Boosting the Quality of Approximate String Matching by Synonyms

Lu, Jiaheng

2015

Lu , J , Lin , C , Wang , W , Li , C & Xiao , X 2015 , ' Boosting the Quality of Approximate String Matching by Synonyms ' , ACM Transactions on Database Systems , vol. 40 , no. 3 , 15 . https://doi.org/10.1145/2818177

http://hdl.handle.net/10138/158483
https://doi.org/10.1145/2818177

publishedVersion

Boosting the Quality of Approximate String Matching by Synonyms

JIAHENG LU, Renmin University of China and University of Helsinki
CHUNBIN LIN, University of California, San Diego
WEI WANG, University of New South Wales
CHEN LI, University of California, Irvine
XIAOKUI XIAO, Nanyang Technological University

A string-similarity measure quantifies the similarity between two text strings for approximate string matching or comparison. For example, the strings "Sam" and "Samuel" can be considered to be similar. Most existing work that computes the similarity of two strings only considers syntactic similarities, for example, number of common words or q-grams. While this is indeed an indicator of similarity, there are many important cases where syntactically-different strings can represent the same real-world object. For example, "Bill" is a short form of "William," and "Database Management Systems" can be abbreviated as "DBMS." Given a collection of predefined synonyms, the purpose of this article is to explore such existing knowledge to effectively evaluate the similarity between two strings and efficiently perform similarity searches and joins, thereby boosting the quality of approximate string matching.

In particular, we first present an expansion-based framework to measure string similarities efficiently while considering synonyms. We then study efficient algorithms for similarity searches and joins by proposing two novel indexes, called SI-trees and QP-trees, which combine signature-filtering and length-filtering strategies. In order to improve the efficiency of our algorithms, we develop an estimator to estimate the size of candidates to enable an online selection of signature filters. This estimator provides strong low-error, high-confidence guarantees while requiring only logarithmic space and time costs, thus making our method attractive both in theory and in practice. Finally, the experimental results from a comprehensive study of the algorithms with three real datasets verify the effectiveness and efficiency of our approaches.

Categories and Subject Descriptors: H.2.4 [**Database Management**]: Systems—*Query processing*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Search process*

General Terms: Algorithms, Experimentation, Performance, Theory

Additional Key Words and Phrases: String similarity search, similarity join, semantic search

ACM Reference Format:
Jiaheng Lu, Chunbin Lin, Wei Wang, Chen Li, and Xiaokui Xiao. 2015. Boosting the quality of approximate string matching by synonyms. ACM Trans. Datab. Syst. 40, 3, Article 15 (October 2015), 42 pages.
DOI: http://dx.doi.org/10.1145/2818177

This research is partially supported by the 973 Program of China (Project No. 2012CB316205), NSF China (No: 61472427), and a research fund from the University of Helsinki.
Authors' addresses: J. Lu (corresponding author), Department of Computer Science, University of Helsinki, Finland, FI-00014; email: jiaheng.lu@helsinki.fi; jiahenglu@gmail.com; C. Lin, University of California, San Diego, La Jolla, CA 92093; email: chunbinlin@cs.ucsd.edu; W. Wang, School of Computer Science and Engineering, University of New South Wales, Australia, 2052; email: weiw@cse.unsw.edu.au; C. Li, Department of Computer Science, University of California, Irvine, CA 92697; email: chenli@ics.uci.edu; X. Xiao, School of Computer Science and Engineering, Nanyang Technological University, Singapore, 639798; email: xkxiao@ntu.edu.sg.

DOI: http://dx.doi.org/10.1145/2818177

1. INTRODUCTION

Approximate string matching is used heavily in data integration [Huang and Madey 2004; Arasu et al. 2006, 2008], data deduplication [Chaudhuri et al. 2006; Christen 2012], and data search [Chaudhuri and Kaushik 2009]. Most existing work that computes the similarity of two strings only considers syntactic similarities, for example, number of common words or q-grams. Although this is indeed an indicator of similarity, there are many important cases where syntactically-different strings can represent the same real-world object. For example, "Big Apple" is a synonym of "New York," and "Transactions on Database Systems" can be abbreviated as "TODS." This equivalence information can help us identify semantically-similar strings that may have been missed by syntactical similarity-based approaches.

In this article, we study three problems related to approximate string matching with synonyms.[1] First, how to define an effective *similarity measure* between two strings based on the synonyms? Traditional similarity functions cannot be easily extended to handle the synonyms in similarity matching. Second, how to perform *string similarity search* efficiently? That is, how to efficiently find all the similar strings from a given collection of strings to a string query. Finally, how to efficiently perform *similarity join* based on newly-defined similarity functions? That is, given two collections of strings, how to efficiently find similar string pairs from the collections? These three problems are closely related, because the similarity measures defined in the first problem will naturally provide a search and join predicate to the second and third problems.

We give several applications in order to shed light on the importance of these three problems. In a gene/protein database, one of the major obstacles that hinders effective use is term variation [Tsuruoka et al. 2007], including Roman-Arabic (e.g., "Synapsin 3" and "Synapsin III"), acronym variation (e.g., "IL-2" and "Interleukin-2"), and term abbreviation (e.g., "Ah receptor" and "Ah dioxin receptor"). A new similarity measure that can handle term variation may help users discover more genes (proteins) of interest. New measures are also applicable in information-extraction systems to aid mapping from text strings to canonical entries by similarity matching. More applications can be found in areas such as term classification and term clustering, which can be used for advanced information retrieval/extraction applications.

String approximate searches and joins are also useful in data integration and data cleaning. For example, in two representative data-cleaning domains, namely, addresses and academic publications, there often exist rich sources of synonyms which can be leveraged to improve the effectiveness of systems significantly. In addition, information from different data sources often have various inconsistencies. The same real-world entity could be represented in slightly different formats. Therefore, data cleaning needs to find from a collection of entities those similar to a given entity. A typical query is "*find records related to Harvard University*," and an entity of "*1350 Massachusetts Ave, Cambridge, MA 02138*" should be found and returned.

For the first problem of similarity measures, there is a wealth of research on string-similarity functions on the syntactical level, such as Levenshtein distance [Wang et al. 2012], Hamming distance [Kondrak 2005], episode distance [Cohen et al. 2003], cosine metric [Salton and Buckley 1988], Jaccard coefficient [Chaudhuri et al. 2006; Li et al. 2008], and dice similarity [Bayardo et al. 2007]. Unfortunately, none of them can be trivially extended to leverage synonyms to improve the quality of similarity measurement. One exception is the *JaccT* technique proposed by Arasu et al. [2008], which generates all possible strings by applying synonym rewriting to both strings, returning

[1]For brevity, we use "*synonym*" to describe any kind of equivalent pairs which may include *synonym*, *acronym*, *abbreviation*, *variation*, and other equivalent expressions.

ID	Name
q_1	Intl WH Conf 2012
q_2	VLDB
...	...

(a) Table Q.

ID	Name
t_1	Very Large Scale Integeration
t_2	Intl Wireless Health Conference 2012 UK
...	...

(b) Table T.

ID	Synonym pairs	ID	Synonym pairs
r_1	WH → Wireless Health	r_2	Conference → Conf
r_3	Intl → International	r_4	UK → United Kingdom
r_5	Wireless Health → WH	r_6	Conf → Conference
...	...	...	...

(c) Transformation synonym pairs.

Fig. 1. An example to illustrate similarity matching with synonyms.

the maximum similarity among the transformed pairs of strings, as illustrated by the following example.

Example 1.1. Figure 1 shows an example of two tables Q and T and a set of synonyms. Consider two strings q_1 and t_2. Their token-based Jaccard similarity is only $\frac{2}{8}$. Now consider applying synonym pairs on them. Obviously, synonyms r_1, r_3, and r_6 can be applied to q_1, and r_2, r_3, r_4, and r_5 can be applied to t_2. We use $\mathcal{A}_{\mathcal{R}}(s)$ to denote the string generated by applying a set of synonym pairs $\mathcal{R}$ to the source string s. For example, $\mathcal{A}_{\{r_1,r_3\}}(q_1)$ is "`International Wireless Health Conf 2012`."

Arasu et al. [2008] considers all possible strings resulting from applying a subset of applicable synonyms on them and then returns the maximum Jaccard similarity. In the preceding example, it needs to consider $2^3 \cdot 2^4 = 128$ possible string pairs. The maximum Jaccard value between $\mathcal{A}_{\{r_1,r_3,r_6\}}(q_1)$ and $\mathcal{A}_{\{r_3\}}(t_2)$, is $\frac{5}{6}$.

The main limitation in the preceding approach is that its string-similarity definition has to generate all possible strings and compare their similarity for each pair in the cross product. In general, given two strings s and t, assuming there are n_1 (resp. n_2) synonym pairs applied on s (resp. t), then the preceding approach needs to compute the similarities for $2^{n_1+n_2}$ pairs. Such a brute-force approach takes time exponential in the number of applicable synonym pairs. A special case of the problem for single-token synonyms is identified in Arasu et al. [2008], where the problem can be reduced to a maximum bipartite graph matching-based problem. Unfortunately, in practice, many synonyms contain multiple tokens, and even in the special case, it still has a worst-case complexity that is quadratic in both the number of string tokens and the number of applicable synonyms. Since such similarity measure has to be called for every candidate pairs when employed in similarity joins, it introduces substantial overhead.

In this article, we propose a novel *expansion-based* framework to measure the similarity of strings. Intuitively, given a string s, we first obtain its token set S and then grow the token set by applying the applicable synonyms of s. When a synonym pair $lhs \rightarrow rhs$ is applied, we merge all the tokens in rhs to the token set, that is, $S = S \cup rhs$. The similarity between two strings is the similarity between their expanded token sets. We have two key ideas here: (i) We can avoid the exponential computational complexity by expanding strings with all applicable synonym pairs; (ii) it costs less to perform expanding operation than transforming operation, as illustrated next.

Example 1.2 (*Continue the Previous Example*). We apply all applicable synonym pairs to q_1 and t_2 to obtain the "closure" of their tokens and then evaluate the Jaccard similarity between the two expanded token sets. For instance, the fully-expanded set

Similarity Algorithm	Computational Complexity	Effectiveness	Comment
Jacc	$O(L)$	*Low*	*Does not consider synonym pairs.*
JaccT [2]	$O(2^n(L+kn))$	*High*	*If only single token synonym pairs are allowed (implying $k = 1$), an optimal algorithm based on maximum bipartite matching can be used with complexity $O(L^2 + Ln^2)$.*
Full Expansion (ours)	$O(L+kn)$	*High*	*Most efficient and may have good quality.*
SE Algorithm (ours)	$O(L+kn^2)$	*High*	*SE is efficient and has good quality; it is optimal under the conditions of Theorem 2, and the complexity can be reduced to $O(L+n\log n+kn)$.*

Fig. 2. Comparison of string-similarity algorithms with synonyms (assume both strings have length L; the maximum number of applicable synonyms is n; k is the maximum size of the right-hand side of a rule).

of t_2 by applying synonym pairs r_2, r_3, r_4, and r_5 is {`International`, `Intl`, `WH`, `Conf`, `2012`, `Wireless`, `Health`,`Conference`, `United`, `Kingdom`, `UK`}. The Jaccard similarity between two fully-expanded token sets is $\frac{8}{11}$, which is also greater than the original similarity $\frac{2}{8}$ without synonyms.

We note that the full expansion-based similarity illustrated in the preceding example is efficient to compute and is also highly competitive in terms of effectiveness in modeling the semantic similarity between strings. Nonetheless, using all the synonym pairs to perform a full expansion may hurt the final similarities. Consider Example 1.2 again, where the use of rule r_4 on t_2 is detrimental to the similarity between q_1 and t_2, as none of the expanded tokens appear in q_1 or its expanded token set. Therefore, we propose a *selective expansion* measure which selects only "*good*" synonym pairs to expand both strings (Section 3), and accordingly, the similarity is defined as the maximum similarity between two appropriately-expanded token sets of two strings. Unfortunately, selective expansion is proved to be NP-hard by a reduction from the 3SAT problem. To provide an efficient solution, we propose a greedy algorithm, called *SE* algorithm, which is optimal if the *rhs* tokens of the useful synonyms satisfy the *distinct* property (the precise definition will be described in Section 3.3). We empirically show that the condition is likely to be true in practice. Figure 2 summarizes the theoretical analysis of the two new similarity algorithms, contrasted with results of existing methods [Arasu et al. 2008].

Given the newly-proposed similarity measures, it is imperative to study efficient similarity search and join algorithms. For string-similarity search, we first show a signature-based search algorithm, called *search-baseline*, by following the *filter-and-verification* framework. It generates a set of tokens as signatures for each string in the table, and then for the query string, finds candidate pairs that overlap in the signatures, and finally verifies against the threshold using the previously proposed similarity measures. Further, we propose *SI-search*, which speeds up the performance by using native indexes, *SI-tree* for strings from the table, and *QP-tree* for the query string, to reduce the candidate size with integrated signatures and length filters (Section 4). Similarly, for string similarity joins, we also propose two algorithms, one is the signature-based baseline method (*join-baseline*), and the other is the SI-tree-based index join algorithm (*SI-join*). We demonstrate that our join algorithms are flexible and may use different signatures, including prefix filtering and locality sensitive hashing (LSH).

The methods for selecting signatures of strings may significantly affect the performance of algorithms [Li et al. 2008; Qin et al. 2011]. Therefore, we propose an online algorithm to dynamically estimate the filtering power of a signature filter. In particular,

we generate multiple signature filters using different strategies offline. Then, given a query to search online or two tables to join online, we quickly estimate the performance of different filters and select the best one to perform the actual filtering.

Our technical contribution here is to propose a hash-based synopsis data structure, termed *Two-Dimensional Hash Sketch* (2DHS), by extending the Flajolet-Martin sketch [Flajolet and Martin 1985] on two join tables. We present novel estimation algorithms to provide high-confidence estimates for comparing the filtering power of two filters. Our method carries both theoretical and practical significance. In theory, we can prove that, with any given high probability 1-δ, 2DHS returns correct estimates on upper and lower bounds, and its space and time complexities are logarithmic in the cardinality of input size. In practice, the experiments show that our estimation technique accurately predicts the quality of different filters in all three datasets and that its running time is negligible compared to that of the actual filtering phase (accounting for only 1% of the total join time).

Finally, we perform a comprehensive set of experiments on three datasets to demonstrate the superior efficiency and effectiveness of our algorithms (Section 8). The results show that our algorithms are up to two orders of magnitude more efficient than the existing methods while offering comparable or better effectiveness.

Organization. The rest of this article is organized as follows. We review related works in Section 2 and propose similarity measures in Section 3. In Section 4 and Section 5, we study the approximate string search and approximate string join problems, respectively. We present the high-confidence and low-error estimator in Section 6. Section 7 is dedicated to the extensions of our algorithms for more similarity measures and the weighted tokens. Then Section 8 empirically evaluates the effectiveness and efficiency of all proposed algorithms and the state-of-the-art approaches. Finally, Section 9 concludes this article and sheds light on future work.

2. RELATED WORK

String Similarity Measures. There is a rich set of string-similarity measures, including character *n*-gram similarity [Kondrak 2005], Levenshtein distance [Xiao et al. 2008; Wang et al. 2012], Jaro-Winkler measure [Winkler 1999], Jaccard similarity [Chaudhuri et al. 2006], TF-IDF-based cosine similarity [Salton and Buckley 1988], and hidden Markov model-based measure [Miller et al. 1999]. A comparison of many string-similarity functions is performed in Cohen et al. [2003]. These metrics can only measure syntactic similarity (or distance) between two strings but cannot capture semantic similarities such as synonyms, acronyms, and abbreviations.

Machine learning-based methods [Bilenko and Mooney 2003; Tsuruoka et al. 2007] have been proposed to improve the quality of traditional syntactic similarity-based approaches. In particular, a learnable string similarity [Bilenko and Mooney 2003] presents trainable measures to improve the effectiveness of duplicate detection, and Tsuruoka et al. [2007] use logistic regression to learn a string-similarity measure from an existing gene/protein name dictionary. Although these methods are effective in capturing semantic similarities between strings, they are limited to special domains and accuracy depends critically on the quality of the training data.

String-Similarity Searches and Joins. State-of-the-art approaches [Gravano et al. 2001; Chaudhuri et al. 2003, 2006; Sarawagi and Kirpal 2004; Bayardo et al. 2007; Li et al. 2012; Wang et al. 2012; Bille 2012] for performing string-similarity searches and joins mainly follow the *filtering-and-verification* framework. The basic idea is to first employ an efficient filter to prune string pairs that cannot be similar, and then verify the surviving string pairs by computing their real similarity. In the filtering step, there are two methods for generating signatures for each string: (i) *prefix filtering scheme*

[Chaudhuri et al. 2006; Bayardo et al. 2007; Xiao et al. 2008; Wang et al. 2012] first transforms each string to a set of tokens. Then the tokens of each string are sorted based on a *global ordering*, and a prefix set for each string is selected based on a given similarity threshold to be signatures. The signatures can be used to guarantee that if two strings are similar, then their signatures have enough overlap; (ii) *LSH-based scheme* [Chaudhuri et al. 2003; Hadjieleftheriou et al. 2008; Xiao et al. 2011] generates the signatures based on the idea of locality-sensitive hashing (LSH) [Broder et al. 1997], that is, each signature is a concatenation of k minhashes of the set s under a minwise independent permutation, and l such signatures are generated. k and l are two user-defined parameters. The main limitation of the LSH approach is that the algorithms are approximate and could miss some correct results. In this article, we will extend these two signature schemes to perform efficient similarity join with synonyms.

Synonyms Generation Approaches. Synonym pairs can be obtained in many ways, such as users' existing dictionaries [Tsuruoka et al. 2007], explicit inputs, and synonym mining algorithms [Arasu et al. 2009]. In this article, we assume that all synonym pairs are predefined, and our focus is how to effectively use the semantic information to provide meaningful similarity measures and efficient join algorithms.

Estimation of Set Size. Our work is also related to the estimation of distinct pairs in set-union. This is because, in the filtering phase, we need to estimate the number of candidates (string pairs) morden to dynamically select a signature scheme. There are many solutions proposed for the estimation of the cardinality of distinct elements. For example, in their influential paper, Flajolet and Martin [1985] propose a randomized estimator for distinct-element counting that relies on a hash-based synopsis; to date, the Flajolet-Martin (FM) technique remains one of the most effective approaches for this estimation problem. FM sketch and its variations (e.g., [Alon et al. 1996; Ganguly et al. 2004]) focus on one-dimensional data. In our problem, we need to estimate the distinct number of pairs, where each dimension is based on an independent FM sketch from one join collection. It turns out that this problem is more complicated, and the existing analysis cannot be extended easily. In addition, we note that the estimation of the resulting size of a string-similarity join has been studied recently (e.g., [Lee et al. 2009, 2011]), by using random sampling and an LSH scheme. Note that these techniques cannot be directly applied here, since given a specific filter, our goal is to estimate the cardinality of candidates, which is often much greater than that of final results.

Finally, this article extends a previous conference paper [Lu et al. 2013] with the following new contributions. (1) We study a new problem of approximate string search with synonyms (Section 4), and show two algorithms, that is, a *search-baseline* and an indexed approach *QP-search* based on a native index (QP-tree); and (2) we add the extensions to several issues (Section 7), including three new similarity functions and the weighted tokens.

3. STRING-SIMILARITY MEASURES

As described in Section 1, an expansion-based framework can efficiently measure the similarity of strings with synonyms to avoid exponential computational complexity. In this section, we describe two expansion-based measures in details. In particular, Section 3.1 first introduces the preliminaries about synonym rules. Then Section 3.2 and Section 3.3 describe two expansion-based measures: *full expansion* and *selective expansion*. Finally, Section 3.4 analyzes the optimality of the proposed methods.

3.1. Preliminaries and Notations

Given two strings s_1 and s_2, let S_1 and S_2 denote the sets generated from s_1 and s_2, respectively, by tokenization (e.g., splitting strings based on delimiters such as white spaces). Let r denote a rule, which has the form: $lhs(r) \rightarrow rhs(r)$, where $lhs(r)$ and $rhs(r)$ are the left- and right-hand sides of r, respectively. Slightly abusing the notation, $lhs(r)$ and $rhs(r)$ can also refer to the *set* generated from the respective strings. Let $\mathbb{R}$ denote a collection of synonym rules, that is, $\mathbb{R} = \{r\colon lhs(r) \rightarrow rhs(r)\}$. Given a string s, we say a rule $r \in \mathbb{R}$ is an *applicable rule* of s if $lhs(r)$ is a *substring* of s and let $\mathcal{R}(s)$ denote all *applicable rules* of s.

Example 3.1. Given two strings: s_1 = "`SIGMOD Conf 2012 Arizona`", s_2 = "`ACM SIGMOD Conference 2012 Arizona`", and two rules: r_1: `Conf` $\rightarrow$ `Conference`, r_2: `ACM` $\rightarrow$ `Association for Computing Machinery`, the *applicable rule sets* are $\mathcal{R}(s_1) = \{r_1\}$, $\mathcal{R}(s_2) = \{r_2\}$.

3.2. Full Expansion

Given a string s, suppose a rule $r : lhs(r) \rightarrow rhs(r) \in \mathbb{R}$ is an applicable rule of s, and we use $rhs(r)$ to expand the set S, that is, $S' = S \cup rhs(r)$. Let $sim(s_1, s_2, \mathbb{R})$ denote the similarity between s_1 and s_2 based on $\mathbb{R}$. Let $\mathcal{R}(s_1)$ and $\mathcal{R}(s_2)$ denote the applicable rule set of s_1 and s_2, respectively. Under the expansion-based framework, $sim(s_1, s_2, \mathbb{R}) = f(S_1', S_2')$, where S_1' and S_2' are expanded from S_1 and S_2 using some synonym rules in $\mathcal{R}(s_1)$ and $\mathcal{R}(s_2)$, and f is a set-similarity function such as Jaccard, Cosine, etc.

Based on how *applicable synonyms* are selected to expand S, there are different kinds of schemes. A simple one is the *full expansion*, which expands S using all their applicable synonyms. That is, the expanded set $S' = S \bigcup_{r \in \mathcal{R}(s)} rhs(r)$.

The full expansion scheme is efficient to compute. Let L be the maximum length of strings s_1 and s_2. The time cost of substring matching to identify the applicable synonym pairs is $O(L)$ using a suffix-trie-based method [Farach 1997]. Then the complexity of the set expansion is $O(L+kn)$, where n is the maximum number of applicable synonym pairs for s_1 and s_2, and k is the maximum size of the right-hand side of a rule. As we will demonstrate in our experimental section, the full expansion provides a highly competitive performance with other optimized schemes to capture semantics of strings using synonyms.

3.3. Selective Expansion

We observe that, as mentioned in Section 1, the full expansion cannot guarantee that each applied rule is useful for similarity measure between two strings. Recall Example 1.2 and Figure 1. The use of rule r_4 on t_2 is detrimental to the similarity between q_1 and t_2, as "`United Kingdom`" does not appear in q_1. To address this issue, we propose an optimized scheme, called *selective expansion*, whose goal is to maximize the similarity value of two expanded sets by a judicious selection of applicable synonyms to apply. Given a collection of synonym rules $\mathbb{R}$ and two strings s_1 and s_2, the selective expansion scheme maximizes $f(S_1', S_2')$. Suppose, without loss of generality, we use Jaccard coefficient to measure the similarity between S_1' and S_2', which can be extended to other functions. The *selective-Jaccard* similarity of s_1 and s_2 based on $\mathbb{R}$ (denoted by $SJ(s_1, s_2, \mathbb{R})$) is defined as follows:

$$SJ(s_1, s_2, \mathbb{R}) = \max_{\mathcal{R}(s_1) \subseteq \mathbb{R}, \mathcal{R}(s_2) \subseteq \mathbb{R}} \{Jaccard(S_1', S_2')\},$$

where S_1' and S_2' are the sets expanded from s_1 and s_2 using $\mathcal{R}(s_1)$ and $\mathcal{R}(s_2)$, respectively.

Example 3.2 (*Continuation of Example 3.1*). Based on the *full expansion scheme*, $S'_1 = S_1 \cup rhs(r_1) = \{$`SIGMOD, Conf, Conference, 2012, Arizona`$\}$. $S'_2 = S_2 \cup rhs(r_2) = \{$`ACM, SIGMOD, Conference, 2012, Arizona, Association, for, Computing, Machinery`$\}$. Therefore, Jaccard$(S'_1, S'_2) = \frac{4}{10}$. However, based on the selective expansion scheme, we can only use r_2 to expand S_1 to achieve the maximal similarity. Then, $S''_1 = \{$`SIGMOD, Conference, Conf, 2012, Arizona`$\}$. Therefore, $SJ(s_1, s_2, \mathbb{R}) = Jaccard(S''_1, S_2) = \frac{4}{5}$.

Unfortunately, we can prove that it is NP-hard to compute the *selective-Jaccard* similarity with respect to the number of applicable synonym rules, by a reduction from the well-known 3SAT problem [Iwama and Tamaki 2004].

THEOREM 3.3. *Selective-Jaccard* $\in$ *NP-hard.*

The detailed proof can be found in Appendix A. Since *selective-Jaccard* is NP-hard, we design an efficient greedy algorithm that guarantees optimality under certain conditions, which has been shown to be met in most cases in practice. Before presenting the algorithm, we will first define the notion *rule gain*, which can be used to estimate the fitness of a rule to strings.

ALGORITHM 1: Computing a Rule Gain

Input : $r, s_1, s_2, \mathbb{R} = \{r_i : lhs(r_i) \rightarrow rhs(r_i)\}$
Output: the rule gain of r, denoted $RG(r, s_1, s_2, \mathbb{R})$
1 $U \leftarrow rhs(r) - S_1$; // r is applicable to s_1, and S_1 is the token set of s_1
2 **if** $(U = \phi)$ **then**
3 | **return** 0;
end
//Let $\mathbb{R}' \subseteq \mathbb{R}$ denote all applicable rules of s_2
4 $S'_2 \leftarrow S_2 \bigcup_{r_i \in \mathbb{R}'} rhs(r_i)$ //full-expanding S_2
5 $G \leftarrow (rhs(r) \cap S'_2) - S_1$;
6 **return** $RG(r, s_1, s_2, \mathbb{R}) = \frac{|G|}{|U|}$;

Given two strings s_1 and s_2, a collection of synonyms $\mathbb{R}$, and a rule $r \in \mathbb{R}$, we assume that r is an applicable rule for s_1. We denote the rule gain of r by $RG(r, s_1, s_2, \mathbb{R})$. When s_1, s_2, and $\mathbb{R}$ are clear from the context, we shall simply speak of $RG(r)$. Informally, the rule gain is defined by the number of useful elements introduced by r divided by the number of new elements in r. In particular, we say that a token $t \in rhs(r)$ in r is useful if t belongs to the full expansion set of s_2 and $t \notin s_1$. The reason we use the full expansion set of s_2 rather than the original set is that s_2 can be expanded using new rules, but all possible new tokens must be contained in the full expansion set. We now go through Algorithm 1. In line 1, we find the new elements U introduced by r. If $U = \varnothing$, then the rule gain is zero (lines 2 $\sim$ 3). Line 4 fully expands s_2, and line 5 computes the useful elements G. Finally, the rule gain $RG(r, s_1, s_2, \mathbb{R}) = \frac{|G|}{|U|}$ is returned (line 6).

Example 3.4 (*Continuation of Example 3.1*). Since r_1 is applicable for s_1, we compute $RG(r_1, s_1, s_2, \mathbb{R})$. Note that $G = U = \{$`Conference`$\}$. Therefore, $RG(r_1, s_1, s_2, \mathbb{R}) = 1$. Further, r_2 is applicable for s_2, then we compute $RG(r_2, s_2, s_1, \mathbb{R})$. Since $U = \{$`Association, for, computing, machinery`$\}$, and $G = \emptyset$, $RG(r_2, s_2, s_1, \mathbb{R}) = 0$.

We develop a *selective expand* algorithm based on rule gains, which is formally described in Algorithm 2. Before doing any expansion, we first compute a candidate set of rules (line 1), which only consists of applicable rules with relatively higher

rule-gain values. In particular, in procedure *findCandidateRuleSets*, we first let $\mathcal{C}_i$ contain all applicable rules whose rule gain is greater than zero (line 4), and then we iteratively remove the useless rules whose rule gain is too small from $\mathcal{C}_i$ (lines 5∼11). In lines 6∼7, we use the elements in $\mathcal{C}_i$ to expand S_i to acquire S_i'. Then, in lines 9∼11, any rule whose rule gain is less than $\frac{\alpha}{1+\alpha}$ is removed from $\mathcal{C}_i$, where α is the Jaccard similarity of the current expanded sets S_1' and S_2' (line 8), as its similarity is lower than the "average" similarity and its usage would be detrimental to the final similarity. Subsequently, in line 2, we iteratively expand s_1 and s_2 using the most gain-effective rule from the candidate sets. Finally, line 3 returns the maximal value between $Jaccard(s_1, s_2)$ (without the synonyms) and the new similarity θ using synonyms.

Example 3.5. We use Example 3.1 again to illustrate Algorithm 2. First, in line 4, $\mathcal{C}_1 = \{r_1\}$, and $\mathcal{C}_2 = \{r_2\}$. Then $S_1' = \{$`SIGMOD, Conf, Conference, 2012, Arizona`$\}$, $S_2' = \{$`ACM, SIGMOD, Conference, 2012, Arizona, Association, for, Computing, Machinery`$\}$ (lines 6 and 7). Therefore, $\alpha = \text{Jaccard}(S_1', S_2') = 4/10$ (line 8). Since $\text{RG}(r_1) = 1$ and $\text{RG}(r_2) = 0$, r_2 is removed from $\mathcal{C}_2$ (line 11). Therefore, only $\mathcal{C}_1 = \{r_1\}$ is returned in line 13. Subsequently, in line 19, $rhs(r_1)$ is added to S_1'. Therefore, only one rule r_1 is used to expand s_1, and the final similarity is $\text{Jaccard}(S_1', S_2) = 4/5$.

Some readers may be wondering whether we can skip procedure *findCandidateRuleSet* in Algorithm 2 and directly use the greedy strategy in line 2 to select rules for expansion based on all applicable rules. Our answer is no, because line 1 wisely selects only part of the applicable rules in order to avoid the blind expansion with rules that are not suitable for global optimization.

Example 3.6. We use this example to demonstrate the importance of procedure *findCandidateRuleSet* in Algorithm 2. Given two short strings: $s_1 =$ "a," $s_2 =$ "b," and five rules: r_1: "a" → "$b\ t_1$," r_2: "a" → "$c\ d\ e$," r_3: "b" → "$c\ t_2\ t_3\ t_4\ t_5$," r_4: "b" → "$d\ t_6\ t_7\ t_8\ t_9$," and r_5: "b" → "$e\ t_{10}\ t_{11}\ t_{12}\ t_{13}$," where all items from t_1 to t_{13} are "dirty" words which cannot contribute to the similarity between s_1 and s_2. It turns out that the maximum similarity is 1/3 = Jaccard("$a\ b\ t_1$","b"), which is achieved by applying only the rule r_1 on s_1. In Algorithm 2, line 1 returns $\mathcal{C}_1 = \{r_2\}$, because those rules from t_2 to t_{13} are all removed from $\mathcal{C}_i$ in line 11. Therefore, only r_1 is used to expand s_1. But if line 1 were cancelled, then all rules could be used to expand s_1 or s_2. Since $RG(r_2) = 1 > RG(r_1) = 1/2$. Then r_2 is used to expand s_1. Subsequently, the other four rules are applied as well. Then the final similarity falls to $5/18 < 1/3$.

In addition, as shown in the proof of Theorem 3.7, line 10 in procedure *findCandidateRuleSet* lies in a key condition in order to guarantee the optimality of our algorithm in some scenarios.

Performance Analysis. The time complexity of the *SE* algorithm is $O(L + kn^2)$, where L is the maximum length of s_1 and s_2, n is the total number of applicable synonyms for s_1 and s_2, and k is the maximum size of the right-hand side of a rule. More precisely, we use the suffix-trie to find applicable rule sets, and its complexity is $O(L)$. In procedure *findCandidateRuleSet*, the complexity to perform full expansion (lines 6 ∼ 7) is $O(kn)$, and the iteration time is no more than n; thus, the complexity is $O(L + kn^2)$.

3.4. Theoretical Analysis

In this section, we carve out one case and show that SE returns the optimal value and then analyze the reduced time complexity in this optimal case.

ALGORITHM 2: Selective Expansion (SE) Algorithm

Input : $s_1, s_2, \mathbb{R} = \{r: lhs(r) \rightarrow rhs(r)\}$.
Output: $Sim(s_1, s_2, \mathbb{R})$.
1 $\mathcal{C}_1, \mathcal{C}_2 \leftarrow$ findCandidateRuleSet$(s_1, s_2, \mathbb{R})$;
2 $\theta \leftarrow$ expand$(s_1, s_2, \mathcal{C}_1, \mathcal{C}_2)$;
3 **return** *max(*$Jaccard(s_1, s_2)$*, θ);*

Procedure findCandidateRuleSet$(s_1, s_2, \mathbb{R})$
4 $\mathcal{C}_i \leftarrow \{$r: r $\in \mathbb{R} \wedge lhs(r)$ is a substring of s_i(i = 1 or 2) $\wedge\ RG(r) > 0\}$;
5 **repeat**
6 $S'_1 \leftarrow S_1 \bigcup_{r \in \mathcal{C}_1} rhs(r)$;
7 $S'_2 \leftarrow S_2 \bigcup_{r \in \mathcal{C}_2} rhs(r)$;
8 $\alpha \leftarrow Jaccard(S'_1, S'_2)$;
9 **for** *(*$r \in \mathcal{C}_1 \cup \mathcal{C}_2$*)* **do**
10 **if** $(RG(r, s_1, s_2, \mathcal{C}_1 \cup \mathcal{C}_2) < \frac{\alpha}{1+\alpha})$ **then**
11 $\mathcal{C}_i \leftarrow \mathcal{C}_i - r$; *//*$r \in \mathcal{C}_i$, $i = 1$ or 2
12 Recompute S'_1, S'_2 and α;
end
end
until *(no rule can be removed in line 11)*;
13 **return** $\mathcal{C}_1, \mathcal{C}_2$*;*

Procedure expand$(s_1, s_2, \mathcal{C}_1, \mathcal{C}_2,)$
//Let S'_1 *and* S'_2 *denote the expanded sets of* S_1 *and* S_2
14 $S'_1 \leftarrow S_1$;
15 $S'_2 \leftarrow S_2$;
16 **while** *(*$\mathcal{C}_1 \cup \mathcal{C}_2 \neq \phi$*)* **do**
17 Find the *current* most rule-gain-effective rule $r \in \mathcal{C}_i$ (i = 1 or 2);
18 **if** *((*$RG(r, s_1, s_2, \mathcal{C}_1 \cup \mathcal{C}_2) > 0$*)* **then**
19 $S'_i \leftarrow S'_i \cup rhs(r)$;
end
20 $\mathcal{C}_i \leftarrow \mathcal{C}_i$- r;
end
21 **return** *Jaccard(*S'_1*,* S'_2*);*

THEOREM 3.7. *Given two strings s_1 and s_2 and their applicable rule sets $\mathcal{R}(s_1)$ and $\mathcal{R}(s_2)$, we call a rule $r \in \mathcal{R}(s_i)$ ($i = 1$ or 2) useful if $RG(r) > 0$. We can guarantee that SE is optimal if the rhs tokens of all useful rules are distinct.*

The proof of Theorem 3.7 can be found in Appendix B. Note that the condition in Theorem 3.7 does not require all the *rhs* tokens to be distinct. It only requires those from useful synonyms of s_1 and s_2 to be distinct, which are typically a very small subset. In addition, under the condition, the time complexity can be reduced to $O(L+n\log n+nk)$. The reason is that the complexity to perform full expansion (lines 6∼7) is reduced to $O(1)$. Since the tokens are distinct, and that of expanding (lines 15∼19) is $O(n\log n)$, since the rule gain of each rule does not change, which can be implemented by a maximal heap. Finally, the *Jaccard* computation in line 20 needs $O(L+kn)$ time in the worst case. Therefore, the complexity of *SE* algorithm is $O(L+n\log n+nk)$.

4. STRING SIMILARITY SEARCH

In this section, we study the similarity search problem. That is, given a query string q and a collection of strings T, a collection of synonyms $\mathbb{R}$, and a similarity threshold θ, *similarity search* finds all strings $t \in T$, such that $sim(q, t, \mathbb{R}) \geq \theta$, where *sim* is one

Query q="ACM SIGMOD 2013"

ID	Synonym pairs	ID	Synonym pairs
r_1	Intl → International	r_3	VLDB → Very Large Data Bases
r_2	Conf → Conference	r_4	Very Large Data Bases → VLDB
r_5	ACM → Association for Computing Machinery		
r_6	SIGMOD → International Conference on Management of Data		

(a) A query string and synonym pairs.

Possible expanded sets	Signatures
ACM SIGMOD 2013	SIGMOD
ACM Association for Computing Machinery SIGMOD 2013	Computing
ACM SIGMOD International Conference on Management of Data 2013	Conference
ACM Association for Computing Machinery SIGMOD International Conference on Management of Data 2013	Conference, on

(b) Possible expanded sets and signatures.

Fig. 3. An Example to illustrate the generation of signatures (threshold = 0.9).

of the similarity functions in Section 3. Two search algorithms are proposed in the following sections.

4.1. Preliminary on Prefix Filters

A naïve algorithm for computing similarity search is to enumerate and compare every pair of records with the query string. This method is obviously prohibitively expensive for large datasets. Efficient algorithms exist by following the filtering-and-verification framework, that is, given a query q and a table T, one or multiple filters are utilized to prune away many records in T and then the verification is invoked for the remaining records after filtering. Clearly, the stronger filtering power the method has, the better performance than the naïve algorithm. In the literature, the current modus operandi is called *prefix filter*, which is based on the intuition that if two canonicalized records are similar, some fragments of them should overlap with each other, as otherwise the two records won't have enough overlap. This intuition can be formally captured by the prefix-filtering principle [Chaudhuri et al. 2006] rephrased here.

LEMMA 4.1 (PREFIX FILTER PRINCIPLE [CHAUDHURI ET AL. 2006]). *Given an ordering O of the token universe U and two strings s and t, each with tokens sorted in the order of O, if Jaccard$(s,t) \geq \theta$, then the first $\lceil(1-\theta)|s|\rceil$ smallest tokens of s and the first $\lceil(1-\theta)|t|\rceil$ smallest tokens of t must share at least one token.*

4.2. Search-Baseline Method

The first search algorithm we propose here is a prefix-filter-based approach, called *search-baseline*, following the filtering-and-verification framework. That is, in the filtering step, it generates candidate strings by identifying all strings t such that the signatures of q and t overlap. In the verification step, it checks if $sim(q,t,\mathbb{R}) \geq \theta$ for each candidate and outputs those that satisfy the similarity predicate.

To generate the signatures, given a string s, the signatures of s are a subset of tokens in s containing the $\lceil(1-\theta)|s|\rceil$ smallest elements of s according to a predefined global ordering. Therefore, we generate signatures for a string s as follows. Let $\mathcal{R} \subseteq \mathbb{R}$ be the set of n application synonym pairs for s. For each subset $\mathcal{R}_i$ of $\mathcal{R}$, we generate an expanded set S_i. For each S_i, we obtain one signature set (denoted by $sig(S_i)$). Note that the number of subset S_i is 2^n. The exponential number is due to the fact that we try to enumerate all possibilities of expanded sets for s with n synonym pairs. Finally, we generate a signature set for s by including all signatures of S_i: $Sig(s,\mathcal{R}) = \bigcup_{i=1}^{2^n} Sig(S_i)$. Note that this generation of signatures can be performed offline.

LEMMA 4.2. *Given two strings s_1 and s_2, a threshold value θ, and a collection of synonym pairs $\mathbb{R}$, if $Sig(s_1,\mathbb{R}) \cap Sig(s_2,\mathbb{R}) = \emptyset$, then $sim(s_1,s_2,\mathbb{R}) < \theta$, where sim denotes any of the similarity functions proposed in Section 3.*

PROOF. $Sig(s_1,\mathbb{R}) \cap Sig(s_2,\mathbb{R}) = \emptyset$. $\Longrightarrow$ For any S_{1i}, S_{2j} expanded from s_1 and s_2, $Sig(S_{1i}) \cap Sig(S_{2j}) = \emptyset$. $\Longrightarrow$ At least $\lceil(1-\theta)|S_{1i}|\rceil$ elements of S_{1i} and $\lceil(1-\theta)|S_{2j}|\rceil$

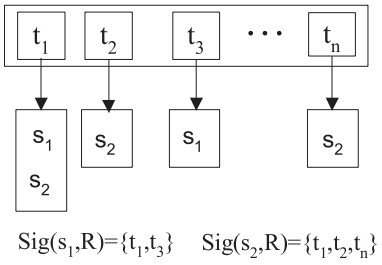
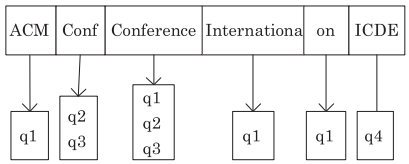


Fig. 4. The structure of I-list.

ID	Strings	Signatures
q1	2013 ACM Intl Conf on Management of Data	ACM, International, Conference, on
q2	Very Large Data Bases Conf	Conf, Conference
q3	VLDB Conf	Conf, Conference
q4	ICDE 2013	ICDE

(a) Strings and signatures



(b) An I-list

Fig. 5. An example to illustrate I-list.

elements of S_{2j} have no common tokens. $\Longrightarrow$ The upper bound of the similarity $< \theta$, as desired. □

Example 4.3. This example is employed to illustrate the generation of signatures with synonyms. Consider the string q and synonyms in Figure 3(a). The applicable synonyms for q are r_5, r_6, and the expanded sets of q are shown in Figure 3(b). For each set, we choose the $\lceil (1 - 0.9)|s| \rceil$ tokens to be the signatures for the set, which are presented in Figure 3(b). Finally, the signature of q can be computed as the union of the signatures, that is, $Sig(q, \mathbb{R}) = \{SIGMOD, Computing, Conference, on\}$.

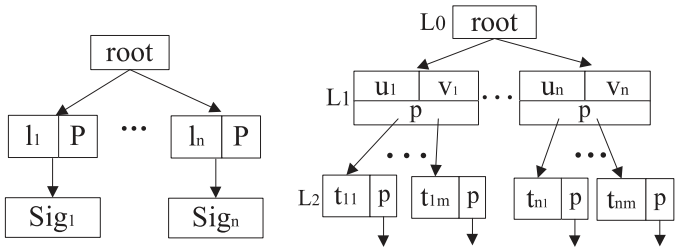
In our implementation, in order to avoid the most time-consuming operation, that is, checking the signature overlap for each string pair to find the candidates, we design a signature inverted list, called *I-list* (see Figure 4). Each token t_i is associated with an inverted list including all string IDs whose signatures contain the token t_i. Thus, given a signature t_i, we can directly obtain all relevant strings without any signature-overlapping check.

The search-baseline algorithm operates with two steps in the filtering phase. In the first step, for each signature of query q, function getIList is employed to get the I-list whose key is g. In the second step, the I-lists are merged together to obtain the final candidates. Then in the verification phase, any of the two measures proposed in Section 3 can be applied to check the candidates.

Example 4.4. This example is used to illustrate the search-baseline method. Consider the query q in Figure 3 and the table in Figure 5(a). Search-baseline first computes signatures for both query and strings in the table. Secondly, an I-list of the signatures is constructed, as shown in Figure 5(b). Then search-baseline uses the query signatures $\{SIGMOD, Computing, Conference, on\}$ to find corresponding inverted lists, that is, $\{q_1, q_2, q_3\}$ and $\{q_1\}$. Finally, only q_1 is returned as the query answer based on the selected-expansion similarity function.

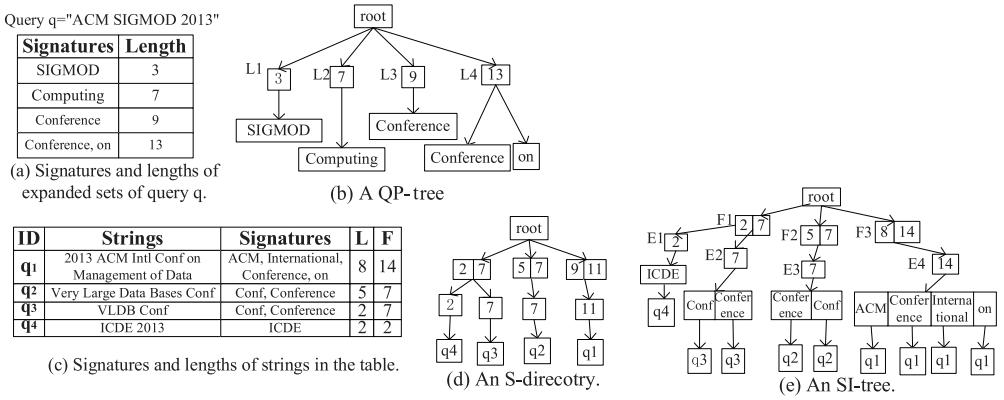
4.3. QP-Search Algorithm

In this section, we propose two native indexes for both query strings and the records in the table, respectively, to improve the efficiency of the search-baseline method.



(a) The structure of QP-tree. (b) The structure of S-directory.

Fig. 6. The structure of QP-tree and S-directory.



(c) Signatures and lengths of strings in the table.

(d) An S-direcotry.

(e) An SI-tree.

Fig. 7. An example to illustrate the indexes. The column L and F denote the number of tokens in the string and in the full expanded set, respectively.

QP-Tree for the Query q. For the query string q, instead of computing the union signatures $Sig(q, \mathbb{R})$, we consider the signatures $Sig(Q_i)$ for each expanded set Q_i. Note that the length $|Sig(Q_i)|$ of the expanded sets here could be utilized to enhance the filtering power. We build a query pattern tree, called *QP-tree* (see Figure 6(a)), which has two levels: the nodes in the first level contain two fields $\langle l, p \rangle$, where l is an integer to denote the size of each expanded set of query q, and p is a pointer to a signature set. For example, the QP-tree in Figure 7(b) is built for the query in Figure 7(a). Building QP-tree is on-the-fly for each coming query, which is fast.

SI-Tree for the Collection of Strings. We extend the idea of *length filter* [Sarawagi and Kirpal 2004; Bayardo et al. 2007] to reduce the number of candidates. Intuitively, given two strings s_1 and s_2 and a threshold value θ, if the size of the full expansion set of s_1 is much less than the size of s_2, then without checking the signatures of s_1 and s_2, one can claim that $sim(s_1, s_2, \mathbb{R})$ must be smaller than θ. Based on the observation, we propose a data structure, called *S-directory*, to organize the strings by their lengths.

The S-directory (see Figure 6(b)) is a tree structure with three levels, and nodes in different levels can be categorized into different kinds.

—*Root Entry*. A node in level L_0 is a root entry which has multiple pointers to the nodes in level L_1.

—*Fence Entry*. A node in level L_1 is called *fence entry* and contains three fields $\langle u, v, p \rangle$, where u and v are integers, and p is a set of of pointers to the nodes in L_2. In particular, u denotes the number of tokens of a string and v is the maximal number of tokens in the fully-expanded sets of strings whose length is u.

—*Leaf Entry*. A node in level L_2 is called *leaf entry* and contains two fields $\langle t, p\rangle$, where t is an integer to denote the number of the tokens in the fully-expanded set of a string whose length is u, and p is a pointer to an inverted list. As leaf entries are organized in the ascending order of t, $v_i = t_{im}$ (see Figure 6(b)).

We argue that S-directory can easily fit in the main memory, as the size of S-directory is only $O((v_{max} - u_{min})^2)$, which is quadratic to the difference between the maximal size of expanded sets and the shortest length of records. For example, in our implementation, the size of S-directory for a table with one million strings on a USPS dataset is only about 2.15KB.

To maximize the pruning power, we combine I-list and S-directory together. That is, for each *leaf-entry* $= \langle t, p\rangle$, p points to a set of signatures (from the string whose size of the full expansion set is t) associated with multiple I-lists. Thus, we have a combined index, called *SI-tree*, that is, each leaf entry in S-directory is associated with an I-list. For example, in Figure 7(d) and Figure 7(e), we draw the S-directory and SI-tree for data in Figure 7(c), respectively.

ALGORITHM 3: QP-Search Algorithm

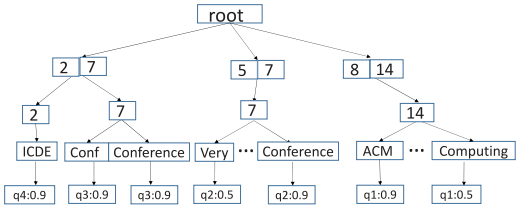
```
   Input  : QP-tree for query q and SI-tree for table T, a threshold θ, a collection of
            synonyms ℝ
   Output: all records r ∈ T: sim(q, r, ℝ) ≥ θ
 1 ℂ ← ∅, R ← ∅;
 2 for (∀ F in SI, ∀ L in QP) do
       //Fence-entry:F=<u, v, P>, Length-entry:L=<l, P>
 3     if (F.u × θ ≤ L.l ≤ F.v/θ) then
 4         for (∀ E∈F.P) do
               //Leaf-entry:E=<t, P>
 5             if (min(E.t, L.l) ≥ max(E.t, L.l) × θ) then
 6                 for (∀ g is an overlapping signature between L.P and E.P) do
 7                     C = E.P → getIList(g);
 8                     ℂ = ℂ ∪ C;
                   end
               end
           end
       end
   end
 9 for (∀ r in ℂ) do
10     if sim(q, r, ℝ) ≥ θ then
           R = R ∪ r;
       end
   end
11 Return R;
```

Based on the SI-tree and QP-tree, we design an algorithm (called *QP-search*) to quickly select candidate strings for verification. See Algorithm 3, where line 3 and line 5 select the desired entries by length filtering. Then, in lines 6 $\sim$ 8, for each signature g that is an overlapping signature between the query and the table, function getIList returns the I-list whose key is g. Finally, in line 9, QP-search returns the candidate strings for query q.

Example 4.5. We use this example to illustrate the QP-search algorithm. Consider the query and the table in Figure 7. The QP-tree and SI-Tree are shown in Figures 7(b) and 7(e). The query is $q =$ "ACM SIGMOD 2013," and the threshold $\theta = 0.9$. Entries L_1

ID	Strings	Signatures(0.9)	Signatures(0.5)
q1	2013 ACM Intl Conf on Management of Data	ACM International Conference on	2013 ACM Intl Conf on Management of Data International for Conference on Computing
q2	Very Large Data Bases Conf	Conf Conference	Very Large Conf Conference
q3	VLDB Conf	Conf Conference	Conf Conference
q4	ICDE 2013	ICDE	ICDE

(a) Signatures of strings with thresholds 0.9 and 0.5.



(b) An FSI-tree.

Fig. 8. An example to illustrate FSI-trees.

and F_1 satisfy the condition in line 3 ($2 \times 0.9 \leq 3 \leq 7/0.9$), but L_1 and E_2 cannot pass the test in line 5. Next, the entry L_2 satisfies the conditions in both lines 3 and 5 with F_2 and E_3, respectively. Unfortunately, there is no overlapping signatures between L_2 and E_3, thus entry L_3 cannot pass the length filter. Finally, consider entries L_4 and E_4, both of which luckily satisfy the conditions and have overlapping signatures "Conference" and "on." Subsequently, we obtain the candidate, that is, q_1. To demonstrate the advantage of the QP-search algorithm over the search-baseline, note that in the filtering phase, search-baseline returns three candidates (see Example 4.4), whereas QP-search returns only one candidate. Finally, the answer is "2013 ACM Intl Conf on Management Data," which refers to the same conference with q.

4.4. Extensions for Flexible Query Thresholds

In the previous sections, our model assumes that the search threshold is fixed and that only the query can be changed online. However, in practice, users might change the threshold at query time. Therefore, we now move to a more general case, where both the search string and the threshold are flexible at query time. The new challenge here is that we do not know the threshold in advance, and thus we cannot determine the number of signatures of each record. A naïve method is to compute the signatures of records in the table online according to the given threshold, which is clearly prohibitively expensive. Next we build a new index called *FSI-trees* (flexible signature indexing) by extending SI-trees to solve this problem.

Suppose that all meaningful thresholds distribute in the range between 0.99 to 0.50. Then we select some *representative thresholds* (e.g., 0.95, 0.90, etc.) For each representative threshold, we generate signatures for each record: see an example in Figure 8(a). Note that the signatures of a string for lower thresholds are guaranteed to become signatures of that for higher thresholds. To build an FSI-tree, the length and fence entries of SI-trees remain the same, but each fence entry points to a set of I-lists which come from all representative thresholds. Further, each element in the I-list of a signature token s is a binary tuple (q, θ), where q is a record ID and θ is the minimal threshold for which this signature s appears in q. For example, in Figure 8(b), the token "Computing" is a signature of q_1 for all thresholds ≥ 0.5.

We extend the QP-search algorithm for flexible thresholds. The algorithm almost remains the same, the only change (see Algorithm 4) being that we select the string candidates in I-lists by checking their thresholds (line 3).

An astute reader may notice that our method possibly introduces more candidates because of the gap between representative thresholds and online thresholds. For example, given a query threshold 0.83, suppose that the closest representative threshold is 0.80. Then the number of signatures for threshold 0.80 may be greater than that for 0.83, but we argue that the problem has actually little impact on the final performance of query processing. To understand this, assume that gap between two representative thresholds is no more than 0.05 (that means, only 11 representative thresholds are

ALGORITHM 4: Extended QP-Search for Flexible Thresholds

```
Input  : QP-tree for query q and DSI-tree for table T, a threshold θ, a collection of
         synonyms ℝ.
Output: all records r ∈ T: sim(q, r, ℝ) ≥ θ
1 Run lines 1 ∼ 7 in Algorithm 3;
2 for (∀ (q, θ_q)∈C) do
3     if (θ_q ≥ θ) then
4         ℂ = ℂ ∪ {q};
      end
  end
5 Continue lines 9 ∼ 11 in Algorithm 3;
```

"materialized" with signatures between 0.99 and 0.50). It can be proved that given a string s, the difference between the numbers of signatures for thresholds θ_1 and θ_2 is $\lceil|\theta_1 - \theta_2| \cdot |s|\rceil$. Considering a string $|s| = 10$, we have $0.05 * 10 = 0.5$, that is, for most records in the table, the extra number of signatures due to the thresholds gap is bounded by 0.5. Therefore, as our experimental results show in Section 8.2.2, the performance of our algorithms for flexible thresholds is comparable to that for static thresholds.

5. STRING SIMILARITY JOINS

In this section, we formulate the string-similarity join problem and develop the corresponding algorithms. Given two collections of strings S and T, a collection of synonyms $\mathbb{R}$, and a similarity threshold θ, a *string-similarity join* finds all string pairs $(s, t) \in S \times T$, such that $sim(s, t, \mathbb{R}) \geq \theta$, where *sim* is one of the similarity functions in Section 3. Two join algorithms would be proposed in the following sections.

5.1. Join-Baseline Method

The *join-baseline* (see Algorithm 5) method follows the filtering-and-verification framework. In the filtering phase, it generates signatures for each string, then filters candidates by checking the overlaps of the signatures. Note that the way to generate signatures is the same to that of the search-baseline method. In the verification phase, it verifies the candidates by employing any of the two measures in Section 3.

ALGORITHM 5: Join-Baseline Algorithm

```
Input  : I_s and I_t //I_s and I_t are I-list index for table S and T, respectively
Output: ℂ: Candidate string pairs (s,t)∈ I_s × I_t
1 ℂ ⟵ ∅;
2 for (∀ g ∈ I_s ∩ I_t) do
      //g is a signature
3     L_s = I_s → getIList(g);
4     L_t = I_t → getIList(g);
5     C = {(s, t)|s ∈ I_s, t ∈ I_t};
6     ℂ = ℂ ∪ C;
  end
7 return ℂ;
```

Example 5.1. We use this example to illustrate the join-baseline algorithm. Consider a self join on the table in Figure 7(c), with $\theta = 0.8$. The corresponding signatures are listed in Figure 7(c). In the filtering phase, the join-baseline method generates candidates (q_1, q_2), (q_1, q_3), (q_2, q_3), and $\{(q_i, q_i)|i \in [1, 4]\}$, since the signatures of those string pairs overlap.

5.2. SI-Join Algorithm

To further optimize the join-baseline method, we propose an index-join algorithm by using SI-trees proposed in Section 4.

ALGORITHM 6: SI-Join Algorithm

Input : SI-Trees SI_S for S and SI_T for T, threshold value θ, a collection of synonym pairs $\mathbb{R}$.
Output: $\mathbb{C}$: Candidate string pairs $(s, t) \in SI_S \times SI_T$

```
1  ℂ ⟵ ∅;
2  for (∀ F_s in SI_S, ∀ F_t in SI_T) do
       //Fence-entry:F=<u, v, P>
3      if (min(F_s.v, F_t.v) ≥ θ × max(F_s.u, F_t.u)) then
4          for (∀ E_s∈F_s.P, ∀ E_t∈F_t.P) do
               //Leaf-entry:E=<t, P>
5              if (min(E_s.t, E_t.t) ≥ θ × max(F_s.u, F_t.u)) then
6                  for (∀ g is an overlapping signature pointed by E_s.P and E_t.P) do
7                      L_s = E_s.P→getIList(g);
8                      L_t = E_t.P→getIList(g);
9                      C = {(s, t)|s ∈ L_s, t ∈ L_t};
10                     ℂ = ℂ ∪ C;
                   end
               end
           end
       end
   end
11 return ℂ;
```

SI-Join. (See Algorithm 6) comprising three steps in the filtering phase: in the first step (lines 2 $\sim$ 3), consider two fence-entries F_s and F_t. If the maximal size of expanded sets of strings below F_s (or F_t) is still smaller than θ times the length of original strings below F_t (or F_s) (i.e., u), then all pairs of strings below F_s and F_t can be pruned safely. Similarly, in the second step (lines 4 $\sim$ 5), given two leaf entries E_s and E_t, where all expanded sets have the same size, if the size of expanded sets below E_s (or E_t) is smaller than θ times the length of original strings below E_t (or E_s), then all string pairs below E_s and E_t can be pruned safely. In the third step (lines 6 $\sim$ 10), for each signature g that is an overlapping signature pointed by $E_s.P$ and $E_t.P$, function getIList returns the I-list whose key is g. Then SI-join generates string pair candidates for every two I-lists.

Example 5.2. We use this example to illustrate the SI-join algorithm. Consider a self join on the table in Figure 7(a) with $\theta = 0.8$ and synonyms in Figure 7(b). The corresponding SI-indices are shown in Figure 7(e). SI-join prunes all the string pairs below F_1 and F_3, since $min(7, 11) < 0.8 \times max(2, 9)$ (lines 2$\sim$3). Similarly, all pairs below (E_1, E_3) can be pruned (lines 4$\sim$5). Then SI-join uses the signatures "Conference" and "Conf" to get I-lists below (E_2, E_3) (lines 7$\sim$8), since they are overlapping signatures pointed by E_2 and E_3 (line 6). Then a candidate (q_2, q_3) is generated and added to $\mathbb{C}$ (lines 9$\sim$10). The final results are (q_2, q_3) and $\{(q_i, q_i)|i \in [1, 4]\}$. To demonstrate the superiority of SI-join, we compare the numbers of candidates for two algorithms: join-baseline outputs seven candidates, while SI-join returns five candidates.

Discussions on the Updates of Synonyms. Finally, we now consider the scenario of supporting the update of synonyms, because it is likely that users perform a similarity

join on a large dataset and then realize that there are some new synonym pairs that should be added into the synonym collection. In this case, we show an incremental algorithm for avoiding rerunning algorithms against the whole datasets. Consider the following three steps. (1) Scan the tables to find the strings for which the new synonyms are applicable; and (2) the fence entries, leaf entries, and I-lists of SI-trees are updated correspondingly; and (3) rerun the SI-join algorithm, but we only consider the entries which have been updated. It is worth pointing out that the newly-added synonyms can only increase the size of results for similarity join with selective expansion, but the full expansion has no such good property, and the newly-added synonyms may decrease the similarity between two strings, and thus all affected string pairs have to be recomputed.

6. ESTIMATION-BASED SIGNATURE SELECTION

6.1. Motivation

The previous algorithms generate signatures for each string according to a global order of tokens. In theory, if N denotes the number of distinct tokens, there are $N!$ ways to permute the tokens to select signatures. A question then arises: which orders are better for the approximate join algorithms? It is impractical to provide an order which is always suitable to all datasets. Therefore, the remaining problem of our model is to study which order of tokens should be used for the two specific join tables.

To address this problem, our main idea is to use multiple ways to generate signatures offline then quickly estimate the quality of each signature filter to select the best one to perform the actual filtering. The main challenge here is two fold. First, we need to generate multiple promising filters offline. Second, the estimator should quickly select the most effective filter online.

To address the first challenge, we use the *itf*-based method [Arasu et al. 2008; Li et al. 2008] to generate filters. In particular, let *tf(w)* denote the occurrences of token w appearing in the join tables, that is, the frequency of w. Then the inverse term frequency of the token w, *itf(w)*, is defined as *1/tf(w)*. The global order is arranged in a descending order based on *itf* values. Intuitively, tokens with a high *itf* value are rare tokens in the collection and have a high probability of being chosen as signatures. Since tokens come from two sources, tables and rules, there are multiple ways to compare the frequency of tokens as follows. (1) *ITF1*: sort the tokens from data tables and synonyms separately by decreasing *itf*, then arrange tokens from tables first, followed by those from synonyms; (2) *ITF2*: sort separately again, but synonyms first, tables second; (3) *ITF3*: sort all the tokens together by a decreasing *itf* order; (4) *ITF4*: generate all possibilities of expanded sets for each string, then sort the tokens by the decreasing *itf* order in all the expanded sets.

In order to give an empirical study of the effect of different signatures on filtering power, we perform experiments with *ITF1*∼*ITF4*. For ease of presentation, we employ the number of candidates to demonstrate the filters' filtering power. The filtering power is measured by $\psi = \frac{|C|}{|N|}$, where $|C|$ denotes the number of pruned pairs and $|N|$ is the total string pairs. We then show some attractive experimental results for string-similarity joins.

We perform a set of experiments using *ITF1*∼*ITF4* against two datasets: CONF and USPS data. On the CONF dataset, we perform a join between two tables S and T using a synonym table R. Table S contains 1,000 abbreviations of conference names, while table T has 1,000 full expressions of conference names. On the USPS dataset, we perform a self join on a table containing one million records, each of which contains a person name, a street name, a city name, etc., using 284 synonyms. The detailed description of the datasets can be found in Section 8.

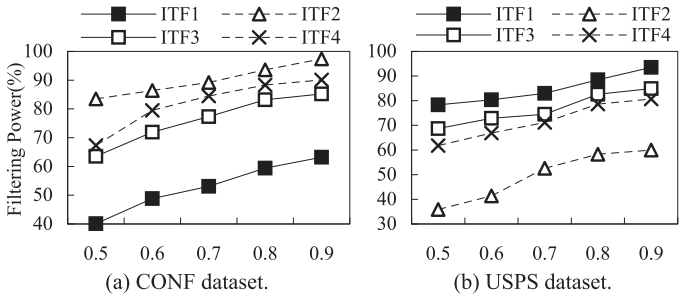


Fig. 9. Filtering powers of different signature filters.

Figure 9 reports the experimental results. We observe that (i) four signature filters achieve quite different filtering powers. For instance, on the CONF dataset, when the join threshold is 0.7, the filtering power of ITF1 is 89.2%, while that of ITF2 is only 53.1%; and (ii) no signature filter can always perform the best in each dataset. For example, ITF1 performs the best in the USPS dataset, but it performs the worst in the CONF dataset, the main reason being that ITF1 gives a higher priority for the tokens in the data table to become signatures, but there are many overlaps in these signatures in the CONF dataset, resulting in a large number of candidates.

These results suggest that it is unrealistic to find a one-size-fits-all solution to selecting signatures. Therefore, we propose an estimator to effectively estimate the filtering power of different filters in order to dynamically select the best filter.

6.2. Estimation-Based Selection Algorithms for String Similarity Joins

Let $\ell(S,g) = \{q_1^g, \ldots, q_n^g\}$ denote a list of string IDs (i.e., I-lists in Figures 6 and 7) associated with the signature token g from the table S. Then an ID pair $(q_i^g, q_j^g) \in \ell(S,g) \times \ell(T,g)$ is an element of the cross product of the two lists. Let G denote the set of all common signatures between S and T. Therefore, our problem is to estimate the cardinality of the union of all ID pairs from all signatures in G, that is, $\cup_{g \in G} \ell(S,g) \times \ell(T,g)$. Before we describe our approach towards this goal, let us examine the lower and upper bounds of this cardinality.

LEMMA 6.1. *Given two tables S and T for similarity join and a signature filter λ, the upper and lower bounds of the cardinality of the candidate pairs are*

$$UB(\lambda) = \sum_{g \in G} (\|\ell(S,g)\| \times \|\ell(T,g)\|),$$

and

$$LB(\lambda) = max\{\|\ell(S,g)\| \times \|\ell(T,g)\|, g \in G\},$$

where $\|\ell(\cdot,g)\|$ denote the number of IDs in a list and G is the set of overlapping signatures between S and T.

Example 6.2. We use this example to illustrate this lemma. Recall Figure 7(c) and assume that there are two tables S and T, where $S = \{q_1, q_2\}$ and $T = \{q_3, q_4\}$. Then $G = \{$Conference, Conf$\}$. Thus, $\ell(S,$Conference$) = \{q_1, q_2\}$, $\ell(T,$Conference$) = \{q_3\}$, $\ell(S,$Conf$) = \{q_2\}$, $\ell(T,$Conference$) = \{q_3\}$. Therefore, according to Lemma 6.1, $UB(\lambda) = 2 + 1 = 3$ and $LB(\lambda) = \max\{2,1\} = 2$.

In Lemma 6.1, the upper bound is the total number of string pairs and the lower bound is the maximal number of string pairs associated with one signature. A signature

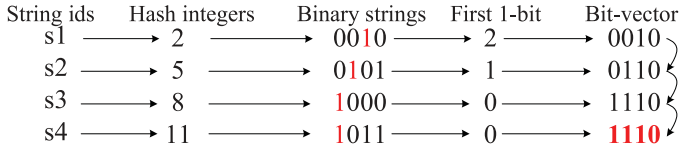


Fig. 10. An example of bit vector generated by the FM method. (The first 1-bit is the leftmost 1-bit.)

Table I. Summary of the Notations Used

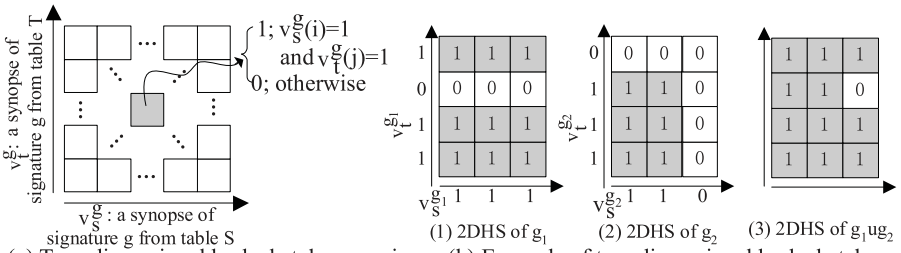
Notation	Explanation
$Sig(q, \mathbb{R})$	the signatures of q with synonyms $\mathbb{R}$
$LB(\lambda)$, $UB(\lambda)$	the lower and upper bounds of the cardinality of the candidates with the filter λ
$\ell(S, g)$	a list of string IDs associated with the signature g from the table S
μ_S	the distinct number of string IDs for all signature tokens from the table S
ν_S^g	an FM sketch for the signature g in I-lists of the table S
$2DHS^g(i, j)$	the bucket (i, j) for a signature g in a two-dimensional hash sketch synopsis (2DHS)

λ guarantees returning fewer candidates than signature λ' if $UB(\lambda) < LB(\lambda')$. Therefore, the tighter the bound, the better the results, since a tighter bound implies more accurate estimates on the ability of a filter. In the following, we introduce tighter upper and lower bounds than Lemma 6.1 by extending the Flajolet-Martin (FM) [Flajolet and Martin 1985] technique.

Flajolet-Martin Estimator. The Flajolet-Martin (FM) technique for estimating the number of distinct elements (i.e., set-union cardinality) relies on a family of hash functions $\mathcal{H}$ that map data values uniformly and independently over the collection of binary strings in the input data domain L. In particular, FM works as follows: (i) use a hash function $h \in \mathcal{H}$ to map string IDs to integers sufficiently uniformly in the range $[0, 2^{L-1}]$; (ii) initialize a bit vector ν (also called synopses) of length L to all zeros and convert the integers into binary strings of length L; and (iii) compute the position of the first 1-bit in the binary string (denoted as $p(h(i))$), and set the bit located at position $p(h(i))$ to be 1. Note that for any $i \in [0, 2^{L-1}]$, $p(h(i)) \in \{0, \ldots, \log M\text{-}1\}$ and $Pr[p(h(i)) = \ell] = \frac{1}{2^{\ell+1}}$, which means the bits in lower position have a higher probability to be set to 1. Figure 10 illustrates the FM method. Assume there are six strings. The length of the domain $L = 4$. Finally, the synopsis ν is "1110".

Our Two-Dimensional Hash Sketch (2DHS). In the following, we describe our ideas for extending the FM sketch to provide better bounds for estimating the number of candidate pairs. The main challenge is that the existing FM-based methods are only applicable to one-dimensional data, but we need to estimate the number of distinct pairs based on two independent FM sketches. Note that the straightforward method is infeasible for mapping two-dimensional data to one unique value and then using the existing FM sketch for our problem. This is because two join tables are given online, and thus we cannot perform the mapping operations offline to construct a one-dimensional sketch. Therefore, we next describe four steps for estimating the number of distinct pairs based on two predefined one-dimensional sketches. Table I summaries the notations used.

Step 1 (*Estimating the Distinct Number of One Table*). First, we estimate the distinct number of string IDs for all signature tokens from one table, that is, μ_S and μ_T, where $\mu_S = \cup_{g \in G} \ell(S, g)$ and $\mu_T = \cup_{g \in G} \ell(T, g)$. Ganguly et al. [2004] provide an (ϵ, δ)-*estimate* algorithm for μ_S (or μ_T) by using the extension of the FM technique, which estimates the quantity of μ_S (or μ_T) within a small relative error ϵ and a high confidence 1-δ.



(a) Two-dimensional hash sketch synopsis (b) Example of two-dimensional hash sketches.

Fig. 11. The structure of two-dimensional hash sketch (2DHS).

Space precludes detailed discussion about their method here; however, readers may find it in Section 3.3 of Ganguly et al. [2004]. Note that after the first step, we indeed obtain a new upper bound, $\mu_S \times \mu_T$, but it is still not tight enough. We proceed to the following steps to get a better value.

Step 2 (*Constructing a Two-Dimensional Hash Sketch*). Let ν_S^g and ν_T^g be two FM sketches of signature g in I-lists of table S and table T, respectively. We construct a two-dimensional hash sketch synopsis ($2DHS$) of signature g, which is a two-dimensional bit vector. The bucket (i, j) in a 2DHS is 1 if and only if $\nu_S^g(i) = 1$ and $\nu_T^g(j) = 1$. Further, let G be the set of common signatures of S and T. We construct a new $2DHS'$, which is the union of all 2DHS's of each signature, namely, $\cup_{g \in G} 2DHS^g$, such that $2DHS'(i, j) = 1$ if $\exists\ 2DHS^g(i, j) = 1$.

Example 6.3. Figure 11 illustrates Step 2. Assume there are two common signatures g_1 and g_2 of S and T, $\nu_S^{g_1} = \{1,1,1\}$, $\nu_T^{g_1} = \{1,1,0,1\}$ and $\nu_S^{g_2} = \{1,1,0\}$, $\nu_T^{g_2} = \{1,1,1,0\}$. Figure 11(b1) is the 2DHS of g_1 and Figure 11(b2) is the 2DHS of g_2. Then the union $2DHS$ is shown in Figure 11(b3).

Step 3 (*Computing Witness Probabilities*). Once the union *2DHS* is built, we select a bucket (i, j) to compute a witness probability as follows. We examine a collection of ω independent *2DHS* $(\nu_1^g, \nu_2^g, \ldots, \nu_\omega^g)$ (each copy using independently-chosen hash functions from $\mathcal{H}$). Algorithm 7 shows the steps for computing witness probabilities. In particular, let $R_s = \lfloor \log_2 \mu_S \rfloor$ and $R_t = \lfloor \log_2 \mu_T \rfloor$. We investigate a column of buckets (R_s, y) (line 2 $\sim$ 8), and a row of buckets (x, R_t) (line 9 $\sim$ 15), and select two smallest indexes y and x at which only a constant fraction 0.3ω of the sketch buckets turns out to be 1. As our analysis will show, the observed fraction $\hat{p}_s$ and $\hat{p}_t$ (line 16) and their positions can be used to provide a robust estimate for bounds in the next step.

Step 4 (*Computing Tighter Upper and Lower Bounds*). The final step is to compute the upper bound TUB and the lower bound TLB of a signature filter λ. Assume, without loss of generality, that $\mu_S \leq \mu_T$.

$$\mathrm{TUB}(\lambda) = \varphi_1 \cdot min(u_1, u_2), \text{ where}$$

$$u_1 = \mu_S \cdot \frac{\ln(1 - p_s)}{\ln(1 - \frac{1}{2^y})}, \text{ and } p_s = \frac{\hat{p}_s}{1 - (1 - \frac{1}{2^{R_s}})^{\mu_S}};$$

$$u_2 = \mu_T \cdot \frac{\ln(1 - p_t)}{\ln(1 - \frac{1}{2^x})}, \text{ and } p_t = \frac{\hat{p}_t}{1 - (1 - \frac{1}{2^{R_t}})^{\mu_T}}.$$

ALGORITHM 7: Computing Two Witness Probabilities

Input : ω independent two-dimensional hash sketches ($2DHS_1$, $2DHS_2, \ldots, 2DHS_\omega$), R_s and R_t.
Output: $\hat{p}_s$ and $\hat{p}_t$ and two witness positions (R_s, y) and (x, R_t)

```
1  f = 0.3ω;
2  for (y from 0 to R_t) do
3  |  count = 0;
4  |  for (i from 1 to ω) do
5  |  |  if (2DHS_i[R_s][y] == 1) then
6  |  |  |  count = count+1;
   |  |  end
   |  end
7  |  if (count ≤ f) then
8  |  |  p̂_s = count/ω and goto Line 9;
   |  end
   end
9  for (x from 0 to R_s) do
10 |  count = 0;
11 |  for (i from 1 to ω) do
12 |  |  if (2DHS_i[x][R_t] == 1) then
13 |  |  |  count = count+1;
   |  |  end
   |  end
14 |  if (count ≤ f) then
15 |  |  p̂_t = count/ω and goto Line 16;
   |  end
   end
16 return p̂_s, p̂_t and their witness positions (R_s, y), (x, R_t)
```

$$\mathrm{TLB}(\lambda) = \varphi_2 \cdot \mu_S \cdot \frac{\ln(1-p_s')}{\ln(1-\frac{1}{2^y})}, \text{ where } p_s' = \frac{\hat{p}_s - \Delta p}{1-(1-\frac{1}{2^{R_s}})^{\mu_S}}$$

$$\Delta p = \left[1-\left(1-\frac{1}{2^{R_s+y}}\right)^{\mu_S}\right] - \frac{1}{2^y} \times \left[1-\left(1-\frac{1}{2^{R_s}}\right)^{\mu_S}\right],$$

where constants $\varphi_1 = 1.43$, $\varphi_2 = 0.8$, are derived in Theorem 6.8.

Example 6.4. This example illustrates the computation of TUB and TLB. Assume $\mu_S = 100$ and $\mu_T = 200$. Thus, $R_s = \lfloor \log_2^{100} \rfloor = 6$, $R_t = \lfloor \log_2^{200} \rfloor = 7$. Assume, in Algorithm 7, that the returned values are $\hat{p}_s = 0.2$, $\hat{p}_t = 0.19$, $x = 5$, and $y = 6$. According to the formulas in Step 4, we have $p_s = 0.2522$, $p_t = 0.24$, $u_1 = 1845$, $u_2 = 1729$, $\Delta p = 0.0117$, $p_s' = 0.2375$, and the upper bound $TUB(\lambda) = 1.43 \times 1729 = 2472$ and $TLB(\lambda) = 1722 \times 0.8 = 1278$.

Therefore, one filter is selected as the best if its TUB is less than all the TLBs of other filters, which means that it returns the smallest number of candidates. In practice, if there are several filters whose bounds have overlaps, then it means that their abilities are similar with a high probability, and we can arbitrarily choose one of them to break the tie.

Theoretical Analysis. We first demonstrate that, with an arbitrarily high probability, the returned lower and upper bounds are correct; then we analyze its space and time complexities.

It is important to note that given n distinct ID pairs and a position (x, y) in a 2DHS matrix, the probability that the bucket in (x, y) is 1 depends on the distribution of the input data. To understand it, consider the following two cases: one is (1,1), (2,1), . . . , (n,1), and the other is (1,n+1), (2,n+2),. . . ,(n,2n). Both cases have n distinct pairs, but their probabilities are different. The first case is $\frac{1}{2^y} \times [1-(1-\frac{1}{2^x})^n]$, but the second case is $[1-(1-\frac{1}{2^{x+y}})^n]$, respectively. Therefore, we need to differentiate the probabilities for these two extremes. Let $MIN(N, \mu_S, \mu_T, x, y)$ and $MAX(N, \mu_S, \mu_T, x, y)$ denote the minimum and maximum probabilities, respectively, that the bucket (x, y) is 1 with N distinct pairs, where μ_S and μ_T denote the number of distinct IDs in each dimension, respectively. The computation of the accurate values of the MIN and MAX is highly complicated, but for the purpose of this article, to differentiate the ability of filters, we derive a lower (resp. upper) bound for the MIN (resp. MAX) probability.

The following lemma implies that the MIN and MAX probabilities are monotonically increasing on μ_S and μ_T. This monotonicity ensures that Lemmas 16 and 6.7 compute the lower and upper bounds.

LEMMA 6.5. *If* $\mu_S \geq \mu'_S$ *and* $\mu_T \geq \mu'_T$, *then* $MIN(N, \mu_S, \mu_T, x, y) \geq MIN(N, \mu'_S, \mu'_T, x, y)$ *and* $MAX(N, \mu_S, \mu_T, x, y) \geq MAX(N, \mu'_S, \mu'_T, x, y)$.

LEMMA 6.6. *The lower bound of the MIN probability can be computed as*

$$p_1 = \left[1-\left(1-\frac{1}{2^x}\right)^{\mu_S}\right] \times \left[1-\left(1-\frac{1}{2^y}\right)^{\frac{N}{\mu_S}}\right];$$

$$p_2 = \left[1-\left(1-\frac{1}{2^y}\right)^{\mu_T}\right] \times \left[1-\left(1-\frac{1}{2^x}\right)^{\frac{N}{\mu_T}}\right];$$

$$MIN(N, \mu_S, \mu_T, x, y) \geq max(p_1, p_2).$$

PROOF (SKETCH). According to Lemma 6.5, the MIN probability is monotonically increasing with μ_S and μ_T. Therefore, the final MIN probability is greater than p_1 and p_2, since $\mu_S \geq \frac{N}{\mu_T}$ and $\mu_T \geq \frac{N}{\mu_S}$. That concludes the proof. □

LEMMA 6.7. *Assume that* $\mu_S \leq \mu_T$. *Then the upper bound of the MAX probability can be computed as*

$$p_1 = \left[1-\left(1-\frac{1}{2^x}\right)^{\mu_S}\right] \times \left[1-\left(1-\frac{1}{2^y}\right)^{\frac{N}{\mu_S}}\right];$$

$$p_2 = \left[1-\left(1-\frac{1}{2^{x+y}}\right)^{\mu_S}\right] - \frac{1}{2^y} \times \left[1-\left(1-\frac{1}{2^x}\right)^{\mu_S}\right];$$

$$MAX(N, \mu_S, \mu_T, x, y) \leq p_1 + p_2.$$

PROOF (SKETCH). According to Lemma 6.5, with a fixed μ_S, the greater the value μ_T, the greater the MAX probability. In the worst case, $\mu_T = N$. Therefore, the MAX probability p_{max} is no greater than

$$p_{max} \leq 1 - \left[\left(1-\frac{1}{2^x}\right) + \frac{1}{2^x} \cdot \left(1-\frac{1}{2^y}\right)^{\frac{N}{\mu_S}}\right]^{\mu_S}.$$

We then prove that $p_{max} - p_1 \leq p_2$. *We can show that* $p_{max} - p_1$ *is a monotonically-decreasing function when* $N \geq \mu_S$. *Therefore, when* $N = \mu_S$ *(in this case,* $N = \mu_S = \mu_T$*),*

we have the minimal value as follows.

$$p_{max} - p_1 \le 1 - \left[\left(1 - \frac{1}{2^x}\right) + \frac{1}{2^x} \cdot \left(1 - \frac{1}{2^y}\right)\right]^{\mu_S} - \left[1 - \left(1 - \frac{1}{2^x}\right)^{\mu_S}\right] \times \left[1 - \left(1 - \frac{1}{2^y}\right)\right] = p_2. \quad \square$$

THEOREM 6.8. *Given a signature filter λ, with any high probability 1-δ, our algorithm guarantees correctly returning the upper bound and the lower bound in order to estimate the number of candidate pairs with λ.*

PROOF (SKETCH). There are three steps in this proof. Firstly, we show that the estimated probabilities $\hat{p}_s$ and $\hat{p}_t$ returned by Algorithm 7 are of low error and high confidence, and then we show that our formulas TLB and TUB are correct. Finally, we show how to adjust formulas based on the estimated $\hat{p}_s$, $\hat{p}_t$ and μ_S, μ_T.

First, in Algorithm 7, by the Chernoff bound, the estimated $\hat{p} = \text{count}/\omega$ satisfies $|\hat{p} - p| \le \epsilon p$ with a probability at least 1-δ as long as $\omega p \ge \frac{2\log(1/\delta)}{\epsilon^2}$. Let $\epsilon = 0.15$, then $\omega p \ge 88\log(1/\delta)$. Since $\hat{p}_s$ and $\hat{p}_t$ returned in Algorithm 7 are the smallest indexes, where only 0.3ω of sketches turn to be 1, their values are no less than $0.3/2 = 0.15$. Therefore, $p \ge 0.15$, and we can generate enough number of sketches such that $\omega \ge 587\log(1/\delta) = \Theta(\log\frac{1}{\delta})$. Strictly speaking, the witness probabilities $\hat{p}_s$ and $\hat{p}_t$ are estimated values, and we need to use the Chernoff bound again to show that they are close to 0.15 with a high confidence. The details are omitted here.

Second, according to Lemmas 16 and 6.7, it is not hard to mathematically compute the lower bound and upper bound to estimate the number of distinct pairs number N.

Finally, the following two lemmas demonstrate that we only need to multiply a constant in the formulas to use the low-error estimated $\hat{p}_s$, $\hat{p}_t$, μ_S, and μ_T, for the accuracy guarantee. In particular, Lemma 6.9 is used to derive φ_1 in *TUB* and Lemma 6.10 for φ_2 in *TLB*, respectively. (A similar proof technique can be found in [Bar-Yossef et al. 2002; Ganguly et al. 2004], even though our estimation is quite different from theirs.) $\square$

LEMMA 6.9. *Let $f(x) = ln(1 - \frac{x}{1-(1-\frac{1}{2^{R_s}})^{\mu_S(1+\varepsilon)}})$. If $y - x \le 0.15x$ for some $x \le 0.3$ and $\varepsilon \le 0.1$, then $f(y) - f(x) \le 0.43 f(x)$.*

PROOF. Since $1-(1-\frac{1}{2^{R_s}})^{\mu_S(1+\varepsilon)} \approx 1-e^{-(1+\varepsilon)}$ and $\varepsilon \le 0.1$, then $\frac{1}{1-e^{-(1+\varepsilon)}} \ge 1.5$. By Taylor series, there is a value $w \in (x, y)$ such that $ln(1 - 1.5y) = ln(1 - 1.5x) - 1.5(y - x)/(1 - 1.5w)$. Thus, we have $f(y) - f(x) \le \frac{1.5(y-x)}{1-1.5max(x,y)} \le \frac{0.225x}{1-1.725x} \le \frac{0.225x}{0.52} \approx 0.43x \le -0.43ln(1 - 1.5x) \le 0.43 f(x)$. $\square$

LEMMA 6.10. *Let $f(x) = ln(1 - \frac{x-\Delta p}{1-(1-\frac{1}{2^{R_s}})^{\mu_S(1+\varepsilon)}})$. If $x - y \le 0.15x$ for some $x \le 0.3$ and $\varepsilon \le 0.1$, then $f(x) - f(y) \le 0.2 f(x)$.*

PROOF (SKETCH). First, we can prove $\frac{\Delta p}{1-(1-\frac{1}{2^{R_s}})^{\mu_S(1+\varepsilon)}} < 0.71$. The details are omitted here. Then by Taylor series again, there is a value $w \in (x, y)$ such that $ln(1.71 - 1.5y) = ln(1.71 - 1.5x) - (1.5(x - y))/(1.71 - 1.5w)$. Thus, we have $|f(y) - f(x)| \le \frac{1.5(x-y)}{1.71-1.5max(x,y)} \le \frac{0.225x}{1.71-1.725x} \le \frac{0.225x}{1.19} \approx 0.189x \le 0.2ln(1.71 - 1.5x)$, since by Maclaurin series, $ln(1.71 - 1.5x) \approx ln1.71 - (1.5/1.71)x \approx 0.54 - 0.88x$. $\square$

THEOREM 6.11. *Given a high probability 1-δ, our estimator needs a total space of $\Theta(T \log M \log(1/\delta))$ bits, where T is the number of overlapping signatures and M is*

the maximal length of inverted list associated with a signature, and its computing complexity is $\Theta(T \log^2 M + logM \log(1/\delta))$.

PROOF. The size of one FM sketch is $\log M$ and there are T signatures, so the space for storing all FM sketches is $2T \log M$. We need to generate a collection of $\omega = \Theta(log\frac{1}{\delta})$ sketches. Therefore, the total space requirement is $\Theta(T \log M \log(1/\delta))$. Further, to compute the time complexity, in step 1, the computing cost is $2\omega logM$. The cost of step 2 is $2T log^2 M$ and that of steps 3 and 4 are $\omega logM$ and O(1), respectively. Therefore the total cost is bounded by $\Theta(T \log^2 M + logM \log(1/\delta))$. □

6.3. LSH Filters and Multiple Filters

Although we focus on prefix filters in the preceding discussion, our indexes and estimators can also be extended to other filters, such as the LSH scheme. In particular, each signature s in LSH is a concatenation of a fixed number k of minhashes of the set, and there are l such signatures (using different minhashes). Then, in the I-lists, each inverted list is associated with a pair (s, i), where i means that signature s comes from the ith minhash, $1 \le i \le l$. Therefore, by replacing the prefix signature g with (s, i), our SN-join (Algorithm 6) and 2DHS estimator can be directly applied on the LSH scheme. As described in Section 8.2.3, we implement both LSH and prefix filters in our algorithms to make a comprehensive comparison.

The approaches that we have mentioned so far mainly focus on one filter, but an interesting alternative is to use multiple filters to enhance the filtering power, as more rounds of filtering prune away more strings. But, by doing so, the time cost in the filtering phase would increase. Therefore, there exists a trade-off between the number of rounds of filtering and the overall running time. The following theorem formally analyzes the filtering power of k rounds of prefix filters.

THEOREM 6.12. *Given two strings s and t, and a collection of synonym pairs $\mathbb{R}$, we randomly generate k global orders for all tokens in s, t, and $\mathbb{R}$. Assume that $Sim(s, t, \mathbb{R}) < \theta$, where θ is a threshold value. Then the probability of the event that all k prefix filters can not prune (s, t), that is, the probability of the false positive, is $(1-e^{-\frac{(\theta l+1)(2l-2\theta l)}{2N}})^k$, where N denotes the total number of tokens and l refers to the maximal length of s and t.*

PROOF (SKETCH). We first estimate the expected size $E(x)$ of the intersection set of the tokens of s and t:

$$E(x) = \sum_{x=0}^{\theta l} x \times Pr[x|Sim(s, t, \mathbb{R}) < \theta] = \frac{\sum_{x=1}^{\theta l} x/(N-2l+x)!}{\sum_{x=1}^{\theta l} 1/(N-2l+x)!} \approx \frac{\theta l+1}{2}.$$

Then we consider the case that only one signature filter is applied to filter the string pair (s, t). To prune (s, t), the condition must be satisfied that none of the tokens in $s \cap t$ appears in the first $2l-2\theta l$ positions. Thus, the probability of filtering (s, t) using one filter is $\frac{(N-(\theta l+1)/2)!(N-2l+2\theta l)!}{(N-(\theta l+1)/2-2l+2\theta l)!N!}$, which can be approximated to $(1-\frac{2l-2\theta l}{N})^{(\theta l+1)/2}$ by Strling's approximation [Kaporis et al. 2006]. Then we extend it to the case that $k \ge 1$ signature filters are applied, the probability of the event that the string pair (s, t) can not be pruned through all filters, that is, the probability of the false positive, is $(1-(1-\frac{2l-2\theta l}{N})^{(\theta l+1)/2})^k$, which can be approximated to be $(1-e^{-\frac{(\theta l+1)(2l-2\theta l)}{2N}})^k$. □

In particular, assume $N = 50$, $l = 6$, and $\theta = 0.5$. If only one prefix filter is applied, that is, $k = 1$, then the probability of false positive is $1 - e^{-0.24} = 0.17$, which indicates that one round of signature filtering can filter out most irrelevant strings. As we will show in the experiments, we favor applying only one signature filter recommended by our estimator because it strikes a good balance between the number of filters and the overall running time.

7. VARIATIONS, EXTENSIONS, AND GENERALIZATIONS

In this section, we discuss interesting extensions and generalizations that can be done to our algorithms, including multiple similarity measures and the weighted tokens.

7.1. Extension to Other Similarity Measures

As mentioned in the related work section, there exists a rich set of string similarity measures. The choice of the similarity function is highly dependent on the application domain and is out of the scope of this article, but the approaches presented here can be extended to other similarity measures. We briefly comment on necessary modifications to adapt our algorithms to three other similarity measures, including *Overlap* similarity, *Dice* similarity, and *Cosine* similarity. Given two string s and t, these similarity measures are defined as follows.

—Overlap similarity is defined as $Overlap(s, t) = |s \cap t|$.

—Dice similarity is defined as $Dice(s, t) = 2\frac{|s \cap t|}{|s|+|t|}$.

—Cosine similarity is defined as $Cosine(s, t) = \frac{\vec{s} \cdot \vec{t}}{\| \vec{s} \| \cdot \| \vec{t} \|} = \frac{\Sigma_i s_i \cdot t_i}{\sqrt{|s||t|}}$.

Overlap Similarity. To adapt to Overlap similarity, the strategy of full expansion and selective expansion can be applied again, but the major changes are related to prefix filters and length filters. First, if $Overlap(s, t) > \theta$, then the first $|s| - \theta$ smallest tokens of s and the first $|t| - \theta$ smallest tokens of t must share at least one token. Therefore, in the prefix filter, the number of signatures of a string s is $|s| - \theta$. Second, given two strings s and t, if $Overlap(s, t) > \theta$, then $min(|s|, |t|) > max(|s|, |t|) - \theta$. Therefore, line 3 in Algorithm 3 will be "**if** (F.u $-\ \theta <$ L.l $<$ F.v $+\ \theta$) **then**," and line 5 will be "**if** (min(E.t, L.l) $>$ max(E.t,L.l) $-\ \theta$) **then**."

Dice Similarity. If $Dice(s, t) > \theta$, then the first $\lceil \frac{2(1-\theta)}{2-\theta}|s| \rceil$ smallest tokens of s and the first $\lceil \frac{2(1-\theta)}{2-\theta}|t| \rceil$ smallest tokens of t must share at least one token. Therefore, the number of signatures of s is $\lceil \frac{2(1-\theta)}{2-\theta}|s| \rceil$. In addition, if $Dice(s, t) > \theta$, then $max(|s|, |t|) < min(|s|, |t|)\frac{2-\theta}{\theta}$. Therefore, line 3 in Algorithm 3 will be "**if** (F.u $\cdot \frac{\theta}{2-\theta}$ $<$ L.l $<$ F.v $\cdot \frac{2-\theta}{\theta}$) **then**," and Line 5 will be "**if** (max(E.t,L.l) $<$ min(E.t,L.l) $\cdot \frac{2-\theta}{\theta}$) **then**."

Cosine Similarity. Finally, if $Cosine(s, t) > \theta$, then the first $\lceil \frac{(2-\theta^2-\theta)|s|}{2} \rceil$ smallest tokens of s and the first $\lceil \frac{(2-\theta^2-\theta)|t|}{2} \rceil$ smallest tokens of t must share at least one token. Therefore, the signatures of s contain $\lceil \frac{(2-\theta^2-\theta)|s|}{2} \rceil$ elements. Regarding the length filter, if $Cosine(s, t) > \theta$, then $min(s, t) \geq max(s, t) \times \theta^2$. Therefore, line 3 in Algorithm 3 will be "**if** (F.u $\times\ \theta^2 <$ L.l $< \frac{F.v}{\theta^2}$) **then**," and line 5 will be "**if** (min(E.t, L.l) $> max$(E.t, L.l)$\times\theta^2$) **then**."

7.2. Generalization to the Weighted Case

In this section, we extend our algorithms to the weighted case for tokens. For example, consider a string $s =$ "`TODS Journal`." It is not difficult to see that "`TODS`" should be assigned a greater weight than "`Journal`" in s. One common method for assigning

weights is to use the *itf*-based weights to generate filters described in Section 6.1. In the following, we describe how to adapt our algorithms to the weighted case. Our search and join algorithms remain the same, but we need to extend the indexing structures for prefix and length filters. Here we use weighted Jaccard (WJ) similarity (defined next) as the example and illustrate important changes. The extensions for other similarity functions are similar. Given two strings s and t,

$$WJ(s,t) = \frac{\sum_{x \in |s \cap t|} w(x)}{\sum_{x \in |s \cup t|} w(x)},$$

where $w(x)$ denotes the weight of a token x. The binary Jaccard similarity we discussed previously is just a special case when all the weights are 1.0.

To choose the global ordering of tokens, one option is to sort the tokens by decreasing order of weight. Let $w(s) = \sum_{i=1}^{|s|} w(x_i)$ and x_i be the ith token in s. Therefore, in the prefix filter, the number of signatures of a string s is

$$min \left\{ n | \sum_{i=1}^{n} w(x_i) \geq (1-\theta) \cdot w(s) \right\}.$$

We need to extend the SI-tree and FSI-tree in Section 4. A node in fence entries contains three fields $\langle u, v, p \rangle$, where u is the total weight of tokens of a string and v is the maximal weight of tokens in the fully expanded sets of strings whose length is u. Similarly, a node in leaf entries contains two fields $\langle t, p \rangle$, where t is the total weight of the tokens in the fully expanded set of a string whose length is u. After the necessary update of these structures, Algorithm 3 and 5 can be used again for this weighted tokens case.

8. EXPERIMENTS

In this section, we report an extensive experimental evaluation of algorithms on three real-world datasets. First, we show the effectiveness and efficiency of the expansion-based framework to measure the similarity of strings with synonyms. Second, we study the performance of the search-baseline and QP-search algorithms. Third, we evaluate the join-baseline, SI-join, and the state-of-the-art approaches to performing similarity joins with prefixes and LSH signatures. Finally, we analyze the performance of 2DHS estimators.

All the algorithms are implemented in Java 1.6.0 and run on a Windows XP with dual-core Intel Xeon CPU 4.0GHz, 2GB RAM, and a 320GB hard disk.

8.1. Datasets and Synonyms

Datasets. We used three datasets: U.S. addresses (*USPS*), conference titles (*CONF*), and gene/protein data (*SPROT*). These datasets differ from each other in terms of rule number, rule complexity, data size, and string length. Our goal in choosing these diverse sources is to understand the usefulness of algorithms in different real-world environments.

USPS. We download common people names, street names, city names, states, and zip codes from the United States Postal Service website (`http://www.usps.com`). We then generate one million records, each of which contains a person name, a street name, a city name, a state, and a zip code. USPS also publishes extensive information about the format of U.S. addresses from which we obtained 284 synonym pairs. The synonym pairs cover a wide range of alternate representations of common strings (e.g., "*University* → *Uni.*").

Table II. Characteristics of Datasets

Dataset	# of Strings	String Len in Words (avg/max)	# of Synonyms	# of Applicable Synonyms Per String (avg/max)
USPS	1,000,000	6.75 / 15	284	2.19 / 5
CONF	10,000	5.84 / 14	1,000	1.43 /4
SPROT	1,000,000	10.32 / 20	10,000	17.96 /68

CONF. We collected 10,000 conference names from more than 10 domains, including Medicine and Computer Science. We obtained 1,000 synonym pairs between the full names of conferences and their abbreviations by manually examining conference websites or homepages of scientists.

SPROT. We obtained one million gene/protein records from the Expasy website (`http://www.expasy.ch/sprot`). Each record contains an identifier (ID) and its name. In this dataset, each ID has $5 \sim 22$ synonyms. We generated 10,000 synonym rules describing gene/protein equivalent expressions.

Table II gives the characteristics of the three datasets.

8.2. Experimental Results

8.2.1. Quality and Efficiency of Similarity Measures. The first experiment demonstrates the effectiveness and efficiency of the various similarity measures. We compared our two measures, full expansion (*Full*) and selective expansion (*SE*), with *Jaccard* without synonyms and *JaccT* using synonyms from Arasu et al. [2008].

For each of the three datasets, we performed the experiments by conducting a similarity join between the query table T_Q and the target table T_T as follows: (1) T_Q consists of 100 manually selected full names, and (2) T_T has 200 records, where 100 of them are the correct abbreviations of the corresponding records in T_Q (i.e., the ground truth), and the other 100 "dirty" records are selected such that each of them is a similar record (in terms of Jaccard coefficient) to the corresponding records in T_Q. This is to ensure that there is only one correct matching record in T_T for each record in T_Q.

Quality of Measures. In Figure 12, we report the quality of the measures by testing the *Precision* (P), *recall* (R), and *F-measure* $= \frac{2\times P\times R}{P+R}$ (F) on three datasets. We observe the following.

—The similarity measures using synonyms (including JaccT, Full, and SE) obtain higher scores than Jaccard, which does not consider synonym pairs. The reason being that without using synonyms, Jaccard has no chance of improving the similarity.
—SE significantly outperforms JaccT in each dataset. For example, on SPROT dataset, the F-measures of SE and JaccT are 0.82 and 0.52, respectively, the main reason being that an abbreviation may have various full expressions and the join records may contain the combination of multiple expressions. Therefore, SE can apply multiple rules, while JaccT can apply only one. Note that such a situation is not rare in the real world, as one fragment of a string likely involves multiple synonym rules. We illustrate one example on each of the three datasets in Figure 12 to compare the performance of four similarity measures. For example, see CONF data in Figure 12: VLDB has two different full expressions, that is, International Conference on Very Large Databases (r_1) and Proceedings of the VLDB Endowment (r_2), and $s1$ contains these two expressions. SE applies both r_1 and r_2 to s_2 to obtain a high similarity score (i.e., 0.93), while JaccT can only apply r_2 to s_2, and the similarity is only 0.57. Assume that the join threshold is 0.8, then JaccT can not find the correct answer, while SE can.

θ	Jacc			JaccT			Full			SE		
	P	R	F	P	R	F	P	R	F	P	R	F
0.50	0.04	0.44	0.07	0.45	0.83	0.60	0.43	0.91	0.58	0.67	0.94	0.78
0.60	0.14	0.44	0.21	0.47	0.83	0.60	0.50	0.85	0.63	0.69	0.90	0.78
0.70	0.44	0.40	0.42	0.52	0.80	0.63	0.51	0.82	0.63	0.72	0.85	0.78
0.80	0.63	0.35	0.45	0.76	0.74	0.75	0.55	0.70	0.62	0.87	0.84	**0.85**
0.90	0.82	0.21	0.33	0.80	0.71	0.75	0.68	0.60	0.64	0.89	0.80	0.84

Example on CONF dataset		
Jacc	0.14	s1= "Proceedings of the VLDB Endowment 2012: 38th International Conference on Very Large Databases, Turkey" s2= "PVLDB 2012 Turkey"
JaccT	0.57	
Full	0.93	r1:PVLDB->International Conference on Very Large Databases r2:PVLDB->Proceedings of the VLDB Endowment
SE	0.93	

(a) Resulting evaluation and an examle on the CONF dataset.

θ	Jacc			JaccT			Full			SE		
	P	R	F	P	R	F	P	R	F	P	R	F
0.50	0.01	0.74	0.02	0.08	0.95	0.15	0.06	0.95	0.11	0.05	1.00	0.10
0.60	0.01	0.42	0.02	0.08	0.82	0.15	0.12	0.93	0.21	0.10	0.95	0.19
0.70	0.06	0.42	0.11	0.22	0.82	0.35	0.26	0.93	0.41	0.36	0.93	0.52
0.80	0.21	0.42	0.28	0.39	0.73	0.51	0.61	0.80	0.70	0.79	0.91	0.85
0.90	0.71	0.42	0.53	0.61	0.73	0.66	0.84	0.75	0.79	0.96	0.89	**0.92**

Example on USPS dataset		
Jacc	0.00	s1= "University of Washington 1705 NE Pacific St Seattle, WA 98195" s2= "UW"
JaccT	0.70	
Full	0.92	r1:UW->University of Washington r2:UW->1705 NE Pacific St Seattle, WA 98195 r3:UW->University of Waterloo
SE	1.00	

(b) Resulting evaluation and an example on the USPS dataset.

θ	Jacc			JaccT			Full			SE		
	P	R	F	P	R	F	P	R	F	P	R	F
0.50	0.01	0.54	0.01	0.07	0.81	0.12	0.05	0.72	0.09	0.05	0.94	0.10
0.60	0.01	0.32	0.02	0.07	0.72	0.13	0.08	0.68	0.15	0.10	0.90	0.18
0.70	0.05	0.32	0.08	0.19	0.71	0.30	0.18	0.65	0.28	0.33	0.87	0.48
0.80	0.16	0.32	0.21	0.33	0.62	0.43	0.52	0.64	0.58	0.69	0.83	0.75
0.90	0.54	0.32	0.40	0.48	0.57	0.52	0.69	0.61	0.65	0.83	0.82	**0.82**

Example on SPROT dataset		
Jacc	0.00	s1= "P93214" s2= "14339_SOLL,14-3-3 protein 9"
JaccT	0.75	
Full	0.83	r1:P93214->14339_SOLL r2:P93214->14-3-3 protein 9 r3:14339_SOLL->UPI0000124DEC
SE	1.00	

(c) Resulting evaluation and an example on the SPROT dataset.

Fig. 12. Quality of similarity measures (P: precision, R: recall, F: F-measure) and examples to illustrate the quality of similarity measures.

Dataset	1000	3000	5000	7000	9000
USPS	70.0%	71.2%	72.8%	72.2%	70.8%
CONF	62.4%	67.6%	68.9%	71.2%	70.4%
SPROT	84.4%	86.7%	88.4%	89.8%	87.3%

Fig. 13. Empirical probabilities of optimal cases for SE.

Optimality Scenarios of Algorithm 2. As described in Theorem 3.7, SE is optimal if the *rhs* tokens of useful rules are distinct. The purpose of this experiment is to verify how likely this optimal condition is satisfied in practice. We perform experiments on three datasets using different data sizes. The results are shown in Figure 13. The average percentages of optimal cases are 71.39%, 68.11%, and 87.32% in USPS, CONF, and SPROT data, respectively. Therefore, the conditions in Theorem 3.7 are likely to be met by the majority of candidate string pairs, which means that the values returned by the SE algorithm are optimal in most cases. Further, the percentage on SPROT data is larger than those on USPS and CONF. This is because many of the synonym pairs on SPROT are single-token-to-single-token synonyms, which are more likely to meet the distinct condition. In contrast, many of the synonyms of the USPS and CONF are multi-token synonyms.

Efficiency of Measures. Having verified the effectiveness of different measures, in the sequel, we assessed their efficiency. Figure 14 shows the time cost of the four measures based on the SPROT dataset (10K records) by running 10^8 times the string-similarity measurement based on the nested-loop self-join. The x-axis denotes the number of synonym rules applied on one single string, and the y-axis is the total running time. We found that although the full-expansion needs to expand the string set using rules, its performance is comparable to that of Jaccard, which does not use any rules at all. This is because the full expansion adds all applied rules directly and its computing

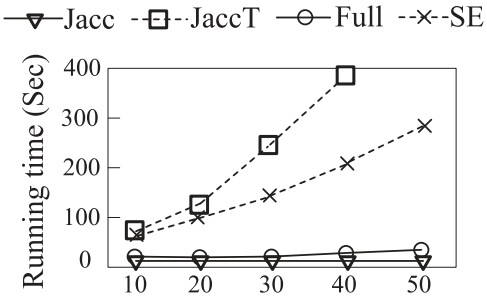


Fig. 14. Performance with varying synonyms.

Table III. Quality of Similarity Measures with Varying Noisy Synonyms

# of noisy rules	Jacc			Full			SE		
	P	R	F	P	R	F	P	R	F
0	0.63	0.35	0.45	0.55	0.70	0.62	0.87	0.84	0.85
20	0.63	0.35	0.45	0.53	0.62	0.57	0.87	0.84	0.85
40	0.63	0.35	0.45	0.57	0.60	0.58	0.86	0.82	0.84
60	0.63	0.35	0.45	0.67	0.53	0.59	0.84	0.82	0.83
80	0.63	0.35	0.45	0.74	0.45	0.56	0.84	0.80	0.82
100	0.63	0.35	0.45	0.00	0.30	0.00	0.82	0.80	0.81

Note: P: precision, R: recall, F: F-measure.

cost increases slowly with the number of rules. In contrast, JaccT needs to enumerate all the possible intermediate strings, and its performance deteriorates rapidly with the increase of rules. Finally, SE is in the "middle" of the two extremes, and avoids enumerating all the possible strings to achieve better performance than JaccT, and which selects only appropriate rules to achieve better effectiveness than full expansion in terms of F-measures (illustrated in Figure 12).

Quality of Measures with Varying Noisy Synonyms. As the last experiment for similarity measures, we studied the impact of the quality of the synonyms on similarity measures. That is, given a list of poor synonyms instead of good ones, how much would they affect the quality of the results? We investigate the quality of different measures when noisy synonyms are added. The examples of noisy rules are shown next.

—*VLDB*→*Volume location database.*
—*ICDE*→*International council for open and distance education.*

We varied the number of new noisy synonyms from 0 to 100 while fixing the threshold = 0.8 to measure the corresponding precision, recall, and F-measure for Jaccard, Full, and SE.

As shown in Table III, the precision, recall, and F-measure of Jaccard remain the same (and very low), which is expected, since Jaccard does not use any rule. For the Full measure, the recall goes down dramatically because Full applies all the possible rules, including good and poor synonyms. Finally, the precision and recall of SE are much better than Jaccard and Full, which indicates that SE is robust and insensitive to noisy synonym rules.

An interesting finding is that the precision of Full increases when the number of noisy synonyms grows from 20 to 80 but drops to zero suddenly when the number reaches 100. It can be explained that with more noisy synonyms, fewer string pairs returned from Full have higher similarity than 0.8. When the number of noisy synonyms goes

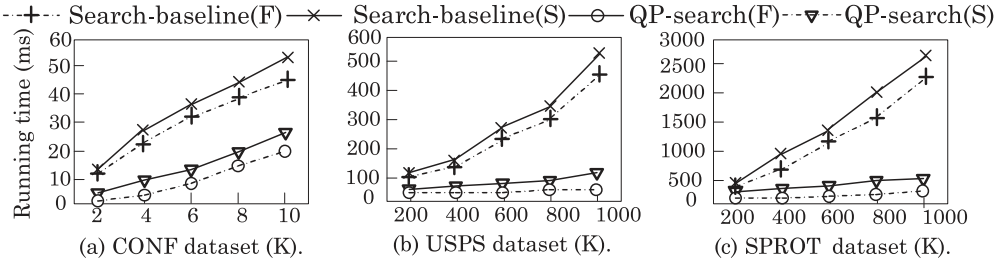


Fig. 15. Running time in three datasets of our search algorithms.

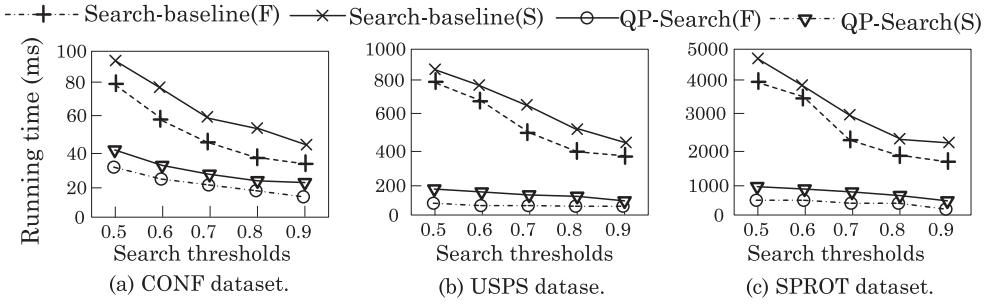


Fig. 16. Running time of our search algorithms with varying thresholds.

up to 100, none of the string pairs have similarity higher than 0.8. Therefore, at this point, the precision of Full is zero.

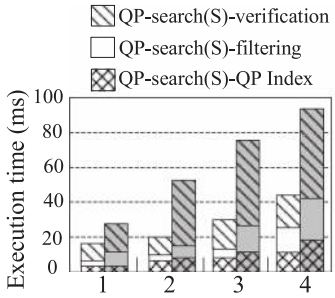
8.2.2. Efficiency and Scalability of Search Algorithms. The second set of experiments tested the efficiency and scalability of various search algorithms. We compared our algorithms Search-baseline and QP-search. For our algorithms, since they can work with the two similarity measures (i.e., full expansion, selective expansion), we denote them as search-baseline(F), search-baseline(S), and QP-search(F), and QP-search(S), respectively.

Metrics. We take the following measures: (i) the running time (including filtering time, verification time, and the time for building QP-tree for a query), and (ii) the size of candidates.

Running Time and Scalability. Figure 15 shows the running time of the four search algorithms, where the threshold is 0.9. As shown, the running times of search-baseline(F) and search-baseline(S) have an exponential growth, whereas QP-search(F) and QP-search(S) scale better (i.e., linear), the reason being that QP-search generates fewer candidates for the final verification.

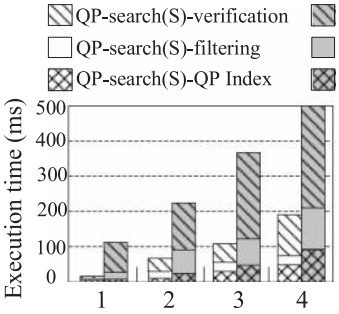
To study the scalability of algorithms with the various threshold values, we plotted Figure 16. As shown, the running time of the algorithms decreases with the growth of the threshold values. In addition, QP-search scales well with various thresholds. In contrast, when the threshold value is small, the running time of search-baseline increases significantly. For example, in Figure 16(b), when threshold = 0.9, the running time of search-baseline is about seven times more than that of QP-search. In addition, when the threshold = 0.5, the performance of QP-search is at least 15 times better than search-baseline.

We then studied the time cost of search-baseline and QP-search with different numbers of synonyms. Recall that the similarity search proceeds to generate signatures (or



	# of candidates	
	SI-search(S)	Search-baseline(S)
1	7	12
2	8	33
3	12	52
4	14	69

Fig. 17. Running time on the CONF dataset with varying numbers of applicable synonyms (# of tokens in the query is two).



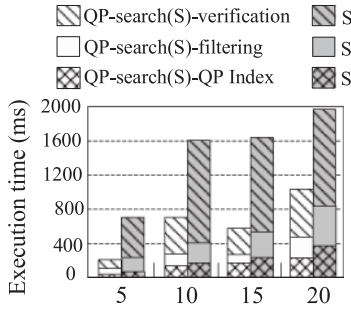
	# of candidates	
	SI-search(S)	Search-baseline(S)
1	3	88
2	25	231
3	38	871
4	135	1034

Fig. 18. Running time on the USPS dataset with varying numbers of applicable synonyms (# of tokens in the query is six).

QP-index) for a given query, and then filters the candidates by checking the overlaps of the signatures, and finally verifies the similarity. Therefore, we reported the individual execution time of signature generation, filtering, and verification in Figure 17. As seen from this figure, we observe that QP-search(S) significantly outperforms search-baseline(S) by one order of magnitude. The main reasons are as follows.

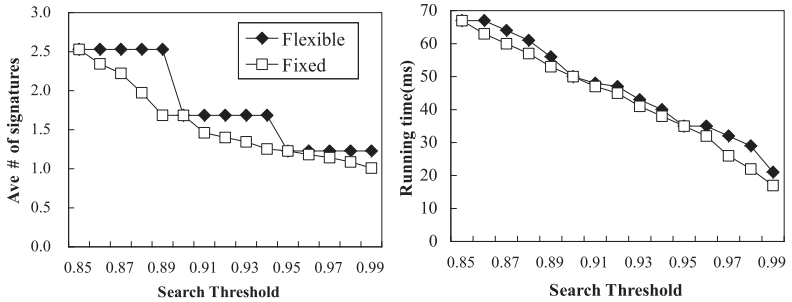
(i) Search-baseline(S) needs to unite all the signatures computed from each possible expanded set, while QP-search(S) directly utilizes the intermediate signatures. Therefore, the time for signature generation of QP-search(S) is less than that of Search-baseline(S).
(ii) For the filtering phase, QP-search(S) utilizes the SI-index and QP-index to achieve stronger filtering power than search-baseline(S). Thus, the filtering time of QP-search(S) is less than search-baseline(S). In addition, due to the powerful filtering, QP-search(S) generates fewer candidates than that of search-baseline(S). For example, in Figure 17, we report the number of candidates for each query. As shown, the number of candidates of search-baseline(S) is about one order of magnitude more than that of QP-search(S).
(iii) QP-search(S) spends less time on verifying the candidates than search-baseline(S). Therefore, the verification time of QP-search(S) is less than that of search-baseline(S).

Therefore, QP-search(S) outperforms search-baseline(S) in each of the three phases. Experiments in the USPS and SPROT datasets have a similar trend (as shown in Figure 18 and Figure 19, respectively).



	# of candidates	
	SI-search(S)	Search-baseline(S)
5	146	2938
10	1176	5723
15	833	5487
20	3216	6776

Fig. 19. Running time on the SPROT dataset with varying numbers of applicable synonyms (# of tokens in the query is two).



(a) Average number of signatures. (b) Running time with varying thresholds.

Fig. 20. Flexible search vs. fixed search on the USPS data.

Experiments for Flexible Similarity Search. All of these experiments assume that the query threshold is fixed. In this experiment, we show the experimental results based on flexible thresholds mentioned in Section 4.4. We varied the search thresholds from 0.85 to 0.99 for a total of 15 thresholds. Figure 20(b) plots the running time for two approaches based on the USPS dataset, where the fixed method means generating SI-trees for each threshold statically, thereby generating 15 SI-trees, but the flexible approach is based on only one FSI-tree. Note that the fixed approach is an ideal method and that it actually cannot handle the dynamic update of query thresholds. The curve of the flexible method nicely gets close to that of the fixed. This empirical result indicates that our algorithms are efficient for handling flexible thresholds. In addition, Figure 20(a) compares the average numbers of signatures of strings for two methods. The gap between the two approaches is small, which explains the reason for the good performance of the flexible approach.

8.2.3. Efficiency and Scalability of Join Algorithms. The third set of experiments tested the efficiency and scalability of various join algorithms. We compared our algorithms join-baseline and SI-join with the algorithm in Arasu et al. [2008] (JaccT). Our implementation of JaccT includes all the optimizations proposed in Arasu et al. [2008]. Since our algorithms can work with the two similarity measures (i.e., full expansion, selective expansion), we denote them join-baseline(F), join-baseline(S), SI-join(F), and SI-join(S), respectively. We implement all algorithms using both prefix and LSH filters. Therefore, with respect to the algorithms using the LSH scheme, we append "-LSH" to the name (e.g., JaccT-LSH denotes the JaccT algorithm using LSH scheme). Note that we use the false negative ratio $\delta \leq 5\%$, that is, the accuracy is $1 - \delta \geq 95\%$.

θ	Prefix filtering scheme				Locality sensitive hashing (LSH) scheme						
	USPS		SPROT		USPS (k=4)			SPROT(k=6)			
	JaccT	Ours	JaccT	Ours	l	JaccT	Ours	l	JaccT	Ours	
	Avg/max	Avg/max	Avg/max	Avg/max		Avg/max	Avg/max		Avg/max	Avg/max	
0.95	1.415/5	1.137/3	8.363/17	7.432/16	**2**	3.384/8	3.254/8	**3**	14.87/46	12.16/44	
0.90	1.527/6	1.447/6	8.654/18	8.322/16	**3**	4.576/19	4.499/18	**4**	19.84/58	19.44/57	
0.85	2.078/8	2.201/7	9.532/18	8.953/17	**5**	8.952/21	8.911/20	**7**	34.72/72	34.58/71	
0.80	3.780/9	2.349/7	10.79/19	9.512/19	**6**	11.14/29	11.09/27	**10**	49.62/103	48.19/100	
0.75	3.913/9	2.806/8	11.55/19	10.10/21	**8**	15.52/36	14.15/35	**16**	69.36/192	68.73/185	
0.70	4.456/10	3.568/10	13.07/20	12.97/23	**11**	20.09/51	19.21/51	**24**	75.04/297	74.92/294	

Fig. 21. Number of signatures (prefix filtering scheme and LSH scheme).

θ	USPS (100K self-join)						SPROT(1000K self-join)					
	Filtering Ratio(%)			Candidate Size(x10^6)			Filtering Ratio(%)			Candidate Size(x10^9)		
	JaccT	Join-baseline	SI-Join	JaccT	Join-baseline	SI-Join	JaccT	Join-baseline	SI-Join	JaccT	Join-baseline	SI-Join
0.95	86.1	88.8	99.0	695.4	561.3	49.8	91.8	93.5	99.5	41.3	32.7	2.6
0.90	85.7	86.8	98.9	714.3	661.3	56.9	90.7	91.7	98.6	46.5	41.9	7.1
0.85	84.5	86.0	98.9	777.6	701.9	57.4	88.5	90.0	95.5	57.5	50.2	22.1
0.80	82.5	84.7	98.7	874.0	764.2	63.0	86.6	89.3	92.1	67.7	53.4	39.5
0.75	82.3	83.7	98.7	885.8	817.6	66.7	85.2	87.5	90.3	74.0	62.4	48.9
0.70	81.9	82.4	98.4	904.0	878.7	78.5	78.1	80.3	89.7	109.5	98.8	51.3

Fig. 22. Filtering ratio with varying thresholds.

The parameters k and l should satisfy $\delta \geq (1-\theta^k)^l$ [Xiao et al. 2011]. For example, if threshold $\theta = 0.8$, $k = 3$, then l should be at least 5 to guarantee $> 95\%$ accuracy.

Metrics. We take the following measures: (i) the size of signatures, (ii) the filtering ratio of the algorithms, which is typically defined as the number of pruned string pairs divided by the total number of string pairs; and (iii) the running time (including filtering time and verification time).

Number of Signatures. We first performed experiments to report the number of signatures of a query string. This has a major impact on the query time, as both JaccT and our join algorithms need to frequently check the overlaps of string signatures. The results are reported in Figure 21. For both prefix and LSH schemes, the number of signatures of our expansion-based algorithms is smaller than that of JaccT, the reason being is that based on a transformation framework, JaccT is more likely to include new tokens into signatures than ours. In addition, we observe that the size of signatures in the LSH scheme is greater than that in the prefix scheme, and the gap increases when the threshold decreases. As we will see shortly, this results in substantial overhead of filtering time for the LSH scheme.

Filtering Power. We investigated the filtering power of different algorithms in Figures 22 and 23, using the prefix and LSH schemes, respectively. The experiments were performed on the USPS and SPROT datasets for self-join. For prefix-based algorithms, join-baseline is slightly better than JaccT, while SI-join has a substantial lead over JaccT and join-baseline. These results are mainly due to the additional filtering power in SI-join brought by length filtering and signature filtering. Next, for LSH-based algorithms, Figure 23 demonstrates a similar trend, that is, SI-join is the winner in all settings by varying parameters k and l. Compared to the prefix scheme, LSH has a slightly higher filtering ratio when parameters are set optimally (e.g., 99.2% v.s. 98.9% in USPS data, with threshold 0.9). On the other hand, note that LSH may filter away

k/L	USPS						SPROT					
	JaccT		Join-baseline		SI-Join		JaccT		Join-baseline		SI-Join	
	FR(%)	FN(%)	FR(%)	FN(%)	FR(%)	FN(%)	FR(%)	FN(%)	FR(%)	FN(%)	FR(%)	FN(%)
k=2,l=2	63.5	4.21	66.5	4.32	76.3	3.61	78.8	4.22	80.1	4.19	84.5	3.71
k=3,l=3	72.4	4.54	73.9	4.77	79.1	3.55	83.1	4.38	85.6	4.28	90.7	3.33
k=4,l=3	**87.9**	4.23	**89.4**	4.56	**99.2**	3.52	89.4	4.13	89.9	3.85	97.1	3.42
k=5,l=4	83.4	3.89	83.1	4.22	87.8	2.78	84.7	3.69	89.5	4.04	95.5	2.94
k=6,l=4	81.1	3.73	86.8	3.88	92.3	2.43	**92.3**	3.77	**93.7**	3.27	**99.8**	3.06
k=7,l=5	82.6	3.29	82.5	3.69	94.1	2.19	90.4	3.59	92.9	3.73	96.2	2.73

Fig. 23. Filter ratio (FR) and false negative (FN) with varying parameters k and l (accuracy = 95%, threshold = 0.9).

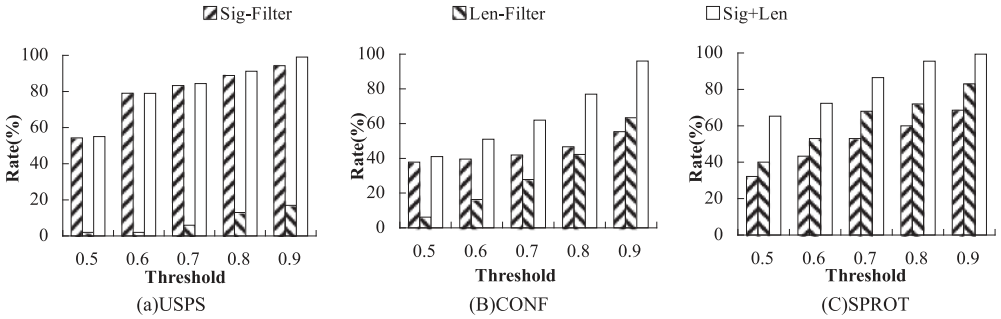


Fig. 24. Filtering rates of the signature-filter (sig-filter), the length-filter (len-filter), and the hybrid filter, that is, combining signature and length filters together (sig+len).

correct answers, resulting in false negatives, as can be seen from the false negative percentages shown in Figure 23.

Effects of Different Filters. We analyze the effects of different filters, that is, signature filters, length filters, and signature-plus-length filters. As shown in Figure 24, signature-plus-length filters obtain the strongest filtering power on all datasets, which explains why our algorithms use these two kinds of filters together. There is no absolute winner between signature filters and length filters. In particular, signature filters have stronger filtering power on the USPS dataset (Figure 24(a)), while the opposite situation occurs in the SPROT dataset (Figure 24(c)), that is, length filters beat signature filters. On the CONF dataset (Figure 24(b)), when the threshold value is small, signature filters win over length filters. However, when the threshold value is high, length filters beat signature filters. Therefore, the individual effects of signature filters and length filters depend on the datasets and the thresholds, and the combined structure of both (i.e., SI-tree) achieves the best results.

Running Time and Scalability. Figure 25 and Figure 26 show the running time of five join algorithms based on prefix and LSH schemes, respectively, where the join threshold is 0.8. The x-axis represents the join data size. As shown, the running times of both JaccT and join-baseline (i.e., join-baseline(F), join-baseline(S)) have exponential growth, whereas SI-join (i.e., SI-join(F), SI-join(S)) scales better (i.e., linear) than join-baseline and JaccT. The reason being that SI-join methods have more efficient filtering strategies and generate smaller sizes of candidates. In addition, in order to study the scalability of algorithms with the increase of the number of rules, we plot Figure 27, where SI-join(F) (i.e., SI-join with the full expansion) is a clear winner in algorithms using rules: it is insensitive to the number of rules and thus able to outperform other methods when one string involves more than 10 rules.

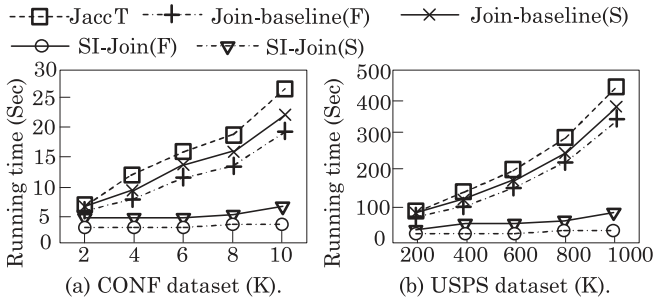


Fig. 25. Running time in CONF and USPS datasets.

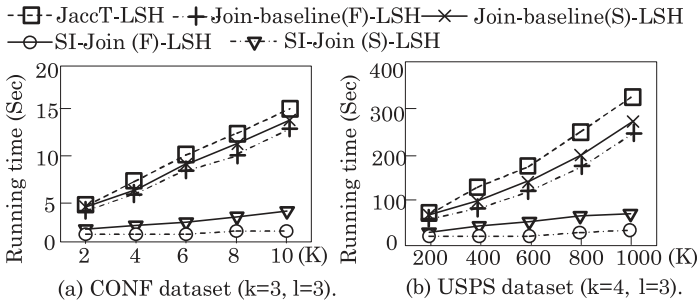


Fig. 26. Performance on LSH signature scheme.

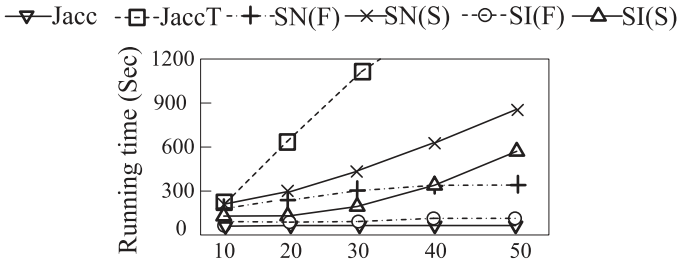
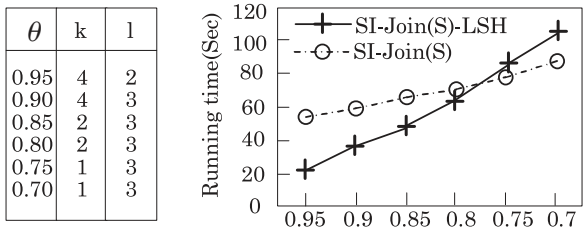


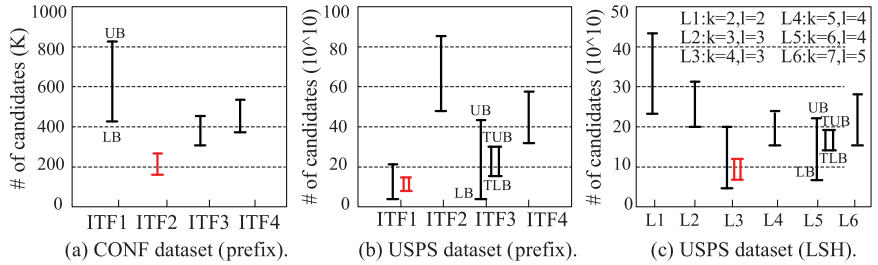
Fig. 27. Performance with varying synonyms.

Prefix vs. LSH. We then seek to analyze system performance with different signature schemes, namely, prefix vs. LSH. Figure 28 plots the running times of SI-join(S)-LSH (LSH) and SI-join(S) (prefix) with varying join thresholds. Results on SI-join(F) and join-baseline are similar. Note that, for SI-join(S)-LSH, the choice of k and l has substantial impact on the query time; hence we tune and use the parameters of k and l for each threshold to achieve the best running time. Before we describe the results, it should be noted that the two schemes, LSH and prefix, have different goals and probably different application scenarios, as LSH is an approximate solution, but prefix must return all correct answers, making it somewhat unfair for a direct comparison between them. We observe that LSH is faster than prefix when the threshold is approximately ≥ 0.8. This is mainly because more than one minhash signature is combined (i.e, $k \geq 2$) in these settings, which makes the LSH method quite selective. However, when the threshold is less than 0.8, we try all reasonable combinations of k and l values and still cannot find a setting to beat prefix, because when the threshold is small, in order to improve the filtering power, we need to select a large k, which in turn requires



(a) k, l settings. (b) USPS dataset.

Fig. 28. Prefix scheme vs. LSH scheme (with varying threshold values).



(a) CONF dataset (prefix). (b) USPS dataset (prefix). (c) USPS dataset (LSH).

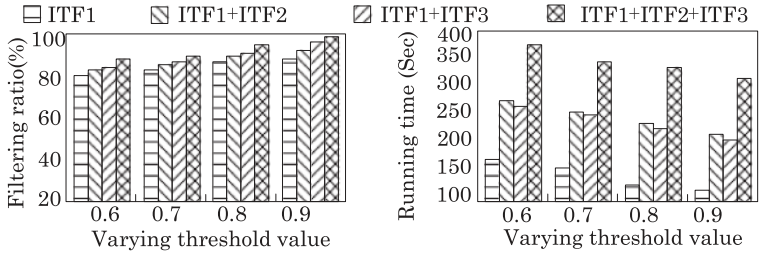
Fig. 29. Estimation results of 2DHS synopsis for prefix and LSH schemes (threshold = 0.90).

a large l to maintain the same confidence level ($\geq$ 95%); this results in substantial overhead in filtering time. Therefore, LSH trades the filtering time for higher filtering power. Then the overall running time of LSH is greater than that of prefix in such a case. This is also the reason why the best running time achieved for threshold < 0.8 is with small k and l values to reduce the overall running time in Figure 28.

8.2.4. Estimation-Based Signature Selection. The fourth set of experiments studies the quality of 2DHS synopsis on prefix and LSH schemes. For the prefix filter, we use the four frequency-based signature filters (i.e., ITF1∼ITF4), described in Section 6. For the LSH scheme, we vary parameters k and l to generate six filters. The offline processing time for the generation of all signatures of one filter is around 200s $\sim$ 300s, which is a one-time cost.

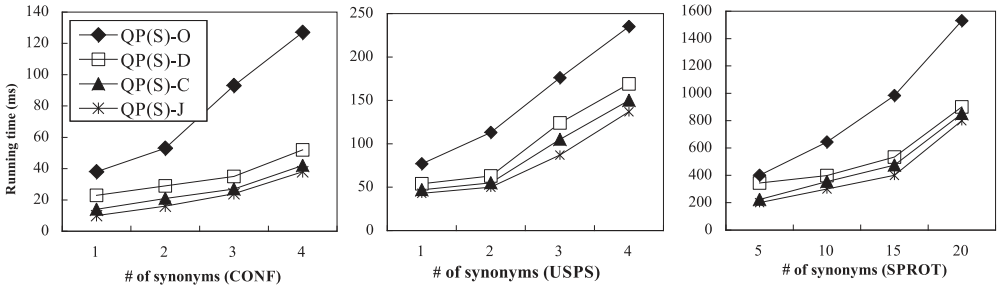
We first compute the upper bound (UB) and the lower bound (LB) of each filter by Lemma 6.1. One filter is returned as the best if its UB is less than all the LBs of other filters, which means that it returns the smallest number of candidates. For example, in Figure 29(a), ITF2 is the best in the CONF dataset. If there is no filter which can beat all other filters using only UB and LB, then we further compute the tighter upper bound (TUB) and lower bound (TLB) using 2DHS synopsis. Consequently, in Figure 29(b), ITF1 is the best filter in the USPS dataset, as its TUB is less than all others TLBs. Note that our estimate correctly predicts the ability of four filters, which can be empirically verified in Figure 9 for the prefix scheme. In addition, the estimate cost accounts for only 1% of the join time. For example, in USPS dataset, the time cost of our estimator is only 0.81s, while the filtering time is 83.7s and the total time is 113.4s.

8.2.5. One Filter versus Multiple Filters. We perform experiments to empirically compare the performances of multiple signature filters versus one signature filter. Figures 30(a) and 30(b) depict the filter ratio and the elapsed time for varied combinations of filters on the USPS dataset. The results demonstrate that more filters, of course, have greater



(a) Filtering power on the USPS dataset. (b) Running time on the USPS dataset

Fig. 30. Performance of combined signature filters.



(a) Running time on CONF. (b) Running time on USPS. (c) Running time on SPROT.

Fig. 31. Performance of different similarity measures.

ability to filter strings, but it also incurs a significant time cost. For example, the combination of three filters ITF1+ITF2+ITF3 can filter away about 99% string pairs (if the join threshold = 0.9), but in this case, the running time is 320s, which is significantly greater than 113*s* with only one filter, ITF1, which is recommended by our estimator. Therefore, this result verifies the analysis of Theorem 6.12, that one effective filter beats multiple filters, as it strikes an appropriate balance between time cost and filtering ratio, thus improving the efficiency of the whole processing.

8.3. Extensions

The final set of experiments focuses on the extensions about other similarity measures and the weighted tokens, which are discussed in Section 7.

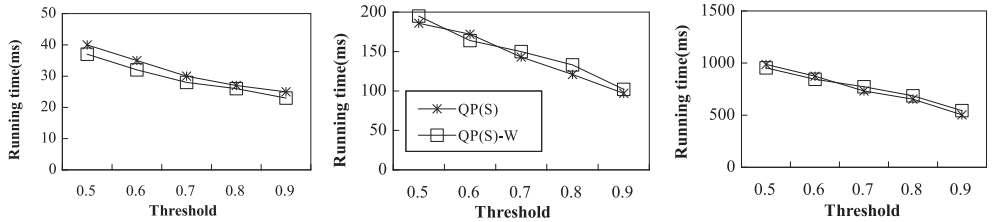
Other Similarity Measures. Experiments are conducted to empirically compare the performance of four similarity measures, that is, *overlap*, *dice*, *cosine* (as mentioned in Section 7.1), and *Jaccard*. We compare four algorithms: QP(S)-O, QP(S)-D, QP(S)-C, and QP(S)-J based on QP-search, where QP(S)-O uses the overlap measure, QP(S)-D applies the dice measure, QP(S)-C utilizes the cosine measure, while QP(S)-J employs the Jaccard measure. In Figure 31, we present the running time of these four algorithms in three different datasets with threshold $\theta = 0.9$ (for dice, cosine, and Jaccard) and $\theta = 5$ (for overlap). As shown, QP(S)-J performs the best, since the average number of signatures of QP(S)-J is the smallest. In contrast, QP(S)-O, having a large size of signatures, takes the longest to output its answers.

An interesting finding is that the precision of *Full* increases when the number of noisy synonyms grows from 20 to 80, but drops to zero suddenly when the number reaches 100. It can be explained that with more noisy synonyms, fewer string pairs returned from Full have higher similarity than 0.8. When the number of noisy synonyms goes

Table IV. Quality of Similarity Measures on Weighted Tokens

Threshold θ	Full			Full(W)			SE			SE(W)		
	P	R	F	P	R	F	P	R	F	P	R	F
0.50	0.43	0.91	0.58	0.49	0.93	0.64	0.67	0.94	0.78	0.74	0.96	0.84
0.60	0.50	0.85	0.63	0.57	0.91	0.70	0.69	0.90	0.78	0.76	0.94	0.84
0.70	0.51	0.82	0.63	0.59	0.86	0.70	0.72	0.85	0.78	0.80	0.90	0.85
0.80	0.55	0.70	0.62	0.64	0.74	0.69	0.87	0.84	0.85	0.88	0.86	0.87
0.90	0.68	0.60	0.64	0.75	0.67	0.71	0.89	0.80	0.84	0.92	0.84	0.88

Note: P: precision, R: recall, F: F-measure.



(a) CONF dataset. (b) USPS dataset. (c) SPROT datasest.

Fig. 32. Running time in three datasets of our search algorithms on weighted tokens.

up to 100, none of the string pairs have similarity higher than 0.8. Therefore, at this point, the precision of Full is zero.

Measures with Weighted Tokens. We investigate the quality and performance of the measures with TF-IDF weighted tokens. Table IV shows the quality of the unweighted measurements (Full and SE) and weighted measurements (Full(W) and SE(W)). As expected, the precision, recall and F-measure of Full(W) (resp. SE(W)) are higher than Full (resp. SE), since rare tokens are assigned with a higher weight. In addition, Full(W) improves Full by around 10%, while SE(W) only improves 5% against SE. This is because the irrelevant applicable rules in Full(W) now have less impact than that in Full, which thereby significantly improves the quality of Full(W).

In Figure 32, we report the running time of QP algorithm (with selective expansion) on weighted and unweighted tokens, that is, QP(S) and QP(S)-W, with varying thresholds. As shown, QP(S) and QP(S)-W obtain similar performances. Note that QP(S)-W is faster than QP(S) in some scenarios, because QP(S)-W may produce fewer candidates than QP(S) due to the difference of prefix filtering conditions (as shown in Section 7.2).

Summary. Finally, we summarize the main findings.

(1) Considering four similarity measures, we observe that Jaccard is inadequate due to its complete neglect of synonym rules. JaccT is not efficient because it enumerates all transformed strings and entails large query overhead. Full-expansion is extremely efficient, but its F-measure is not as good as selective-expansion, which makes a good balance between effectiveness and efficiency.
(2) With respect to the similarity searches and joins, QP-search and SI-join are the winners in the settings, respectively, over the previous algorithms (i.e., JaccT) and baseline methods. Note that we achieve a speedup of up to 50∼300x on three datasets over the state-of-the-art approach.
(3) Finally, 2DHS synopsis offers very good results and is extremely efficient at estimating the filtering power of different filters, (<1 second, accounting for only 1% of the total running time), which strongly motivates its application in practice.

9. CONCLUSION AND FUTURE WORK

In this article, we studied a set of efficient algorithms to boost the quality of approximate string matching with synonyms. We present an expansion-based framework to measure string similarities with synonyms and then several indexing structures and algorithms for similarity searches and joins. We proposed an estimator for selecting signatures online to optimize the efficiency of signature filters in algorithms. This estimator provides provably high-confidence and low-error estimates to compare the filtering power of filters with the logarithmic space and time complexity. The extensive experiments demonstrated the advantages of our proposed approaches over state-of-the-art methods.

Substantial room for future work exists. For example, it would be interesting to explore the context of words to select synonyms for expansion. For instance, the rule "*UW* → *University of Waterloo*" is more appropriate than "*UW* → *University of Washington*" if "*Canada*" appears in the context. Another direction is exploring a deeper investigation of the different similarity functions. Exciting followup research could be centered around *Levenshtein distance*. Finally, exploring other approximate algorithms for selective expansion measures is also fertile ground for further research to provide better worst-case guarantee.

APPENDIX

A. PROOF OF THEOREM 3.3

To prove *Selective-Jaccard* is NP-hard, we show that the decision version of the problem, which is given two sets S_1 and S_2, a rule set $\mathbb{R}$, and a threshold τ, we asks whether the Jaccard similarity between the expanded sets S_1' and S_2' can be $\geq \tau$, is NP-complete. To do this, we will prove that the decision version $\in NP$ and that all NP-problems are polynomial-time reducible to it. The first part is easy: the certificate is simply a subset of applicable synonym pairs. To prove the second part, we show that 3SAT is polynomial-time reducible to it.

We start the construction with a 3cnf-formula φ containing m clauses: $\varphi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \ldots \wedge (a_m \vee b_m \vee c_m)$, where each a, b, and c is a literal x_i or $\neg x_i$. Let $x_1, x_2, \ldots, x_n$ be the n variables of φ.

Now we show how to convert φ to an instance of the Selective-Jaccard problem polynomially. The construction of S_1 and S_2 is simple: $S_1 = \{l_1, l_2, \ldots, l_n\}$, $S_2 = \{u_1, u_2, \ldots, u_m, v_1, v_2, \ldots, v_n\}$. The threshold $\tau = \frac{n+m}{3n+m}$.

The rule set $\mathbb{R}$ is constructed as follows. For each variable x_i and $\neg x_i$, if x_i ($\neg x_i$) does not appear in any clause, then generate a rule $\{l_i\} \rightarrow \{x_i\ (\overline{x_i}),\ v_i\}$; otherwise assuming x_i appears in the jth clause, generate $\{l_i\} \rightarrow \{x_i\ (\overline{x_i}),\ v_i, u_j\}$.

To prove that this reduction works, we need to show that φ is satisfiable if and only if Jaccard$(S_1', S_2') \geq \tau$. We start with a satisfying assignment through which φ is true. For each variable x_i, if it is assigned true and appears in the jth clause, the rule $\{l_i\} \rightarrow \{x_i, v_i, u_j\}$ will be chosen. And if x_i does not appear in any clause, we use $\{l_i\} \rightarrow \{x_i, v_i\}$. Similarly, if x_i is assigned false, we can choose the corresponding synonym pairs of $\overline{x_i}$. S_1 can be expanded to $S_1' = \{l_1, \ldots, l_n, u_1, \ldots, u_m, v_1, \ldots, v_n, x_1(\overline{x_1}), \ldots, x_n(\overline{x_n})\}$, and S_2 can not be expanded; thus, Jaccard$(S_1', S_2) = \frac{n+m}{3\times n+m} = \tau$.

Second, if Jaccard$(S_1', S_2) \geq \tau$, we show that φ is satisfiable. From the construction, we can conclude that to make the similarity $\geq \tau$, we should add more v_i and u_j elements into S_1 and meanwhile add less x_i or $\overline{x_i}$ elements. In fact, if any u_j is not added to S_1, the similarity is at most $\frac{n+m-1}{3\times n+m}$. If any v_i is not added, the similarity is at most $\frac{n+m-1}{3\times n+m-1}$. If both x_i and $\overline{x_i}$ are added to S_1, the similarity is at most $\frac{n+m}{3\times n+m+1}$. Thus, we can conclude that if a proper expansion exits, it just brings all v_i or u_j elements and at most n

different x_i or $\overline{x_i}$ elements. Thus, we assign $x_i \in \varphi$ true if x_i appears in S_1'. Otherwise, x_i is false. As all the u_j elements are added, each clause in φ is true, and φ is satisfiable, which concludes the proof. □

B. PROOF OF THEOREM 3.7

Let $\mathcal{R}^*$ denote the rule set used by the optimal algorithm. Let $\mathcal{R}$ denote the set $\mathcal{C}_1 \cup \mathcal{C}_2$ returned in line 1 of Algorithm 2. Then we will prove that $\mathcal{R}^* = \mathcal{R}$ as follows. Given an applicable rule r_i in $\mathcal{R}$, let G_i denote the useful elements, U_i denote the new elements, and let $Z_i = U_i - G_i$. Let θ^* and θ denote the full expansion similarity using $\mathcal{R}^*$ and $\mathcal{R}$, respectively. Note that all the rules in both $\mathcal{R}$ and $\mathcal{R}^*$ are useful. When all the *rhs* tokens are distinct, $\theta = \frac{|S_1 \cap S_2 + \sum G_i|}{|S_1 \cup S_2 + \sum Z_i|}$. For a rule $r^* \in \mathcal{R}^*$, we have $\frac{G^*}{Z^*} > \theta^* \geq \theta$, that is, $\frac{G^*}{U^*} > \frac{\theta}{1+\theta}$, that is, $RG(r^*) > \frac{\theta}{1+\theta}$, which means that r^* will not be removed in Procedure *findCandidateRuleSet* (line 11), and $r^* \in \mathcal{R}$. Thus, $\mathcal{R}^* \subseteq \mathcal{R}$.

We then prove $\mathcal{R} \subseteq \mathcal{R}^*$ by contradiction. Assume that there exists a rule $r \in R$ but $r \notin R^*$, $\theta = \frac{|S_1 \cap S_2 + \sum G_i^* + G_i|}{|S_1 \cup S_2 + \sum Z_i^* + Z_i|}$. According to Algorithm 2, $\frac{G_i}{Z_i} > \theta$. Thus, $\theta > \frac{|S_1 \cap S_2 + \sum G_i^*|}{|S_1 \cup S_2 + \sum Z_i^*|} = \theta^*$, which contradicts to the optimal property and such r does not exist. Thus, $\mathcal{R} \subseteq \mathcal{R}^*$.

Finally, we show that all the candidate rules in $\mathcal{C}_1 \cup \mathcal{C}_2$ will be used in Procedure *expand* (line 2). If all the *rhs* tokens are distinct, the rule gain value will not change. The rule gain value of each rule is greater than zero, and therefore all the rules will be used by SE in the expanding phase, which concludes the proof. □

REFERENCES

N. Alon, Y. Matias, and M. Szegedy. 1996. The space complexity of approximating the frequency moments. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC'99)*. 20–29.

A. Arasu, S. Chaudhuri, and R. Kaushik. 2008. Transformation-based Framework for Record Matching. In *Proceedings of the IEEE 24th International Conference on Data Engineering (ICDE'08)*. 40–49.

A. Arasu, S. Chaudhuri, and R. Kaushik. 2009. Learning string transformations from examples. *Proc. VLDB* 2, 1, 514–525.

A. Arasu, V. Ganti, and R. Kaushik. 2006. Efficient exact set-similarity joins. In *Proceedings of the VLDB Conference (VLDB'06)*. 918–929.

Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. 2002. Counting distinct elements in a data stream. In *Proceedings of the 6th International Workshop on Randomization and Approximation Techniques (RANDOM'02)*. 1–10.

R. J. Bayardo, Y. Ma, and R. Srikant. 2007. Scaling up all pairs similarity search. In *Proceedings of the 16th International World Wide Web Conference (WWW'07)*. 131–140.

M. Bilenko and R. J. Mooney. 2003. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03)*. 39–48.

P. Bille. 2012. Faster approximate string matching for short patterns. *Theory Comput. Syst.* 50, 3, 492–515.

A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. 1997. Syntactic clustering of the Web. *Comput. Netw.* 29, 8–13, 1157–1166.

S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. 2003. Robust and efficient fuzzy match for online data cleaning. In *Proceedings of the SIGMOD Conference*. 313–324.

S. Chaudhuri, V. Ganti, and R. Kaushik. 2006. A primitive operator for similarity joins in data cleaning. In *Proceedings of the IEEE 22nd International Conference on Data Engineering (ICDE'06)*. 5–16.

S. Chaudhuri and R. Kaushik. 2009. Extending autocompletion to tolerate errors. In *Proceedings of the SIGMOD Conference*. 707–718.

P. Christen. 2012. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Trans. Knowl. Data Eng.* 24, 9, 1537–1555.

W. W. Cohen, P. D. Ravikumar, and S. E. Fienberg. 2003. A comparison of string distance metrics for name-matching tasks. In *Proceedings of the IJCAI-03 Workshop on Information Integration (IIWeb'03)*. 73–78.

M. Farach. 1997. Optimal suffix tree construction with large alphabets. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS'97)*. 137–143.

P. Flajolet and G. N. Martin. 1985. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.* 31, 2, 182–209.

S. Ganguly, M. N. Garofalakis, and R. Rastogi. 2004. Tracking set-expression cardinalities over continuous update streams. *VLDB J.* 13, 4, 354–369.

L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. 2001. Approximate string joins in a database (almost) for free. In *Proceedings of the VLDB Conference (VLDB'01)*. 491–500.

M. Hadjieleftheriou, X. Yu, N. Koudas, and D. Srivastava. 2008. Hashed samples: Selectivity estimators for set similarity selection queries. *Proc. VLDB* 1, 1, 201–212.

Y. Huang and G. R. Madey. 2004. Web data integration using approximate string join. In *Proceedings of the 13th International World Wide Web Conference (Alternate Track Papers & Posters)*. 364–365.

K. Iwama and S. Tamaki. 2004. Improved upper bounds for 3-SAT. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'04)*.

A. C. Kaporis, C. Makris, S. Sioutas, A. K. Tsakalidis, K. Tsichlas, and C. D. Zaroliagis. 2006. Dynamic interpolation search revisited. In *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming (ICALP'06)*. 382–394.

G. Kondrak. 2005. *N*-gram similarity and distance. In *Proceedings of the 12th International Conference on String Processing and Information Retrieval (SPIRE'05)*. 115–126.

H. Lee, R. T. Ng, and K. Shim. 2009. Power-law based estimation of set similarity join size. *Proc. VLDB* 2, 1, 658–669.

H. Lee, R. T. Ng, and K. Shim. 2011. Similarity join size estimation using locality sensitive hashing. *Proc. VLDB* 4, 6, 338–349.

C. Li, J. Lu, and Y. Lu. 2008. Efficient merging and filtering algorithms for approximate string searches. In *Proceedings of the IEEE 24th International Conference on Data Engineering (ICDE'08)*. 257–266.

G. Li, D. Deng, J. Wang, and J. Feng. 2012. Pass-join: A partition-based method for similarity joins. In *Proceedings of the VLDB Conference (VLDB'12)*.

J. Lu, C. Lin, W. Wang, C. Li, and H. Wang. 2013. String similarity measures and joins with synonyms. In *Proceedings of the International Conference on Management of Data (SIGMOD/PODS'13)*. 373–384.

D. R. H. Miller, T. Leek, and R. M. Schwartz. 1999. A hidden markov model information retrieval system. In *Proceedings of the 22nd Annual International ACMSIGIR Conference on Research and Development in Information Retrieval (SIGIR'99)*. 214–221.

J. Qin, W. Wang, Y. Lu, C. Xiao, and X. Lin. 2011. Efficient exact edit similarity query processing with the asymmetric signature scheme. In *Proceedings of the SIGMOD Conference*. 1033–1044.

G. Salton and C. Buckley. 1988. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.* 24, 5, 513–523.

S. Sarawagi and A. Kirpal. 2004. Efficient set joins on similarity predicates. In *Proceedings of the SIGMOD Conference*. 743–754.

Y. Tsuruoka, J. McNaught, J. Tsujii, and S. Ananiadou. 2007. Learning string similarity measures for gene/protein name dictionary look-up using logistic regression. *Bioinformatics* 23, 20, 2768–2774.

J. Wang, G. Li, and J. Feng. 2012. Can we beat the prefix filtering?: An adaptive framework for similarity join and search. In *Proceedings of the SIGMOD Conference*. 85–96.

W. E. Winkler. 1999. *The state of record linkage and current research problems*. Tech. rep., Statistical Research Division, U.S. Census Bureau.

C. Xiao, W. Wang, and X. Lin. 2008. Ed-join: An efficient algorithm for similarity joins with edit distance constraints. *PVLDB* 1, 1, 933–944.

C. Xiao, W. Wang, X. Lin, J. X. Yu, and G. Wang. 2011. Efficient similarity joins for near-duplicate detection. *ACM Trans. Datab. Syst.* 36, 3, 15.

Received September 2014; revised April 2015; accepted June 2015