Sets	Sizes	*avg_token_no*	*max_token_no*	*min_token_no*
Query Log	1,000,000	3.35	132	1
Author	613,542	2.77	8	1

($\mathtt{ob}^1$,8), ($\mathtt{ko}^1$,8), ($\mathtt{cy}^3$,13), ($\mathtt{be}^1$,8) and obtain $\mathcal{H}[1] = 8$ and $\mathcal{H}[2] = 0$ and $\mathcal{H}[3] = 13$. Next we scan the next signature ($\mathtt{an}^2$,7.5). Since its weight is larger than $\mathcal{H}[2] = 0$, we set $\mathcal{H}[2] = 7.5$. As $\mathcal{H}[1]+\mathcal{H}[2]+\mathcal{H}[3] = 8+7.5+13 = 28.5$ is not smaller than $c = 25$, we stop removing signatures and return the final signature set $Sig_t^\delta(T_1) = \{(\mathtt{ac}^3, 10), (\mathtt{an}^2, 7.5)\}$.

LEMMA 6.4. *Given two weighted token sets T and T' and a threshold c, if $\mathcal{W}(T \widetilde{\cap}_\delta T') \geq c$, then $Sig_t^\delta(T) \cap Sig_t^\delta(T') \neq \phi$.*

PROOF. Let $\Omega = Sig^\delta(T) - Sig_t^\delta(T)$ denote the set of removed signatures from $Sig^\delta(T)$. Let $\Omega' = Sig^\delta(T') - Sig_t^\delta(T')$ denote the set of removed signatures from $Sig^\delta(T')$. We prove it by contradiction. Assume the lemma does not hold. That is, there exists T and T' such that if $\mathcal{W}(T \widetilde{\cap}_\delta T') \geq c$, then $Sig_t^\delta(T) \cap Sig_t^\delta(T) = \phi$.

Based on the definition of $Sig_t^\delta(T)$, we have

$$Sig_t^\delta(T) \cap Sig^\delta(T') \neq \phi. \tag{26}$$

And since $Sig^\delta(T') = Sig_t^\delta(T') + \Omega'$, and $Sig_t^\delta(T) \cap Sig_t^\delta(T') = \phi$, we have

$$Sig_t^\delta(T) \cap \Omega' \neq \phi. \tag{27}$$

Similarly, we can also deduce that

$$Sig_t^\delta(T') \cap \Omega \neq \phi. \tag{28}$$

Since the signatures in $Sig^\delta(T)(Sig^\delta(T'))$ are sorted based on the same global order, for any signature in $Sig_t^\delta(T)$ ($Sig_t^\delta(T')$), it must rank before all the signatures in Ω (Ω'). Suppose $s_1 \in Sig_t^\delta(T) \cap \Omega'$ and $s_2 \in Sig_t^\delta(T') \cap \Omega$. On one hand, since $s_1 \in Sig_t^\delta(T)$ and $s_2 \in \Omega$, s_1 ranks before s_2 in the global order. However, on the other hand, since $s_2 \in Sig_t^\delta(T')$ and $s_1 \in \Omega'$, we can also deduce that s_2 ranks before s_1 in the same global order. Hence, $s_1 = s_2$. However, as $s_1 \in Sig_t^\delta(T)$ and $s_2 \in Sig_t^\delta(T')$, $Sig_t^\delta(T)$ and $Sig_t^\delta(T')$ have the common signature s_1 ($= s_2$), which contradicts with $Sig_t^\delta(T) \cap Sig_t^\delta(T') = \phi$. Therefore, the assumption does not hold, and the lemma is proved. □

7. EXPERIMENTAL STUDY

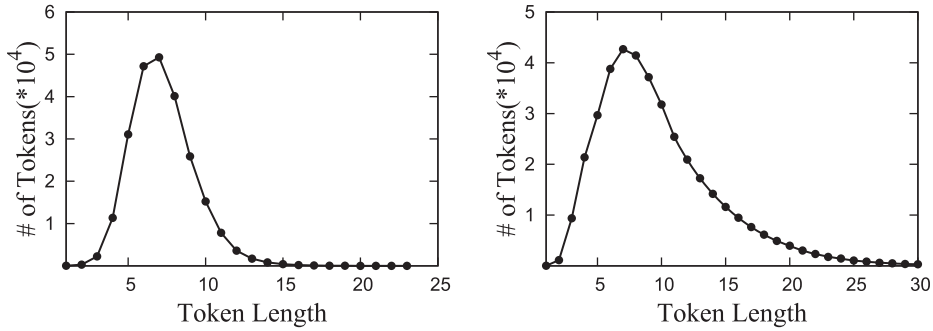
We used two real datasets and evaluated the effectiveness and the efficiency of our proposed methods.

Dataset. (1) AOL Query Log:[3] we generate two sets of strings, and each dataset included one million distinct real keyword queries. (2) DBLP Author: we extracted author names from DBLP dataset.[4] We also generate two sets of strings and each dataset included 0.6 million real person names. Table I illustrates detailed statistical information of the datasets, which gives the number of strings, the average number of tokens in a string, the maximal number of tokens in a string, and the minimal number of tokens in a string. Figures 11(a)–11(b) show the length distribution of tokens.

We implemented all the algorithms in C++ and compiled using GCC 4.2.3 with -O3 flag. We used inverse document frequency (IDF) to sort the signatures. All the

[3]`http://www.gregsadetsky.com/aol-data/`.

[4]`http://www.informatik.uni-trier.de/~ley/db`.



(a) Author. (a) Query Log.

Fig. 11. Token length distribution.

experiments were run on a Ubuntu Linux machine with an Intel Core 2 Quad E5420 2.50GHz processor and 4GB memory.

7.1. Fuzzy-Token Similarity

In this section, we aim to make a thorough analysis of fuzzy-token similarity in order to examine the motivation of designing this new similarity function. We chose 100,000 queries from the Query Log dataset and computed the similar string pairs using fuzzy-token similarity and existing similarity functions. It is worth noting that there are a lot of similarity functions for string matching. Some previous papers [Chandel et al. 2007; Cohen et al. 2003] have done excellent works in comparing their performance in various domains. They have shown that sophisticated similarity functions, such as SoftTFIDF [Cohen et al. 2003] and Language Model Similarity [Chandel et al. 2007], can typically quantify string similarity better. Unfortunately, there does not exist any efficient similarity-join algorithm for these similarity functions. Since the goal of this article is to extend string similarity join to tolerant fuzzy-token matching, in the experiment, we only compare with the existing similarity functions that can support efficient similarity-join algorithms. Nevertheless, it is also important to explore new similarity-join algorithms for other sophisticated similarity functions, and we will study this problem in future work.

7.1.1. Evaluating Fuzzy-Token Matching. Fuzzy-token similarity not only takes into account exact-token matching, but also fuzzy-token matching. To investigate the importance of fuzzy-token matching, we compared the quality of the results generated by jaccard similarity, which only considers exact-token matching, and fuzzy-jaccard similarity, which considers both exact-token matching and fuzzy-token matching. Specifically, we varied the (fuzzy-)jaccard threshold from 0.95 to 0.7 and compared the similar string pairs generated by jaccard similarity and fuzzy-jaccard similarity, respectively. Table II shows the number of results (i.e., the number of string pairs above the threshold) and the result precision (i.e., the percentage of correctly identified matching pairs out of all string pairs above the threshold). To evaluate the result precision, we randomly selected 100 results from the generated similar pairs and asked five research members from our group to evaluate the results blindly. In the following experiments, unless otherwise stated, we used the same method to evaluate the result precision. From the table, we see that fuzzy-jaccard similarity can identify many more results without decreasing the precision than jaccard similarity. For example, when the threshold is $\tau = 0.8$, fuzzy-jaccard similarity returned 1,520 pairs with 93% precision, while, jaccard similarity only returned 415 pairs with 94%. That is, fuzzy-token matching

Table II. Evaluating the Effectiveness of Fuzzy-Token Matching ($\delta = 0.8$)

τ	Jaccard Similarity		Fuzzy-Jaccard Similarity	
	# of Results	Precision(%)	# of Results	Precision(%)
0.95	127	100	212	99
0.9	132	99	560	100
0.85	166	99	986	98
0.8	405	94	1,520	93
0.75	1,100	90	2,344	86
0.7	1,201	69	2,698	84

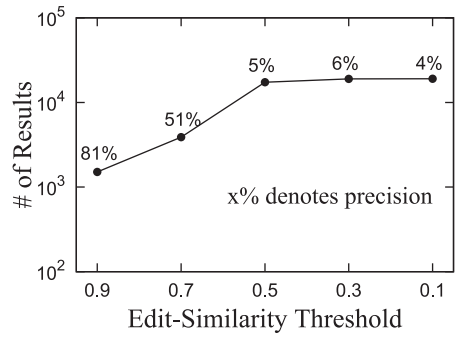


Fig. 12. Evaluating result quality of fuzzy-jaccard for different edit-similarity thresholds ($\tau = 0.7$).

helped to identify almost four times more results than exact-token matching, and there was only a drop of 1% in precision. These experimental results validated the effectiveness of fuzzy-token matching for string matching.

By analyzing the results generated by fuzzy-jaccard similarity and jaccard similarity, we found there are mainly two types of errors in the results.

(1) *Incorrect Fuzzy-Token Matching.* We used edit similarity along with a threshold to decide whether two tokens are fuzzy matching or not. An inappropriate threshold may mistakenly take two different meaning tokens as fuzzy-matching tokens. For example, consider two real queries `boxing com` and `boeing com` in our experiments. If we set the edit-similarity threshold to $\delta = 0.8$, since $\text{NED}(\texttt{boxing}, \texttt{boeing}) = 0.83 \geq 0.8$, `boxing` and `boeing` will be considered as fuzzy matching, thus their edit-similarity value (i.e., 0.83) will be incorrectly incorporated into the fuzzy-jaccard similarity between the two queries. To avoid the problem, we can specify a higher edit-similarity threshold (e.g., $\delta = 0.9$). In this case, since $\text{NED}(\texttt{boxing}, \texttt{boeing}) = 0.83 < 0.9$, `boxing` and `boeing` will not be considered as fuzzy matching.

In order to investigate the effect of edit-similarity threshold on result quality, we fixed the fuzzy-jaccard similarity as $\tau = 0.7$ and varied the edit-similarity threshold from 0.9 to 0.1. We plotted the number of results and the result precision for every edit-similarity threshold as shown in Figure 12. From the figure, we can see that edit-similarity threshold had a significant effect on the result quality, and a small threshold may lead to quite low precision. For example, when the threshold was $\delta = 0.9$, fuzzy-jaccard similarity can achieve 81% precision. If we reduced the threshold to $\delta = 0.5$, although more results were returned, the precision was only 5%. Therefore, it is required to consider edit-similarity threshold when defining fuzzy-token similarity.

(2) *Non-Weighed Tokens.* Fuzzy-token similarity assumes every token has the same weight. But in practice, some tokens may be more important than others. Consider

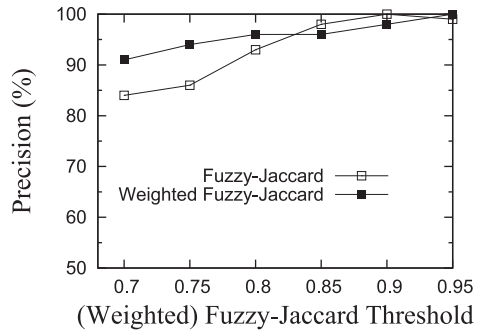


Fig. 13. Comparing result quality of fuzzy-jaccard and weighted fuzzy-jaccard ($\delta = 0.8$).

two real queries `www holiday inn com` and `www edinburgh holiday inn com` in our experiments. They match four tokens and mismatch one token, thus their fuzzy-jaccard similarity is $\frac{4}{4+5-4} = 0.8$. But it is easy to see that the unmatched token `edinburgh` is more important than some other matching tokens (e.g., `www` and `com`), thus it might be better to employ weighted similarity function to quantify their similarity.

To investigate the effect of weights on result quality, we assigned each token t with an IDF weight, computed by $\log \frac{N}{N_t}$, where N is the total number of records and N_t is the total number of records containing t. We fixed edit-similarity threshold $\delta = 0.8$, and varied (weighted) fuzzy-jaccard threshold τ from 0.95 to 0.7. For each τ, we generated results using fuzzy-jaccard and weighted fuzzy-jaccard, respectively, and compared their precision, as shown in Figure 13. We can see both fuzzy-jaccard and weighted fuzzy-jaccard can achieve good precision for large τ. For example, when $\tau = 0.85$, the precision of fuzzy-jaccard and weighted fuzzy-jaccard was 98% and 96%, respectively. For smaller τ (e.g., 0.7), weighted fuzzy-jaccard can still keep the precision larger than 90%, while the precision of fuzzy-jaccard was decreased to 84%. This is because there were a lot of queries with three or four tokens. For such queries, if they mismatch an important token, they are very likely to be dissimilar. In this case, weighted fuzzy-jaccard would give them a much lower similarity value, while fuzzy-jaccard could still give them a higher similarity value than the small threshold (e.g., 0.7).

7.1.2. Comparing with Existing Solutions. In our article, we propose a new similarity function that not only tolerates fuzzy-token matching but also supports efficient similarity-join algorithms. One natural question is whether existing solutions can achieve the same goal. To answer this question, we compare with the following existing similarity functions.

GES and AGES. Two hybrid similarity functions proposed by Chaudhuri et al. [2003]. They have already been described in Section 2.1.

Jaccard+ES. A weighted combination of jaccard similarity and edit similarity, denoted by $\alpha \cdot \text{JACCARD} + (1 - \alpha) \cdot \text{NED}$, where α is a parameter that is used to adjust the weights of jaccard similarity and edit similarity. If we choose a larger α, Jaccard+ES can capture more token swap errors (i.e., `nba mcgrady` and `mcgrady nba`) since jaccard similarity is assigned a larger weight. Otherwise, if we choose a smaller α, Jaccard+ES can capture more edit errors (i.e., `nba mcgrady` and `nba macgrady`). We varied α from 0 to 1 with the step size of 0.1, and found that Jaccard+ES achieved the best performance when $\alpha = 0.5$. Thus, we chose $\alpha = 0.5$ in our experiments.

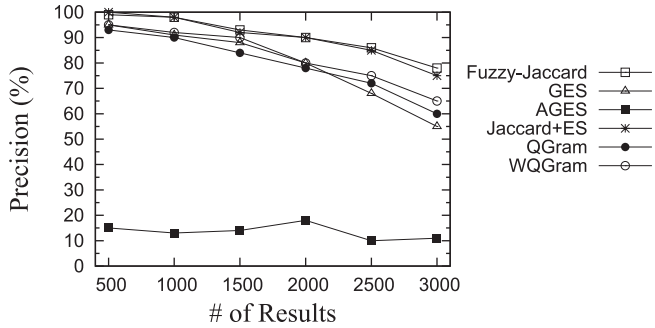


Fig. 14. Comparing the result quality of fuzzy-jaccard similarity and existing similarity functions.

QGram. Denotes jaccard similarity based on q-gram tokenization. In the experiment, we adopted the most common way to generate q-gram sets [Chandel et al. 2007; Hassanzadeh and Miller 2009], and set $q = 2$, since it achieves the best performance.

WQGram. Denotes weighted jaccard similarity based on q-gram tokenization. Each gram was assigned an IDF weight.

We computed similar string pairs using Fuzzy-Jaccard, GES, AGES, Jaccard+ES, and QGram, respectively. For each similarity function, we identified the top-k pairs with the highest similarity values as results. We varied the number of results (i.e., k) from 500 to 3,000, and compared the result precision as shown in Figure 14.

We can see Fuzzy-Jaccard achieved better results than existing hybrid similarity functions, GES and AGES. This is because GES gave a low similarity value to the similar string pairs where the same keywords occurred in different positions, and the closest tokens chosen by AGES may not be real fuzzy-matching tokens. QGram and WQGram did not perform well either,[5] since they typically select a smaller value of q in order to capture typos, but a smaller q may make the strings match a lot of grams that are generated from dissimilar words. For example, consider two real queries `blood test for dog` and `best dog food`. Obviously, they have different meanings. But both QGram and WQGram will compute a high similarity value for them since their 2-gram sets share thirteen common grams, that is, {`$b`, `oo`, `od`, `d$`, `es`, `st`, `t$`, `$f`, `fo`, `$d`, `do`, `og`, `g$`}. But it is easy to see that many grams are generated from dissimilar words. For example, the first gram `$b` is generated from `blood` and `best`.

Figure 14 also shows that Jaccard+ES can achieve as good performance as Fuzzy-Jaccard in terms of precision. But existing works [Wang et al. 2011b; Chaudhuri et al. 2007] have shown that there is no method that can efficiently find similarity-join results for a weighted combination of similarity functions like Jaccard+ES. In order to compare with the proposed similarity-join algorithm for Fuzzy-Jaccard in this article, we implemented the following similarity-join algorithm for Jaccard+ES. Given a threshold τ, we can easily deduce that if the Jaccard+ES similarity between two strings is no smaller than τ, their jaccard similarity must be no smaller than $2\tau - 1$ since their edit similarity is at most equal to one. Based on this idea, we first utilized existing similarity-join methods [Wang et al. 2012] to find a candidate set of pairs whose jaccard similarity is no smaller than $2\tau - 1$, and then check the candidate pairs whether their Jaccard+ES similarity satisfies the threshold τ. We adopted this method to find similar string pairs for $\tau = 0.8, 0.75, 0.7, 0.65$. Figure 15 shows the number of results and the running time for each threshold. For comparison's sake, Figure 15 also shows

[5]The minimum, average and maximum token counts of the picked strings for the quality evaluation of QGram and WQGram are 1, 3.6, and 9, respectively.

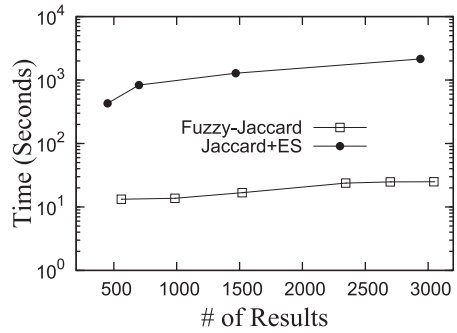


Fig. 15. Comparing the running time of generating results for Fuzzy-Jaccard ($\delta = 0.8$) and Jaccard+ES.

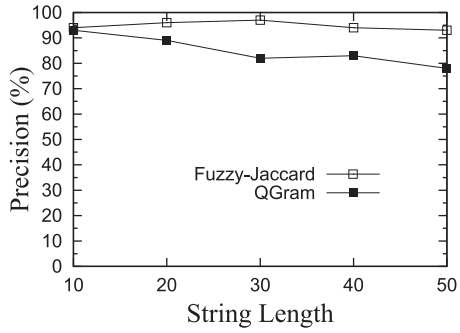


Fig. 16. Comparing the result quality of Fuzzy-Jaccard and QGram for different string lengths ($\tau = 0.8$).

the running time of using the efficient fuzzy-jaccard-similarity join method proposed in this article to generate similar string pairs for $\tau = 0.9, 0.85, 0.8, 0.75, 0.7, 0.65$. From the figure, we can see the time to obtain the results for Fuzzy-Jaccard is 20× to 100× faster than that for Jaccard+ES. Therefore, Fuzzy-Jaccard is more applicable than Jaccard+ES in practice.

We also compared the running time of Fuzzy-Jaccard and QGram and found that QGram is more efficient than Fuzzy-Jaccard. For instance, when the threshold is 0.8, Fuzzy-Jaccard spent 17.01s, while QGram only consumed 3.29s. This is because QGram did not consider whether the matching grams are generated from similar tokens or not. Although it saved running time, it may lead to lower result quality, as discussed in Figure 14. To further evaluate the result quality on the datasets with different string lengths, we constructed five datasets from the Query Log dataset, which consisted of the strings with the lengths smaller than 10, 20, 30, 40, and 50, respectively. Figure 16 compared the result quality of Fuzzy-Jaccard and QGram on the five datasets with the threshold of 0.8. We can see Fuzzy-Jaccard consistently achieved good quality for different string lengths, while QGram can only obtain comparable performance as Fuzzy-Jaccard for the string length smaller than 10. For longer strings, QGram performed much worse than Fuzzy-Jaccard, since longer strings contain more tokens, thus it is more necessary to consider whether the matching grams come from the similar tokens or not.

7.1.3. Comparing with a Simple Fuzzy-Jaccard Similarity-Join Method. Having seen the effectiveness of fuzzy-jaccard similarity for string matching, we next evaluate a simple fuzzy-jaccard similarity-join method, which combines jaccard-similarity prefiltering with fuzzy-jaccard postverification. Specifically, the method first utilizes efficient

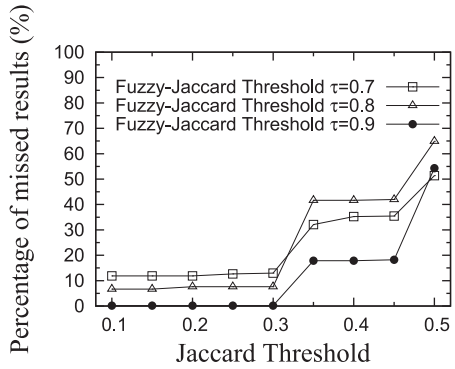


Fig. 17. Evaluating the percentage of missed results for different jaccard thresholds ($\delta = 0.8$).

algorithms to filter the pairs with very low jaccard similarity, and then computes the fuzzy-jaccard similarity of the remaining pairs to obtain the final results. If this method is as good as the proposed fuzzy-jaccard-similarity-join algorithm in this article, we could use only this method instead of inventing new similarity-join algorithms.

Due to jaccard-similarity prefiltering, the simple similarity-join method may miss some results whose jaccard-similarity is smaller than the jaccard threshold, but fuzzy-jaccard similarity is larger than or equal to the fuzzy-jaccard threshold. To evaluate the missed results, we varied the jaccard threshold from 0.1 to 0.5 and computed the percentage of missed results for different fuzzy-jaccard thresholds $\tau = 0.7, 0.8, 0.9$. In Figure 17, we can see the simple similarity-join method missed a lot of results. For example, when the jaccard threshold was 0.4, the simple method would miss about 40% results for $\tau = 0.8$. Although decreasing the jaccard threshold can reduce the number of missed results, the simple similarity-join method would still miss some results even if the threshold was decreased to 0.1 (e.g., 12% results for $\tau = 0.7$).

On the other hand, decreasing the jaccard threshold would also increase the running time, since we need to verify more unfiltered pairs. We compared the running time of the simple similarity-join method and the efficient similarity-join algorithm proposed in our article. Figure 18 shows the comparison results for fuzzy-jaccard thresholds $\tau = 0.7, 0.8, 0.9$. We only plotted one line for the simple similarity-join method, since its performance remained the same for various fuzzy-jaccard thresholds. From the figure, we can see that the simple similarity-join method runs much slower than our efficient fuzzy-jaccard similarity algorithm, with the differences being greater for smaller thresholds. For example, if the threshold was 0.3, the simple method was about ten times slower than our algorithm.

Based on this analysis, the simple similarity-join method would not only miss some results, but also run very inefficiently. This motivates us to study the efficient similarity-join algorithm for fuzzy-jaccard similarity that cannot miss any result. In the following experiments, we will make a thorough analysis on the efficient similarity-join algorithm proposed in this article.

It is worth noting that the datasets (i.e., Query Log and Author) used in this article mainly consist of the strings with a small number of tokens. For some other datasets with much larger number of tokens in each string, we find that it is less necessary for them to consider fuzzy-matching tokens, since the similarity of these strings are typically dominated by the exactly matching tokens. To examine the idea, we constructed a new dataset, denoted by Title, with 100,000 paper titles randomly selected from the DBLP dataset. We compared the simple similarity-join method with our similarity-join algorithm on this new dataset. As shown in Figure 19, the simple similarity-join

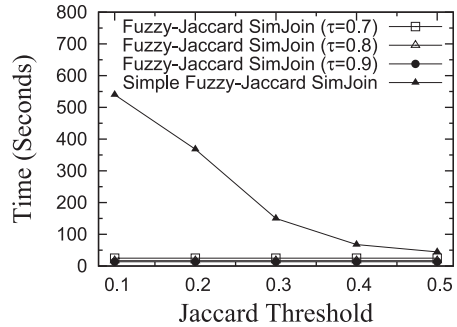


Fig. 18. Comparing the running time of simple similarity-join algorithm and fuzzy-jaccard similarity-join algorithm ($\delta = 0.8$).

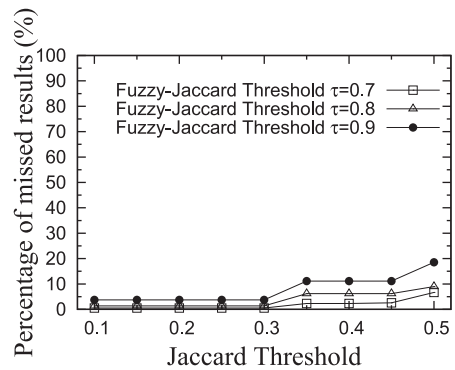


Fig. 19. Evaluating the percentage of missed results for different jaccard thresholds ($\delta = 0.8$, Title dataset).

method still missed some results, but not as many as on the Query Log dataset (see Figure 17). For example, when the jaccard threshold was 0.4, the simple method only missed 6.2% results for $\tau = 0.8$ on the Title dataset, but it missed about 40% results on the Query Log dataset.

7.2. Evaluation on Different Signature Schemes for Tokens

In this section, we compared the performance of different token signature schemes. We implemented five methods: q-gram-based method [Xiao et al. 2008a], deletion-based neighborhood generation [Bocek et al. 2007], Part-Enum [Arasu et al. 2006], Partition-ED [Wang et al. 2009], and Partition-NED. We extended them to support edit similarity using the methods in Section 5.1. We used the token-sensitive signature scheme for generating token sets. Figure 20 gives the results.

We see that the q-gram based method achieved the worst performance, as it can only use small q for short tokens, but small q resulted in large numbers of false-positive results. Part-Enum also performed worse, since converting a token to the feature vector destroyed the position information of grams. The deletion-based neighborhood generation scheme achieved higher performance for the Author dataset, as the tokens are usually short in person names. But for the Query Log dataset, the method generated large numbers of signatures for long tokens and achieved very low performance, and it did not report any result within 10^6 seconds. Thus, in the figure, we did not show the results of the deletion-based neighborhood generation. Partition-NED performed the best of all the signature schemes. When the edit-similarity threshold is large, Partition-ED has the comparable performance with Partition-NED. However when

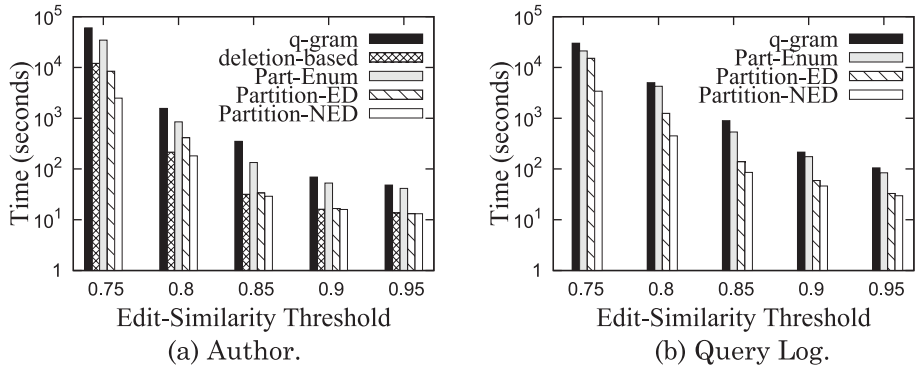


Fig. 20. Performance for different token signature schemes ($\tau = 0.8$).

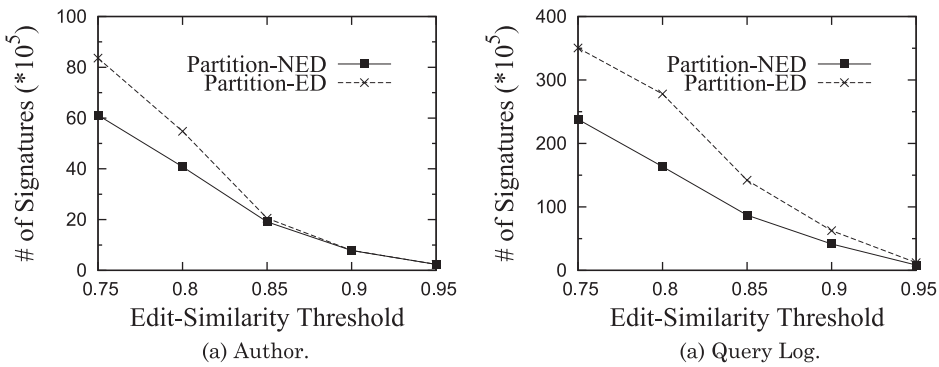


Fig. 21. Comparison of the number of signatures between Partition-ED and Partition-NED ($\tau = 0.8$).

the edit-similarity threshold becomes smaller, Partition-ED will be less efficient than Partition-NED. This is because Partition-ED generated large numbers of signatures, but Partition-NED used the pruning techniques to remove unnecessary signatures.

In addition, we compared the numbers of token signatures generated from Partition-ED and Partition-NED. Figure 21 shows the results. We can see our method can reduce large numbers of signatures. For instance, on the Query Log dataset, for $\delta = 0.8$, Partition-NED generated $2.8 * 10^7$ signatures, while Partition-ED only generated $1.8 * 10^7$ signatures.

As Partition-NED achieved the highest performance, we used Partition-NED for generating token signatures in the remainder experiments of this article.

7.3. Evaluation on Signature Schemes of Token Sets

In this section, we compared the performance of the token-sensitive signature scheme and the prefix-filtering signature scheme. We first compared the number of removed signatures. Figure 22 shows the results.

We can see that the token-sensitive signature scheme can remove many more signatures as it considered token information in the removal step. For example, on the Author dataset, for $\tau = 0.8$, the token-sensitive signature scheme can remove $1.5 * 10^6$ signatures, and the prefix-filtering signature scheme only removed $0.9 * 10^6$ signatures.

We also compared the number of candidates gotten from the two token-set signature schemes. Figure 23 shows the results. We see that the token-sensitive signature scheme generated fewer candidates than the prefix-filtering signature scheme. This is because

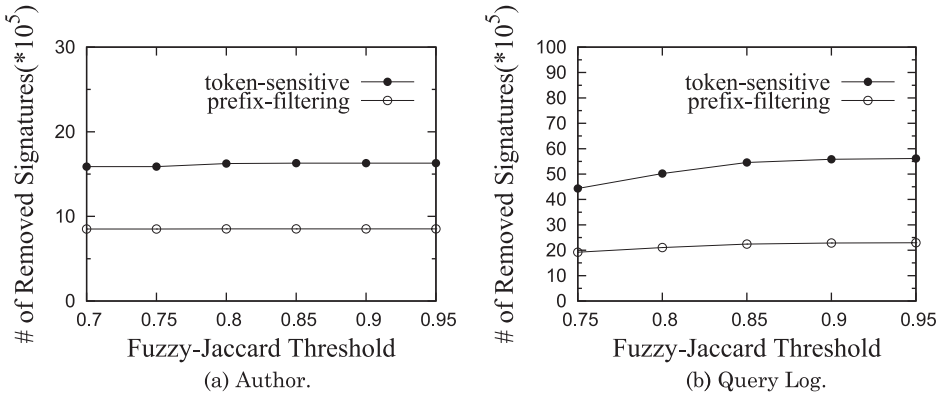


Fig. 22. Comparison of the number of signatures removed by prefix-filtering and token-sensitive signature schemes ($\delta = 0.85$).

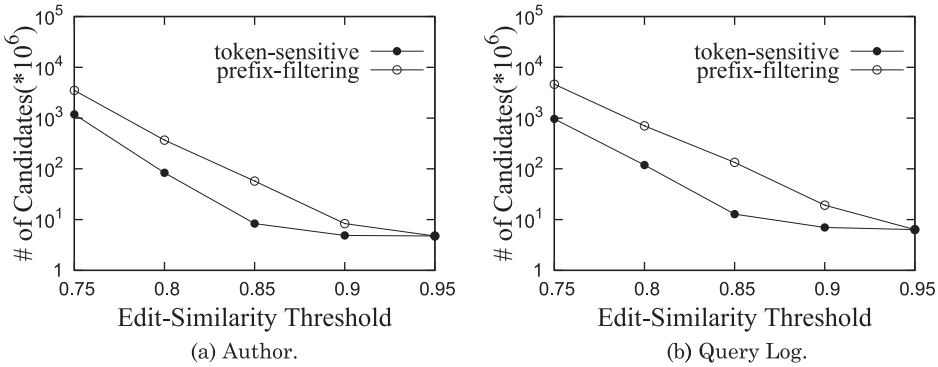


Fig. 23. Comparison of the number of candidates between prefix-filtering and token-sensitive signature schemes ($\tau = 0.8$).

it removed many more unnecessary signatures. For example, on Query Log, for $\delta = 0.85$, the token-sensitive signature scheme generated less than $1.2 * 10^6$ candidates, while the prefix-filtering signature scheme generated $1.3 * 10^7$ candidates.

Finally, we compared the running time of using the two token-set signature schemes to solve the similarity-join problem, and Figure 24 shows results. We can see the algorithm using the token-sensitive signature scheme is 3 to 5 times faster than that using the prefix-filtering signature scheme, as the former can remove large numbers of unnecessary token signatures. For example, on the Author dataset, for $\tau = 0.8$, if using the token-sensitive signature scheme, the algorithm took less than 30s, while if using the prefix-filtering signature scheme, the time increased to 130s.

7.4. Evaluation on Signature Schemes of Weighted Token Sets

In this section, we evaluated the signature schemes of weighted token sets, that is, the weighted-prefix-filtering signature scheme and the weighted-token-sensitive signature scheme. We, respectively, used them to perform Fast-Join on Author and Query Log datasets, and compared their performance in terms of the number of removed signatures, the number of candidates, and the total running time.

In the article, we have proved that the weighted-token-sensitive signature scheme can remove as many or more signatures than the weighted-prefix-filtering signature scheme. To see how the weighted-token-sensitive signature scheme performs in

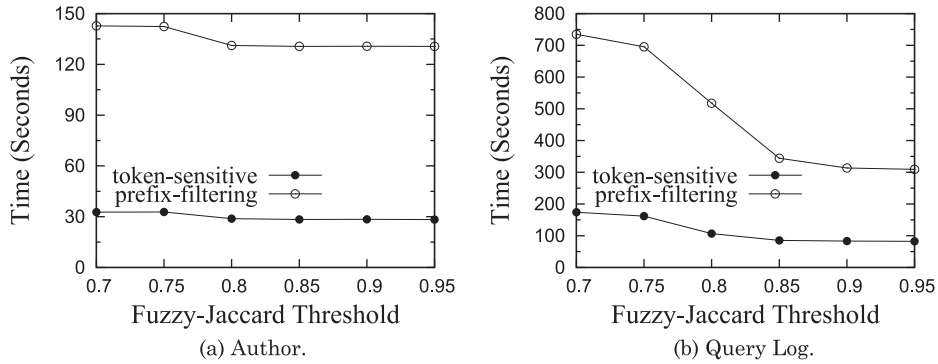


Fig. 24. Comparison of the running time of using prefix-filtering and token-sensitive signature schemes ($\delta = 0.85$).

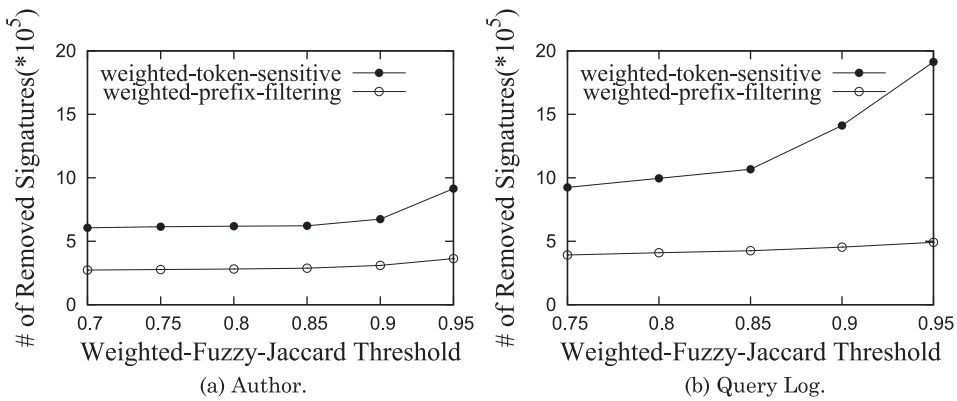


Fig. 25. Comparison of the number of signatures removed by weighted-prefix-filtering and weighted-token-sensitive signature schemes ($\delta = 0.85$).

practice, we compared the number of signatures removed by the two signature schemes on real datasets. Figure 25 shows the results. We can see the weighted-token-sensitive removed 2 to 5 times more signatures than weighted-prefix-filtering. For example, on the Query Log dataset, for $\tau = 0.85$, the weighted-token-sensitive removed $14.1 * 10^5$ signatures but the weighted-prefix-filtering only removed $4.53 * 10^5$ signatures.

Next, we evaluated the number of candidates generated by the two signature schemes. Note that the fewer the numbers of candidates, the less the verification time. As shown in Figure 26, weighted-token-sensitive generated much fewer numbers of candidates than weighted-prefix-filtering. This is because weighted-token-sensitive took into consideration token information and removed more signatures, thus leading to fewer candidates. For example, on the Query Log datasest, for $\delta = 0.85$, weighted-token-sensitive generated $7.2 * 10^5$ candidates, while weighted-prefix-filtering generated $67.8 * 10^5$ candidates.

Finally, we measured the running time of performing Fast-Join by using the two signature schemes. Figure 27 shows the results. We can see the algorithm using weighted-token-sensitive is 2 to 3 times faster than that using weighted-prefix-filtering, as the former can remove large numbers of signatures. For example, on the Author dataset, for $\tau = 0.8$, if using weighted-token-sensitive, the algorithm took less than 3.5s, while if using weighted-prefix-filtering, the time increased to 10.6s.

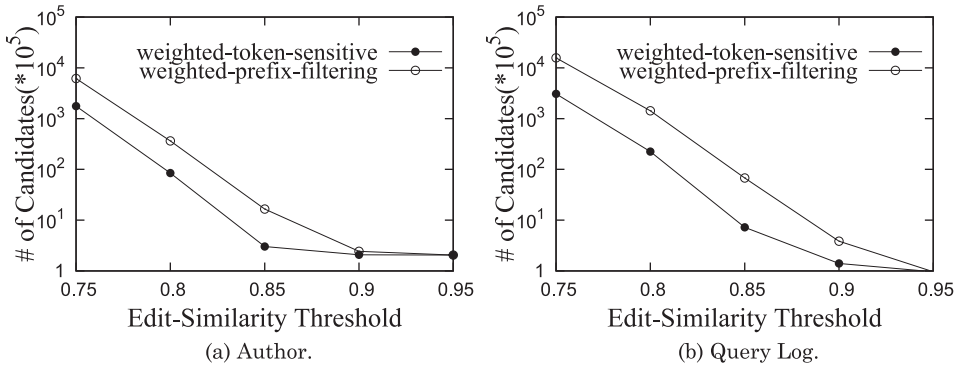


Fig. 26. Comparison of the number of candidates between weighted-prefix-filtering and weighted-token-sensitive signature schemes ($\tau = 0.8$).

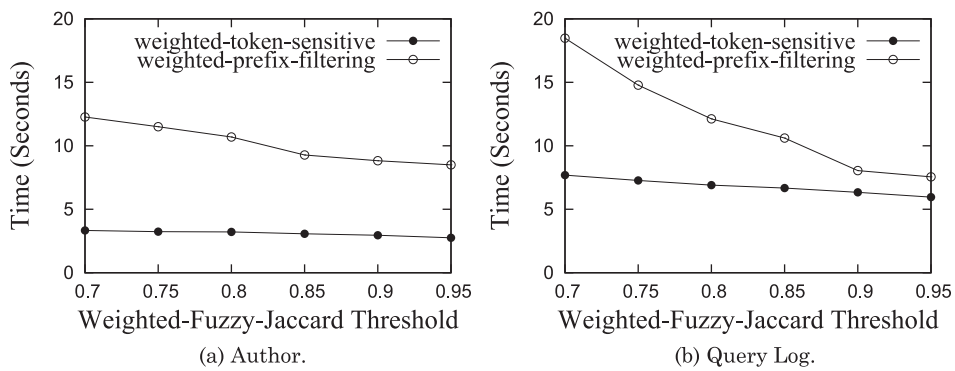


Fig. 27. Comparison of the running time of using weighted-prefix-filtering and weighted-token-sensitive signature schemes ($\delta = 0.85$).

7.5. Put Everything Together

In this section, we further evaluated the algorithm of solving the similarity-join problem, which included three phases: (1) generating signatures; (2) filtering dissimilar pairs and computing candidates; (3) verifying the candidates to get the final results. We used a token-sensitive signature scheme for token sets and Partition-NED for token signatures. Figure 28 shows the results by varying the fuzzy-jaccard threshold τ.

For the Author dataset, the three phases took a similar amount of time. For the Query Log dataset, the phase of generating signatures was rather expensive. This is because in the dataset, the tokens have larger length, which resulted in larger edit-distance thresholds. When τ became smaller the filter and the verification time increased. The reason being that a smaller τ will result in more candidate pairs. It is worth noting that we do not need to perform weighted bigraph matching for every candidate. As shown in Section 3.2, we can compute an upper bound of fuzzy-token similarity; if the upper bound is smaller than the threshold, we can prune the candidate.

7.6. Evaluation on Other Similarity Functions

We evaluated the performance of different fuzzy-token similarities, fuzzy-jaccard similarity, fuzzy-dice similarity, and fuzzy-cosine similarity. Figure 29 shows the results. We see that fuzzy-dice similarity and fuzzy-cosine similarity took more time

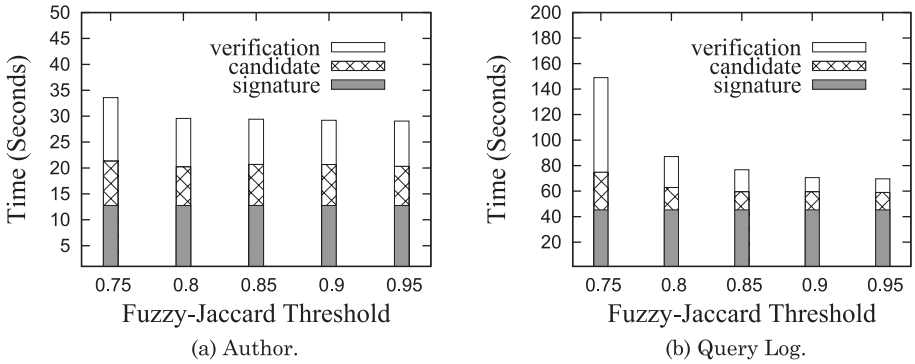


Fig. 28. Performance for different steps ($\delta = 0.85$).

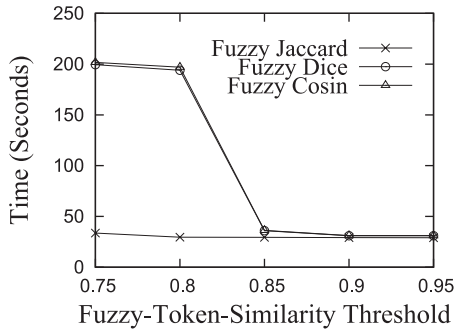


Fig. 29. Performance for different functions on the Author dataset ($\delta = 0.85$).

than fuzzy-jaccard similarity. This is because for the same τ, they deduced a smaller fuzzy-overlap threshold than fuzzy-jaccard similarity. We also evaluate the result quality of the three similarities. We find that when fixing the same thresholds δ and τ, fuzzy-jaccard similarity archived higher precision but returned fewer relevant pairs than the other two similarities. For example, when $\delta = 0.85$ and $\tau = 0.8$, fuzzy-jaccard similarity returned 1,029 relevant pairs with precision 95%, while fuzzy-dice similarity returned 3,298 pairs with precision 71% and fuzzy-dice similarity returned 3,324 pairs with precision 70%.

8. RELATED WORKS

String similarity join is a basic operation in data cleaning and integration and has attracted significant attention recently [Gravano et al. 2001; Sarawagi and Kirpal 2004; Arasu et al. 2006; Bayardo et al. 2007; Xiao et al. 2008b, 2008a; Wang et al. 2010, 2012; Vernica et al. 2010; Zhang et al. 2010; Qin et al. 2011; Li et al. 2011b]. Most of the prior works use signature-based methods and focus on the development of an effective signature scheme. Gravano et al. [2001] devised a gram-based signature scheme for edit similarity join. Xiao et al. [2008b] optimized this method by pruning unnecessary grams. Qin et al. [2011] proposed a hybrid gram and chunk signature scheme that is proved to be able to generate the minimum number of signatures. Chaudhuri et al. [2006] proposed a prefix-filtering signature scheme that is applicable to a variety of similarity functions. Bayardo et al. [2007] developed several optimization techniques to make the prefix-filtering signature scheme scale to large datasets.

Wang et al. [2012] found different prefix lengths lead to different performance and proposed a cost model to adaptively select an appropriate prefix. There are also some studies on partition-based signature schemes [Arasu et al. 2006; Li et al. 2011b] which mainly focus on hamming distance and edit distance, respectively. However, these signature schemes are not developed specific to the fuzzy-token similarity. Although the prefix-filtering signature scheme can be extended to the fuzzy-token similarity, it was quite expensive and generated a large number of candidates. Therefore, we proposed the token-sensitive signature scheme which is proved to be better than the prefix-filtering signature scheme. In the experiment, we have extensively compared the two signature schemes. The experimental results also proved our claim. In addition to the signature-based methods, Jacox and Samet [2008] studied the metric-space similarity join. The method cannot solve our problem, since fuzzy-token similarity does not obey the triangle inequality.

There are some studies on fuzzy-token-matching-based similarity. Chaudhuri et al. [2003] proposed generalized edit similarity (GES) and an approximation of generalized edit similarity (AGES), which extends the character-level edit operator to the token-level edit operator. Arasu et al. [2008] proposed a transformation-based framework for similarity join by considering user-defined string transformations, such as synonyms and abbreviations. They employed a signature-based method and defined the signature scheme as the union of string transformations related to each string. When applying the signature scheme to our problem, as discussed in Section 4.2, it is less effective than the token-sensitive signature scheme, since it ignored the fact that some signatures can be removed. Jestes et al. [2010] studied probabilistic string similarity joins with expected edit distance constrains. The method needs some extra inputs such as probabilistic string attributes, while Fast-Join needs little human effort, and thus is an application-independent method to combine two types of similarity measures. More importantly, our similarity can subsume existing ones. A big benefit of our method is that it can be easily extended to support existing similarity functions.

Weighted similarity was widely used in real applications, and many existing works explored how to make their approaches support weighted similarity [Arasu et al. 2006; Bayardo et al. 2007; Hadjieleftheriou and Srivastava 2010]. However, existing weighted similarity is defined without considering the fuzzy matching tokens. Therefore, we extend our earlier conference paper [Wang et al. 2011a] to support weighted token similarity and make the following significant contributions.

—We proposed weighted fuzzy-token similarity to quantify the fuzzy-token similarity of weighted token sets.
—To support weighted fuzzy-token similarity, we proposed effective signature schemes for weighted token sets, that is, weighted-prefix-filtering signature scheme and weighted-token-sensitive signature scheme.
—We conducted new experiments to evaluate our newly proposed signature schemes, and compared with more existing similarity functions on result quality.
—We formally proved the correctness of our techniques in all lemmas and theorems.

Some other related fields are approximate string searching [Kim et al. 2005; Li et al. 2007, 2008; Hadjieleftheriou et al. 2008, 2009; Lee et al. 2007; Zhang et al. 2010; Deng et al. 2013; Kim et al. 2005], which given a query string and a set of strings, finds all similar strings of the query string in the string set, and approximate entity extraction [Wang et al. 2009; Chakrabarti et al. 2008; Agrawal et al. 2008; Lu et al. 2009; Chaudhuri et al. 2009; Li et al. 2011a], which given a document and a set of strings, extracts all substrings from the document that are similar with the strings in the string set.

9. CONCLUSION

In this article, we have studied the problem of string similarity join. We proposed a new similarity function by combing token-based similarity and character-based similarity. We proved that existing similarities are special cases of fuzzy-token similarity. We proposed a signature-based framework to address the similarity join using fuzzy-token similarity. We proposed token-sensitive signature scheme, which is superior to the state-of-the-art signature scheme. We extended existing signature schemes for edit distance to support edit similarity. We devised a partition-based token signature scheme and developed pruning techniques to improve the performance. We also extended fuzzy-token similarity to support weighted tokens, and developed effective signature schemes, that is, weighted-prefix-filtering signature scheme and weighted-token-sensitive signature scheme, to improve the performance. The experimental results on real datasets show that our method achieves high result quality and performance.

REFERENCES

Sanjay Agrawal, Kaushik Chakrabarti, Surajit Chaudhuri, and Venkatesh Ganti. 2008. Scalable ad-hoc entity extraction from text collections. *Proc. VLDB Endow.* 1, 1, 945–957.

Arvind Arasu, Surajit Chaudhuri, and Raghav Kaushik. 2008. Transformation-based framework for record matching. In *Proceedings of the IEEE 24th International Conference on Data Engineering (ICDE)*. 40–49.

Arvind Arasu, Venkatesh Ganti, and Raghav Kaushik. 2006. Efficient exact set-similarity joins. In *Proceedings of the 32nd International Conference on VLDB (VLDB)*. 918–929.

Ricardo A. Baeza-Yates and Berthier A. Ribeiro-Neto. 1999. *Modern Information Retrieval*. ACM Press/ Addison-Wesley.

Roberto J. Bayardo, Yiming Ma, and Ramakrishnan Srikant. 2007. Scaling up all pairs similarity search. In *Proceedings of the 16th International Conference on WWW (WWW)*. 131–140.

Dimitri P. Bertsekas. 1993. A simple and fast label correcting algorithm for shortest paths. *Netw.* 23, 7, 703–709.

Kaushik Chakrabarti, Surajit Chaudhuri, Venkatesh Ganti, and Dong Xin. 2008. An efficient filter for approximate membership checking. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*. 805–818.

Amit Chandel, Oktie Hassanzadeh, Nick Koudas, Mohammad Sadoghi, and Divesh Srivastava. 2007. Benchmarking declarative approximate selection predicates. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*. 353–364.

Surajit Chaudhuri, Bee-Chung Chen, Venkatesh Ganti, and Raghav Kaushik. 2007. Example-driven design of efficient record matching queries. In *Proceedings of the 33rd International Conference on VLDB (VLDB)*. 327–338.

Surajit Chaudhuri, Kris Ganjam, Venkatesh Ganti, and Rajeev Motwani. 2003. Robust and efficient fuzzy match for online data cleaning. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*. 313–324.

Surajit Chaudhuri, Venkatesh Ganti, and Raghav Kaushik. 2006. A Primitive operator for similarity joins in data cleaning. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE)*. 5–16.

Surajit Chaudhuri, Venkatesh Ganti, and Dong Xin. 2009. Mining document collections to facilitate accurate approximate entity matching. *Proc. VLDB Endow.* 2, 1, 395–406.

William W. Cohen, Pradeep D. Ravikumar, and Stephen E. Fienberg. 2003. A comparison of string distance metrics for name-matching tasks. In *Proceedings of the JCAI-03 Workshop on Information Integration*. 73–78.

Dong Deng, Guoliang Li, Jianhua Feng, and Wen-Syan Li. 2013. Top-k string similarity search with edit-distance constraints. In *Proceedings of the 29th International Conference on Data Engineering (ICDE)*. 925–936.

Luis Gravano, Panagiotis G. Ipeirotis, H. V. Jagadish, Nick Koudas, S. Muthukrishnan, and Divesh Srivastava. 2001. Approximate string joins in a database (almost) for free. In *Proceedings of the 27th International Conference on VLDB (VLDB)*. 491–500.

Marios Hadjieleftheriou, Amit Chandel, Nick Koudas, and Divesh Srivastava. 2008. Fast indexes and algorithms for set similarity selection queries. In *Proceedings of the 24th International Conference on Data Engineering (ICDE)*. 267–276.

Marios Hadjieleftheriou, Nick Koudas, and Divesh Srivastava. 2009. Incremental maintenance of length normalized indexes for approximate string matching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*. 429–440.

Marios Hadjieleftheriou and Divesh Srivastava. 2010. Weighted set-based string similarity. *IEEE Data Eng. Bull.* 33, 1, 25–36.

Oktie Hassanzadeh and Renée J. Miller. 2009. Creating probabilistic databases from duplicated data. *VLDB J.* 18, 5, 1141–1166.

Edwin H. Jacox and Hanan Samet. 2008. Metric space similarity joins. *ACM Trans. Database Syst.* 33, 2 (2008).

Jeffrey Jestes, Feifei Li, Zhepeng Yan, and Ke Yi. 2010. Probabilistic string similarity joins. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*. 327–338.

Min-Soo Kim, Kyu-Young Whang, Jae-Gil Lee, and Min-Jae Lee. 2005. n-Gram/2L: A space and time efficient two-level n-gram inverted index structure. In *Proceedings of the 31st International Conference on VLDB (VLDB)*. 325–336.

Hongrae Lee, Raymond T. Ng, and Kyuseok Shim. 2007. Extending q-grams to estimate selectivity of string matching with low edit distance. In *Proceedings of the 33rd International Conference on VLDB (VLDB)*. 195–206.

Chen Li, Jiaheng Lu, and Yiming Lu. 2008. Efficient merging and filtering algorithms for approximate string searches. In *Proceedings of the 24th International Conference on Data Engineering (ICDE)*. 257–266

Chen Li, Bin Wang, and Xiaochun Yang. 2007. VGRAM: Improving performance of approximate queries on string collections using variable-length grams. In *Proceedings of the 33rd International Conference on (VLDB)*. 303–314.

Guoliang Li, Dong Deng, and Jianhua Feng. 2011a. Faerie: Efficient filtering algorithms for approximate dictionary-based entity extraction. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*. 529–540.

Guoliang Li, Dong Deng, Jiannan Wang, and Jianhua Feng. 2011b. Pass-Join: A partition-based method for similarity joins. *Proc. VLDB Endow.* 5, 3 (2011), 253–264.

Jiaheng Lu, Jialong Han, and Xiaofeng Meng. 2009. Efficient algorithms for approximate member extraction using signature-based inverted lists. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM)*. 315–324.

Jianbin Qin, Wei Wang, Yifei Lu, Chuan Xiao, and Xuemin Lin. 2011. Efficient exact edit similarity query processing with the asymmetric signature scheme. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*. 1033–1044.

Sunita Sarawagi and Alok Kirpal. 2004. Efficient set joins on similarity predicates. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*. 743–754.

T. Bocek, E. Hunt, and B. Stiller. 2007. Fast similarity search in large dictionaries. Tech. rep. ifi-2007.02, Department of Informatics, University of Zurich. http://fastss.csg.uzh.ch/.

Rares Vernica, Michael J. Carey, and Chen Li. 2010. Efficient parallel set-similarity joins using mapreduce. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*. 495–506.

Jiannan Wang, Guoliang Li, and Jianhua Feng. 2010. Trie-Join: Efficient trie-based string similarity joins with edit-distance constraints. *Proc. VLDB Endow.* 3, 1 (2010), 1219–1230.

Jiannan Wang, Guoliang Li, and Jianhua Feng. 2011. Fast-join: An efficient method for fuzzy token matching based string similarity join. In *Proceedings of the 27th International Conference on Data Engineering (ICDE)*. 458–469.

Jiannan Wang, Guoliang Li, and Jianhua Feng. 2012. Can we beat the prefix filtering? An adaptive framework for similarity join and search. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*. 85–96.

Jiannan Wang, Guoliang Li, Jeffrey Xu Yu, and Jianhua Feng. 2011b. Entity matching: how similar is similar. *Proc. VLDB Endow.* 4, 10 (2011), 622–633.

Wei Wang, Chuan Xiao, Xuemin Lin, and Chengqi Zhang. 2009. Efficient approximate entity extraction with edit distance constraints. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*. 759–770.

Chuan Xiao, Wei Wang, and Xuemin Lin. 2008a. Ed-Join: An efficient algorithm for similarity joins with edit distance constraints. *Proc. VLDB Endow.* 1, 1 (2008), 933–944.

Chuan Xiao, Wei Wang, Xuemin Lin, and Haichuan Shang. 2009. Top-k set similarity joins. In *Proceedings of the International Conferenc on Data Engineering (ICDE)*. 916–927.

Chuan Xiao, Wei Wang, Xuemin Lin, and Jeffrey Xu Yu. 2008b. Efficient similarity joins for near duplicate detection. In *Proceedings of the 17th International Conference on WWW (WWW)*. 131–140.

Zhenjie Zhang, Marios Hadjieleftheriou, Beng Chin Ooi, and Divesh Srivastava. 2010. Bed-tree: An all-purpose index structure for string similarity search based on edit distance. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*. 915–926.

Received August 2012; revised March, July, October 2013; accepted October 2013