String Similarity Measures and Joins with Synonyms

Jiaheng Lu †, Chunbin Lin †, Wei Wang ‡, Chen Li #, Haiyong Wang †
†School of Information and DEKE, MOE, Renmin University of China; Beijing, China
‡University of New South Wales,Australia, #University of California, Irvine, CA
†{jiahenglu,chunbinlin,why}@ruc.edu.cn, ‡weiw@cse.unsw.edu.au, #chenli@ics.uci.edu

ABSTRACT

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 similar. Most existing work that computes the similarity of two strings only considers syntactic similarities, e.g., number of common words or q-grams. While these are indeed indicators 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". Given a collection of predefined *synonyms*, the purpose of the paper is to explore such existing knowledge to evaluate string similarity measures more effectively and efficiently, thereby boosting the quality of string matching.

In particular, we first present an expansion-based framework to measure string similarities efficiently while considering synonyms. Because using synonyms in similarity measures is, while expressive, computationally expensive (NP-hard), we propose an efficient algorithm, called *selective-expansion*, which guarantees the optimality in many real scenarios. We then study a novel indexing structure called SI-tree, which combines both signature and length filtering strategies, for efficient string similarity joins with synonyms. We develop an estimator to approximate the size of candidates to enable an online selection of signature filters to further improve the efficiency. 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 results from an empirical study of the algorithms verify the effectiveness and efficiency of our approach.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Textual Databases*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Search Process*

Keywords

Similarity Search, Similarity Join, Filter Estimation

1. INTRODUCTION

Approximate string matching is used heavily in areas such as data integration [2, 4], data deduplication [9], and data mining [10]. Most existing work that computes the similarity of two strings only considers syntactic similarities, e.g., number of common words or q-grams. Although syntactic similarities are indeed indicative, 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", "Big Apple" is a synonym of "New York", and "Database Management Systems" can be abbreviated as "DBMS". Such equivalence information can help us identify semantically similar strings that may have been missed by syntactical similarity-based approaches.

In this paper, we study two problems related to approximate string matching with synonyms[1]. Firstly, how to define an effective similarity measure between two strings based on the synonyms? The traditional similarity functions cannot be easily extended to handle the synonyms in similarity matching. Secondly, 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? The two problems are closely related, because the similarity measures defined in the first problem will naturally provide a join predicate to the second.

We give several applications about approximate string measures and joins with synonyms to shed light on the importance of these two problems. In a gene/protein databases, one of the major obstacles that hinders the effective use is term variation [27], 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 join is typically 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.

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 [29], Hamming distance [17], Episode

[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
q1	Intl WH Conf 2012
q2	VLDB
...	...

(a) Table Q

ID	Name
t1	Very Large Scale Integeration
t2	Intl Wireless Health Conference 2012 UK
...	...

(b) Table T

ID	Synonym pairs	ID	Synonym pairs
r1	WH $\rightarrow$ Wireless Health	r2	Conference $\rightarrow$ Conf
r3	Intl $\rightarrow$ International	r4	UK $\rightarrow$ United Kingdom
r5	Wireless Health $\rightarrow$ WH	r6	Conf $\rightarrow$ Conference
...	...	...	...

(c) Transformation synonym pairs

Figure 1: An example to illustrate similarity matching with synonyms

distance [11], Cosine metric [25], Jaccard Coefficient [9, 20], and Dice similarity [6]. Unfortunately, none of them can be easily extended to leverage synonyms to improve the quality of similarity measurement. One exception is the JaccT technique proposed by Arasu et al. [2], which generates all possible strings by applying synonym rewriting to both strings, and returns the maximum similarity among the transformed pairs of strings, as illustrated by the following example.

EXAMPLE 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`*".*

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

The main limitation in the above 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 above 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 [2] 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 paper 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, i.e., $S = S \cup rhs$. The similarity between two strings are the similarity between their expanded token sets. We have two key ideas here: (i) We can avoid the exponential computational complexity by expanding strings using *all* the applicable synonym pairs; (ii) it costs less to perform expanding operation than transforming operation, as illustrated below.

EXAMPLE 2. *(Continue the previous example.) We apply all the applicable synonym pairs to q_1 and t_2 to expand their token sets, and then evaluate the Jaccard similarity between the two expanded token sets. For instance, the fully expanded set 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 above 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 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 appears 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 3.2). We empirically show that the condition is likely to be true in practice — the percentage of cases that such optimality hold in our experiments with three real-world data sets ranges between 70.0% and 89.8%. Figure 2 summarizes the results of the two new similarity algorithms, contrasted with results of existing methods [2].

Given the newly proposed similarity measures, it is imperative to study an efficient similarity join algorithm. In this paper, we first show a signature-based join algorithm, called *SN-Join*, by following the *filter-and-verification* framework (Section 4). It generates a set of tokens as signatures for each string using synonyms, finds candidate pairs that overlaps in the signatures, and finally verifies against the threshold using the above proposed similarity measure. In addition, we further propose the *SI-Join*, which speeds up the performance by using a native index, *SI-tree*, to reduce the candidate size with integrated signature and length filtering. Our join algorithms are also general enough to work with signatures generated by either prefix filtering or locality sensitive hashing (LSH).

The methods for selecting signatures of strings may significantly affect the performance of algorithms [20, 24]. Therefore, we propose, for the first time (to our best knowledge), 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 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 novel, hash-based synopsis data structure, termed "*Two Dimensional Hash Sketch (2DHS)*", by extending the Flajolet-Martin sketch [13] on two join tables. We present novel estimation algorithms to provide high-confidence estimates to compare the filtering power of two filters. Our method carries both theoretical and practical significances. 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 data sets and its running time

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 a complexity of 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+nlogn+kn).*

Figure 2: Comparison of String Similarity Algorithms with synonyms (Assume both strings have a length of L; The maximum number of applicable synonyms is n; k is the maximum size of the right hand side of a rule.)

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 data sets to demonstrate the superior efficiency and effectiveness of our algorithms (Section 5). The results show that our algorithms are up to two orders of magnitude as efficient as the existing methods while offering comparable or better effectiveness.

2. RELATED WORK

String similarity measures: There is a rich set of string similarity measures including character n-gram similarity [17], Levenshtein Distance [15], Jaro-Winkler measure [31], Jaccard similarity [9], tf-idf based cosine similarity [25], and Hidden Markov Model-based measure [22]. A comparison of many string similarity functions is performed in [11]. 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 [7, 27] have been proposed to improve the quality of traditional syntactic similarity-based approaches. In particular, learnable string similarity [7] presents trainable measures to improve the effectiveness of duplicate detection, and [27] uses logistic regression to learn a string similarity measure from an existing gene/protein name dictionary.

String similarity join: Majority of the state-of-the-art approaches [33, 24, 21, 29] for performing string similarity joins 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 remaining string pairs by computing their real similarity. *Prefix filtering* [9] is a popular filtering scheme, which 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. Other efficient approaches are based on enumeration [4, 30, 21] or tries [28, 32].

LSH-based schemes are widely used to process the queries approximately, meaning that certain correct answers may be missed. For Jaccard similarity, one can generate l super-signatures, each of which is a concatenation of k minhashes [8] of the set s under a minwise independent permutation; only string pairs sharing one common super-signature are deemed as candidates. k and l are two parameters that can be tuned to achieve trade-off between efficiency and accuracy [33].

Estimation of set size: Our work is also related to the estimation of distinct pairs in set-union. There are many solutions proposed for the estimation of the cardinality of distinct elements. For example, in their influential paper, Flajolet and Martin [13] proposed a randomized estimator for distinct-element counting that relies on a hash-based synopsis; to this date, the Flajolet-Martin (FM) technique remains one of the most effective approaches for this estimation problem. FM sketch and its variation (e.g., [1, 14]) focus on one dimensional data, and we extend it to two-dimensional cases in this work.

Finally, the estimation of the result size of a string similarity join has been studied recently, e.g., in [18, 19], by using random sampling and 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.

3. STRING SIMILARITY MEASURES

As described in Section 1, an expansion-based framework can efficiently measure the similarity of strings with synonyms to avoid the exponential computational complexity. In this section, we describe two expansion-based measures in detail. In particular, Section 3.1 describes an efficient full-expansion measure, and Section 3.2 shows a selective expansion to achieve better effectiveness. Finally, Section 3.3 analyzes the optimality of the methods.

3.1 Full Expansion

We first introduce 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: $lhs(r) \to rhs(r)$, where $lhs(r)$ and $rhs(r)$ are the left and right hand sides of r, respectively. Slightly abusing the notations, $lhs(r)$ and $rhs(r)$ can also refer to the *set* generated from the respective strings. Let $\mathbb{R}$ denote a collection of synonym rules, i.e., $\mathbb{R} = \{r\colon lhs(r) \to rhs(r)\}$. 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. 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 $S' = S \bigcup\limits_{r \in \mathcal{R}(s)} rhs(r)$.

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 doing substring matching to identify the applicable synonym pairs is $O(L)$ using a suffix-trie-based method [12]. 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 scheme to capture semantics of strings using synonyms.

3.2 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 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.

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 [16]. Due to space limitation, we leave the detailed proof in the full version. Since *Selective-Jaccard* is NP-hard, we design an efficient greedy algorithm that guarantees the 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: 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
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;
  //Let $\mathbb{R}^{'}$ 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 $\frac{|G|}{|U|}$;
```

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 $RG(r) = \frac{|G|}{|U|}$ is returned (line 6).

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 $\mathcal{C}_i$ (in lines 5$\sim$11). In lines 6$\sim$7, we use the elements in $\mathcal{C}_i$ to expand S_i to acquire S_i'. Then, in lines 9$\sim$11, any rule whose rule gain is less than $\frac{\theta}{1+\theta}$ is removed from $\mathcal{C}_i$, where θ is defined as 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. Note that after a new rule is used to expand the set, the rule gains of the remaining rules need to be recomputed in line 16, since the new elements U (in Algorithm 1) of the remaining rules will change.

Performance Analysis: The time complexity of *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 $\sim$ 7) is $O(kn)$, and the iteration time is no more than n, thus the complexity is $O(L + kn^2)$.

Algorithm 2: 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($Jaccard(s_1, s_2)$, $\theta$);

  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     $\theta \leftarrow Jaccard(S_1^{'}, S_2^{'})$
9     for ($r \in \mathcal{C}_1 \cup \mathcal{C}_2$ ) do
10        if ($RG(r) < \frac{\theta}{1+\theta}$) then
11            $\mathcal{C}_i \leftarrow \mathcal{C}_i - r$; //$r \in \mathcal{C}_i$, $i$=1 or 2
  until (no rule can be removed in line 19);
12 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 $S_2$
13 $S_1^{'} \leftarrow S_1$;
14 $S_2^{'} \leftarrow S_2$;
15 while ($\mathcal{C}_1 \cup \mathcal{C}_2 \neq \phi$) do
16     Find the current most gain-effective rule $r \in \mathcal{C}_i$ (i = 1 or 2);
17     if ($RG(r) > 0$) then
18         $S_i^{'} \leftarrow S_i^{'} \cup rhs(r)$;
19     $\mathcal{C}_i \leftarrow \mathcal{C}_i$- $r$;
20 return Jaccard($S_1^{'}$, $S_2^{'}$);
```

3.3 Theoretical Analysis

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

THEOREM 1. *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.*

PROOF. 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, $\frac{G^*}{U^*} > \frac{\theta}{1+\theta}$, i.e., $RG(r^*) > \frac{\theta}{1+\theta}$, 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 (line 17), so all the rules will be used by SE in the expanding phase, which concludes the proof. □

Note that the condition in Theorem 1 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 JOINS

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.

One join algorithm is a signature-based nested-loop approach, called SN-Join, following the filtering-and-verification framework. That is, in the filtering step, SN-Join generates candidate string pairs by identifying all pairs (s, t) such that the signatures of s and t overlap. In the verification step, SN-Join checks if $sim(s, t, \mathbb{R}) \geq \theta$ for each candidate and outputs those that satisfy the similarity predicate.

To generate the signatures, one way is to extend the prefix-filter scheme. 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.

We observe that SN-join needs to check the signature overlap for each string pair $(s, t) \in S \times T$ to find the candidates, which can be time-consuming. In this section, we propose an optimized native indexes and the corresponding join algorithm (Section 4.1). We then study the impact of signature filters on the performance of algorithms in depth by proposing a novel hash-based synopsis data structure to effectively select signatures (Section 4.2). Finally we extend our discussion on LSH scheme (Section 4.3).

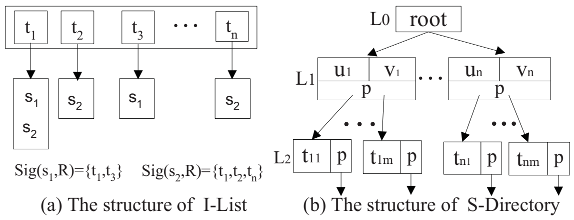


Figure 3: The structure of I-List and S-Directory.

4.1 SI-Join Algorithm

To optimize the above SN-Join method, we show a native index by (i) avoiding signature-overlap checking; and (ii) reducing the number of candidates for the final verification, as follows.

Avoiding signature-overlapping check. We design a signature inverted list, called **I-List** (see Figure 3(a)). 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 string IDs without any signature-overlapping check.

Reducing the number of candidates. We extend the idea of *length filter* [6, 26] 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 $sim(s_1, s_2, R)$ 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 3(b)) is a tree structure with three levels, and nodes in different levels could 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*, which 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 full expanded sets of strings whose length is u.

• *leaf-entry*. A node in level L_2 is called *leaf entry*, which contains two fields $\langle t, P \rangle$, where t is an integer to denote the number of the tokens in the full 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 3(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, to perform a self-join for a table with one million strings on USPS dataset, the size of S-Directory 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**, i.e., each leaf entry in S-Directory is associated with an I-List. Based on the SI-tree, we design a join algorithm (called **SI-Join**), which follows the filtering-and-verification framework and the verification phase is the same as that of SN-Join.

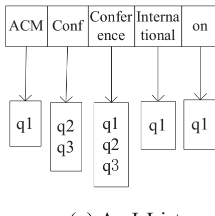
SI-Join (see Algorithm 3) has three steps in the filtering phase. In the first step (lines 2 ∼ 3), consider two fence-entries F_s and

ID	Strings	Signatures	L	F
q_1	2013 ACM Intl Conf on Management of Data USA	ACM, International, Conference, on	9	11
q_2	Very Large Data Bases Conf	Conf, Conference	5	7
q_3	VLDB Conf	Conf, Conference	2	7
q_4	ICDE 2013	ICDE	2	2

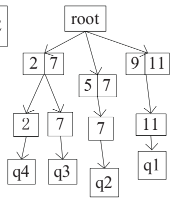
(a) Strings and signatures

ID	Synonym pairs	ID	Synonym pairs
r_1	Intl $\longrightarrow$ International	r_3	VLDB $\longrightarrow$ Very Large Data Bases
r_2	Conf $\longrightarrow$ Conference	r_4	Very Large Data Bases $\longrightarrow$ VLDB

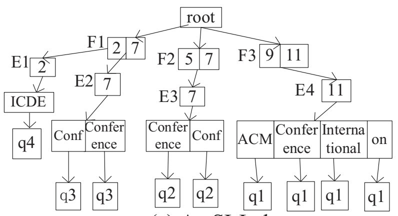
(b) Synonym pairs



(c) An I-List



(d) An S-Direcotry



(e) An SI-Index

Figure 4: 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.

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.

Algorithm 3: SI-Join Algorithm

```
Input  : SI-Trees $SI_S$ for $S$ and $SI_T$ for $T$, threshold value $\theta$, a collection
         of synonym pairs $\mathbb{R}$.
Output : $\mathbb{C}$: Candidate string pairs $(s, t) \in SI_S \times SI_T$
1  $\mathbb{C} \longleftarrow \emptyset$;
2  for ($\forall$ $F_s$ in $SI_S$, $\forall$ $F_t$ in $SI_T$) do
       //Fence-entry:F=<u, v, P>
3      if ($min(F_s.v, F_t.v) \geq \theta \times max(F_s.u, F_t.u)$) then
4          for ($\forall$ $E_s \in F_s.P$, $\forall$ $E_t \in F_t.P$) do
               //Leaf-entry:E=<t, P>
5              if ($min(E_s.t, E_t.t) \geq \theta \times max(F_s.u, F_t.u)$) then
6                  for ($\forall$ $g$ is an overlapping signature pointed by $E_s.P$ and
                   $E_t.P$) do
7                      $L_s$=$E_s.P \rightarrow$getIList($g$);
8                      $L_t$=$E_t.P \rightarrow$getIList($g$);
9                      $C$=$\{(s, t) | s \in L_s, t \in L_t\}$;
10                     $\mathbb{C}$=$\mathbb{C} \cup C$;
11 return $\mathbb{C}$;
```

EXAMPLE 3. *We use this example to illustrate the SI-Join algorithm. Consider a self-join on the table in Fig. 4(a) with $\theta = 0.8$ and synonyms in Fig. 4(b). The corresponding I-Lists, S-Directory, and SI-index are shown in Figs. 4(c), (d) and (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∼3). Similarly, all pairs below (E_1, E_3) can be pruned (lines 4∼5). Then SI-Join uses the signatures "Conference" and "Conf" to get I-Lists below (E_2, E_3) (lines 7∼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∼10). The final results are (q_2, q_3) and $\{(q_i, q_i) | i \in [1, 4]\}$.*

4.2 Estimation-based signature selection

4.2.1 Motivation

The SI-join algorithm generates 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. Given two string tables for approximate join, it is impractical to design an efficient algorithm to select signatures for each string online. This is because even one scan of data is time consuming and it is meaningless to spend comparative time as the real join to find good signatures. In this section, we propose a novel strategy to solve this problem. Our main idea is to use multiple ways to generate signatures offline, and then given two tables to join, we quickly estimate the quality of each signature filter and select the best one to perform the actual filtering. The main challenge in this idea is two-fold. First, we need to select multiple promising signature filters offline. Second, the estimator should quickly determine the best filter online according to their filtering power.

To address the first challenge, we use the *itf*-based method ([2, 20]) to generate filters. In particular, let *tf(w)* denote the occurrences of token w appearing in the join tables, i.e., 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 not to be 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*, and 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 using only signature filters with *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 1000 abbreviations of conference names, while table T has 1000 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 5.

Figure 5 reports the experimental results. We observe that (i) four signature filters achieve quite different filtering powers. 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. 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 filer 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 is 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. Therefore, in the following, given two tables for join,

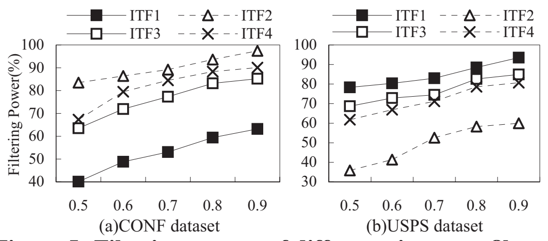


Figure 5: Filtering powers of different signature filters.

we propose an estimator to effectively estimate the filtering power of different filters to dynamically select the best filter.

4.2.2 Estimation-based selection algorithms

Let $\ell(S,g)=\{q_1^g,...,q_n^g\}$ denote a list of string IDs (i.e., I-lists in Figs. 3 and 4) 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, given a signature filter λ, the union of all ID pairs from all signatures in G constitutes the set of candidates after signature filtering, that is, $\underset{g\in G}{\cup}\ell(S,g)\times\ell(T,g)$.

We can use the following lemma to estimate the lower bound and the upper bound of the cardinality of candidates.

LEMMA 1. *Given 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.

In the above lemma, the upper bound is defined as the total number of string pairs and the lower bound is the maximal number of string pairs associated with one signature. A signature λ guarantees to return fewer candidates than signature λ' if $UB(\lambda)<LB(\lambda')$. Obviously, the tighter the bound is, the better the results are, since a tighter bound implies more accurate estimates on the ability of a filter. Therefore, in the following, we introduce tighter upper and lower bounds by extending the Flajolet-Martin (FM) [13] technique.

The Flajolet-Martin estimator: The Flajolet-Martin (FM) technique [13] 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 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 $i\in[0,2^{L-1}]$, $p(h(i))\in\{0,\dots,\log M\text{-}1\}$ 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 6 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 to extend FM sketch to provide better bounds to estimate the number of candidate pairs. The main challenge is that the existing FM-based methods [13, 14] are only applicable to one dimensional data, but we need to estimate the number of distinct

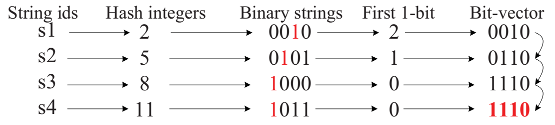
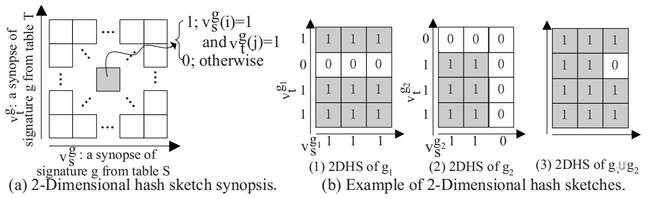


Figure 6: An example of bit-vector generated by FM method. (The first 1-bit is the left-most 1-bit.)


ν_T^g: a synopse of signature g from table T
ν_S^g: a synopse of signature g from table S
1; $\nu_S^g(i)$=1 and $\nu_T^g(j)$=1
0; otherwise
(a) 2-Dimensional hash sketch synopsis.
(1) 2DHS of g_1
(2) 2DHS of g_2
(3) 2DHS of $g_1\cup g_2$
(b) Example of 2-Dimensional hash sketches.
Figure 7: The structure of 2-dimensional hash sketch (2DHS)

pairs based on two independent FM sketches. In addition, the extra complication occurs, since we require the bounds to be correct with arbitrarily high probability. We now describe four steps to address the challenges.

Step 1 (Estimating the distinct number of one table): At first, we estimate the distinct number of string IDs for all signature tokens from one table, that is, μ_S and μ_T, where $\mu_S=\underset{g\in G}{\cup}\ell(S,g)$ and $\mu_T=\underset{g\in G}{\cup}\ell(T,g)$. Ganguly et al. [14] 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. Space precludes detailed discussion about their method in this paper. However, readers may find it in Section 3.3 of [14]. 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 an FM sketch of signature g in I-Lists of table S and table T respectively. We construct a 2-dimensional hash sketch synopsis ($2DHS$) of signature g, which is a 2-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 $\underset{g\in G}{\cup}2DHS^g$, such that $2DHS'(i,j)$=1 if $\exists$ $2DHS^g(i,j)$=1.

EXAMPLE 4. *Figure 7 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 7(b1) is the 2DHS of g_1 and Figure 7(b2) is the 2DHS of g_2. Then the union $2DHS$ is shown in Figure 7(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* (ν_1^g,ν_2^g,...,ν_ω^g) (each copy using independently-chosen hash functions from $\mathcal{H}$). Algorithm 4 shows the steps to compute 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

Algorithm 4: Computing two witness probabilities

```
   Input  : $\omega$ independent 2-dimensional hash sketches
            ($2DHS_1$,$2DHS_2$,...,$2DHS_\omega$), $R_s$ and $R_t$.
   Output : $\hat{p_s}$ and $\hat{p_t}$ and two witness positions ($R_s$, $y$) and ($x$, $R_t$)
1  $f$=0.3$\omega$;
2  for ($y$ from 0 to $R_t$) do
3      $count$=0;
4      for ($i$ from 1 to $\omega$) do
5          if ($2DHS_i[R_s][y]$==1) then
6              $count = count$+1;
7      if ($count \leq f$) then
8          $\hat{p_s}$=$count/\omega$ and goto Line 9;
9  for ($x$ from 0 to $R_s$) do
10     $count$=0;
11     for ($i$ from 1 to $\omega$) do
12         if ($2DHS_i[x][R_t]$==1) then
13             $count = count$+1;
14     if ($count \leq f$) then
15         $\hat{p_t}$=$count/\omega$ and goto Line 16;
16 return $\hat{p_s}$, $\hat{p_t}$ and their witness positions ($R_s$, $y$), ($x$, $R_t$)
```

of a signature filter λ. Assume, without loss of generality, that $\mu_S \leq \mu_T$.

TUB(λ) $= \varphi_1 \cdot min(u_1, u_2)$, where

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

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

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

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

where constants φ_1=1.43, φ_2=0.8, are derived in Theorem 2.

EXAMPLE 5. *This example illustrates the computation of TUB and TLB. Assume μ_S=100 and μ_T=200. Thus R_s=$\lfloor \log_2^{100} \rfloor$=6, R_t=$\lfloor \log_2^{200} \rfloor$=7. Assume, in Algorithm 4, 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, Δp=0.0117, p'_s=0.2375, and the upper bound $TUB(\lambda)$ = 1.43*1729 = 2472 and $TLB(\lambda)$ = 1722×0.8=1278.*

Therefore, one filter is selected as the best one 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, and then we analyze its space and time complexities. Due to space constraint, some proofs are omitted here.

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 $[1-(1-\frac{1}{2^{x+y}})^n]$. 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 paper 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 can ensure Lemmas 3 and 4 to compute the lower and upper bounds.

LEMMA 2. *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 3. *The lower bound of the MIN probability can be computed as:*

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

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

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

PROOF. *(Sketch) According to Lemma 2, 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 4. *Assume that $\mu_S \leq \mu_T$. Then the upper bound of the MAX probability can be computed as:*

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

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

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

PROOF. *(Sketch) According to Lemma 2, with a fixed μ_S, the greater the value μ_T is, the greater the MAX probability is. In the worst case, μ_T = N. Therefore, the MAX probability p_{max} is no greater than:*

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

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.

$p_{max} - p_1 \leq 1 - [(1-\frac{1}{2^x}) + \frac{1}{2^x} \cdot (1-\frac{1}{2^y})]^{\mu_S}$ - $[1-(1-\frac{1}{2^x})^{\mu_S}] \times [1-(1-\frac{1}{2^y})] = p_2$. □

THEOREM 2. *Given a signature filter λ, with any high probability 1-δ, our algorithm guarantees to correctly return the upper bound and the lower bound 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 4 are low-error and high-confidence, and then we show 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 4, by Chernoff bound, the estimated $\hat{p}$=count/ω satisfies $|\hat{p} - p| \leq \epsilon p$ with a probability at least 1-δ as long as $\omega p \geq \frac{2\log(1/\delta)}{\epsilon^2}$. Let ϵ=0.15, Then $\omega p \geq 88\log(1/\delta)$. Since $\hat{p_s}$ and $\hat{p_t}$ returned in Algorithm 4 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, $p \geq 0.15$, and we can generate enough number of sketches such that $\omega \geq 587\log(1/\delta) = \Theta(\log\frac{1}{\delta})$. 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 3 and 4, 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 5 is used to derive φ_1 in TUB and Lemma 6 for φ_2 in TLB, respectively. (A similar proof technique can be found in [5, 14], even though our estimation is quite different from theirs.)

LEMMA 5. *Let f(x) = $ln(1 - \frac{x}{1-(1-\frac{1}{2^{R_s}})^{\mu_S(1+\varepsilon)}})$. If $y - x \leq 0.15x$ for some $x \leq 0.3$ and $\varepsilon \leq 0.1$, then $f(y) - f(x) \leq 0.43f(x)$.*

PROOF. *Since $1-(1-\frac{1}{2^{R_s}})^{\mu_S(1+\varepsilon)} \approx 1-e^{-(1+\varepsilon)}$ and $\varepsilon \leq 0.1$, then $\frac{1}{1-e^{-(1+\varepsilon)}} \geq 1.5$. 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 $f(y) - f(x) \leq \frac{1.5(y-x)}{1-1.5max(x,y)} \leq \frac{0.225x}{1-1.725x} \leq \frac{0.225x}{0.52} \approx 0.43x \leq -0.43ln(1 - 1.5x) \leq 0.43f(x)$.* □

LEMMA 6. *Let f(x) = $ln(1 - \frac{x-\Delta p}{1-(1-\frac{1}{2^{R_s}})^{\mu_S(1+\varepsilon)}})$. If $x - y \leq 0.15x$ for some $x \leq 0.3$ and $\varepsilon \leq 0.1$, then $f(x) - f(y) \leq 0.2f(x)$.*

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 $|f(y) - f(x)| \leq \frac{1.5(x-y)}{1.71-1.5\max(x,y)} \leq \frac{0.225x}{1.71-1.725x} \leq \frac{0.225x}{1.19} \approx 0.189x \leq 0.2\ln(1.71 - 1.5x)$, since by Maclaurin series, $\ln(1.71 - 1.5x) \approx \ln 1.71 - (1.5/1.71)x \approx 0.54 - 0.88x$.* □

THEOREM 3. *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 $\Theta(T \log^2 M + \log M \log(1/\delta))$.*

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 ω=$\Theta(\log \frac{1}{\delta})$ 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 $2\omega \log M$. The cost of Step 2 is $2Tlog^2M$ and that of Step 3 and 4 are $\omega \log M$ and O(1) respectively. Therefore the total cost is bounded by $\Theta(T \log^2 M + \log M \log(1/\delta))$.* □

4.3 Extension to LSH filters

Although we focus on prefix filters in the above discussion, our indexes and estimators can also be extended to other filters, such as 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 the signature s comes from the i-th minhash, $1 \leq i \leq l$. Therefore, by replacing the prefix signature g with (s, i), our SN-join (Algorithm 3) and 2DHS estimator can be directly applied on LSH scheme. As described in the next section, we implemented both LSH and prefix filters in our algorithms to make a comprehensive comparison.

Dataset	# of Strings	String Len (avg/max)	# of Synonym Pairs	# of Applicable Synonym Pairs Per String (avg/max)
USPS	1,000,000	6.75 / 15	300	2.19 / 5
CONF	10,000	5.84 / 14	1000	1.43 /4
SPROT	1,000,000	10.32 / 20	10,000	17.96 /68

Figure 8: Characteristics of Datasets.

5. EXPERIMENTAL RESULTS

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 evaluate our SN-join, SI-Join and the state-of-the-art approaches to perform similarity join with prefixes and LSH signatures. Finally, we analyze the performance of 2DHS estimators.

All the algorithms were 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.

5.1 Datasets and Synonyms

Data Sets. We use three datasets: US 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 downloaded common person names, street names, city names, states, and zip codes from the United States Postal Service website (http://www.usps.com). We then generated 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 US addresses, from which we obtained 284 synonym pairs. The synonym pairs covers a wide range of alternate representations of common strings, e.g., *street* → *st*.
CONF: We collected 10,000 conference names from more than ten domains, including Medicine and Computer Science. We obtained 1000 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.

Figure 8 gives the characteristics of the three datasets.

5.2 Experimental results

5.2.1 Quality and Efficiency of Similarity Measures

The first experiment is to demonstrate 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 [2].

For each of the three datasets, we performed the experiments by conducting a similarity join between a query table T_Q and a 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.

θ	CONF dataset												USPS dataset												SPROT dataset											
	Jacc			**JaccT**			**Full**			**SE**			**Jacc**			**JaccT**			**Full**			**SE**			**Jacc**			**JaccT**			**Full**			**SE**		
	P	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**	**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	0.01	0.74	0.02	0.08	0.95	0.15	0.06	0.95	0.11	0.05	1.00	0.10	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	0.14	0.44	0.21	0.47	0.83	0.60	0.50	0.85	0.63	0.69	0.90	0.78	0.01	0.42	0.02	0.08	0.82	0.15	0.12	0.93	0.21	0.10	0.95	0.19	0.01	0.32	0.02	0.07	0.72	0.13	0.08	0.68	0.15	0.10	0.90	0.18
0.70	0.44	0.40	0.42	0.52	0.80	0.63	0.51	0.82	0.63	0.72	0.85	0.78	0.06	0.42	0.11	0.22	0.82	0.35	0.26	0.93	0.41	0.36	0.93	0.52	0.05	0.32	0.08	0.19	0.71	0.30	0.18	0.65	0.28	0.33	0.87	0.48
0.80	0.63	0.35	0.45	0.76	0.74	0.75	0.55	0.70	0.62	0.87	0.84	**0.85**	0.21	0.42	0.28	0.39	0.73	0.51	0.61	0.80	0.70	0.79	0.91	0.85	0.16	0.32	0.21	0.33	0.62	0.43	0.52	0.64	0.58	0.69	0.83	0.75
0.90	0.82	0.21	0.33	0.80	0.71	0.75	0.68	0.60	0.64	0.89	0.80	0.84	0.71	0.42	0.53	0.61	0.73	0.66	0.84	0.75	0.79	0.96	0.89	**0.92**	0.54	0.32	0.40	0.48	0.57	0.52	0.69	0.61	0.65	0.83	0.82	**0.82**

Figure 9: Quality of similarity measures (P: precision, R: recall, F: F-measure).

CONF dataset			USPS dataset			SPROT dataset		
Jacc	0.14	s1= "Proceedings of the VLDB Endowment 2012: 38th International Conference on Very Large Databases, Turkey" s2= "PVLDB 2012 Turkey"	Jacc	0.00	s1= "University of Washington 1705 NE Pacific St Seattle, WA 98195" s2= "UW"	Jacc	0.00	s1= "P93214" s2= "14339_SOLL, 14-3-3 protein 9"
JaccT	0.57		JaccT	0.70		JaccT	0.75	
Full	0.93	r1:PVLDB−>International Conference on Very Large Databases r2:PVLDB−>Proceedings of the VLDB Endowment	Full	0.92	r1:UW->University of Washington r2:UW->1705 NE Pacific St Seattle, WA 98195 r3:UW->University of Waterloo	Full	0.83	r1:P93214−>14339_SOLL r2:P93214−>14-3-3 protein 9 r3:14339_SOLL−>UPI0000124DEC
SE	0. 93		SE	1.00		SE	1.00	

Figure 10: Three examples to illustrate the quality of similarity measures.

Quality of Measures. In Figure 9, we report the quality of the measures by testing the *Precision* (short for "P"), *Recall* ("R"), and *F-measure*= $\frac{2 \times P \times R}{P+R}$ ("F") on three datasets. We observe that:

• The similarity measures using synonyms (including *JaccT*, *Full* and *SE*) obtain higher scores than Jaccard which does not consider synonym pairs. The reason is that without using synonyms, Jaccard has no chance to improve 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 is 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 situation is not rare in the real world data, as one fragment of a string likely involves multiple synonym rules. We illustrate one example on each of three datasets in Figure 10 to compare the performance of four 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%

Figure 11: Empirical Probabilities of Optimal Cases for SE

Optimality Scenarios of SE measure As described in Theorem 1, 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 performed experiments on three datasets using different data sizes. The results are shown in Figure 11. The average percentages of optimal cases are 71.39%, 68.11%, and 87.32% in USPS, CONF and SPROT data, respectively. Therefore, the condition in Theorem 1 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 assess their efficiency. Figure 12(a) shows the time cost of the four measures on SPROT dataset (10K records) by running 10^8 times of 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 find 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 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, which 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 9).

5.2.2 *Efficiency and Scalability of Join Algorithms*

The second experiment is to test the efficiency and scalability of various join algorithms. We compared our algorithms, **SN-Join**, and **SI-Join** with the algorithm in [2] (also termed **JaccT**). Our implementation of JaccT includes all the optimizations proposed in [2]. For our algorithms, since they can work with the two similarity measures (i.e., full expansion, selective expansion), we denote them as **SN(F)**, **SN(S)**, and **SI(F)** and **SI(S)** respectively. We implemented all algorithms using both prefix and LSH filters. Therefore, with respect to the algorithms using LSH scheme, we append "-L" to the name, e.g., JaccT-L denotes the JaccT algorithm using LSH scheme. Note that, we use the false negative ratio $\delta \leq 5\%$, i.e, the accuracy is $1 - \delta \geq 95\%$. The parameters k and l should satisfy: $\delta \geq (1 - \theta^k)^l$ [33]. For example, if threshold $\theta = 0.8$, $k = 3$, then l should be at least 5 to guarantee at least 95% accuracy.

Metrics. We took 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).

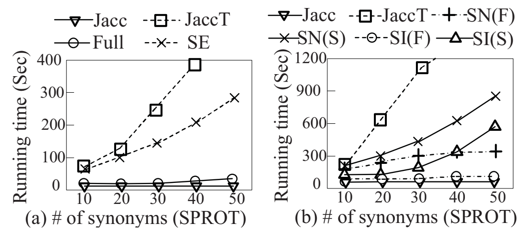


Figure 12: Performance with varying synonyms.

Number of Signatures. We first performed experiments to report the number of signatures of a query string. It has a major impact on the query time, as both JaccT and our join algorithms need to

	Prefix filtering scheme				Locality sensitive hashing (LSH) scheme					
	USPS		SPROT		USPS (k=4)			SPROT(k=6)		
θ	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

Figure 13: Number of signatures (Prefix filtering scheme and LSH scheme).

	USPS (100K self-join)						SPROT(1000K self-join)					
	Filtering Ratio(%)			Candidate Size(x10^6)			Filtering Ratio(%)			Candidate Size(x10^9)		
θ	JaccT	SN-Join	SI-Join	JaccT	SN-Join	SI-Join	JaccT	SN-Join	SI-Join	JaccT	SN-Join	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

Figure 14: Filtering ratio with varying thresholds.

frequently check the overlaps of string signatures. The results are reported in Figure 13. For both prefix and LSH schemes, the number of signatures of our expansion-based algorithms is smaller than that of JaccT. The reason 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 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 then investigated the filtering power of different algorithms in Figures 14–15 using prefix and LSH schemes, respectively. The experiments were performed on USPS and SPROT datasets for self-join. For prefix-based algorithms, SN-Join is slightly better than JaccT, while SI-Join has a substantial lead over JaccT and SN-Join. These results are mainly because of the additional filtering power in SI-Join bought by length filtering and signature filtering. Next, for LSH-based algorithms, Figure 15 demonstrates the similar trend. That is, SI-Join is the winner in all settings by varying the 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 correct answers, resulting in false negatives, as can be seen from the false negative percentages shown in Figure 15.

Running Time and Scalability. Figure 16 and Figure 17 show the running time of five join algorithms based on prefix and LSH schemes respectively (the join threshold is 0.8). The x-axis represents the join data size. As shown, the running times of both JaccT and SN-Join (i.e., SN(F), SN(S)) have an exponential growth, whereas SI-Join (i.e., SI(F), SI(S)) scales better (i.e., linear) than SN-Join and JaccT. The reason is that SI-Join methods have more efficient filtering strategies and generate smaller size of candidates. In addition, in order to study the scalability of algorithms with the increase of the number of rules, we plot Figure 12(b), where SI(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.

Prefix vs. LSH We then sought to analyze system performances with different signature schemes, namely, prefix vs. LSH. Figure 18 plots the running times of SI(S)-L (LSH) and SI(S) (Prefix) with varying join thresholds. Results on SI(F) and SN-join are similar. Note that, for SI(S)-L, the choice of k and l has substantial impact on the query time, hence we tuned and used the

k/L	USPS						SPROT					
	JaccT		SN-Join		SI-Join		JaccT		SN-Join		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

Figure 15: Filter ratio (FR) and false negative (FN) with varying parameters k and l. (accuracy=95%, threshold=0.9).

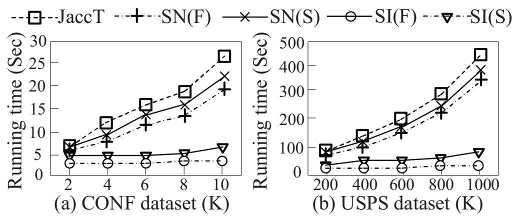


Figure 16: Running time in CONF and USPS datasets.

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 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 signatures is combined (i.e, $k \geq 2$) in these settings, which makes LSH method quite selective. However, when the threshold is less than 0.8, we tried all reasonable combinations of k and l values and still cannot find a setting to beat prefix. This is 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 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 case. This 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 18.

5.2.3 Estimation-based signature selection

The last set of experiments (Figure 19) studies the quality of *2DHS* synopsis on prefix and LSH schemes. For prefix filter, we use the four *frequency*-based signature filters, i.e., ITF1∼ITF4, which are described in Section 4.2. For LSH scheme, we vary the parameters k and l to generate six filters. The off-line processing time for the generation of all signatures of one filter is around 200s ∼ 300s, which is a one-time cost.

We first computed the upper bound (UB) and the lower bound (LB) of each filter by Lemma 1. One filter is returned as the best one 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 19(a), ITF2 is the best one in 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 19(b), ITF1 is the best filters in 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 5 for prefix scheme. In addition, the estimate cost accounts for only 1% of the join time. For example, in USPS dataset, the time cost of

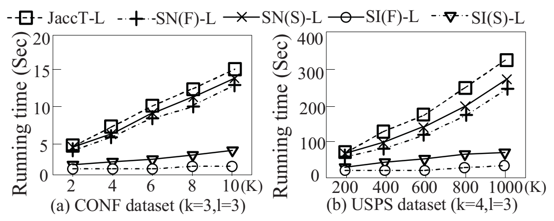
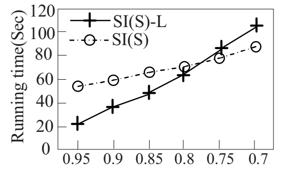


Figure 17: Performance on LSH signature scheme.

θ	k	l
0.95	4	2
0.90	4	3
0.85	2	3
0.80	2	3
0.75	1	3
0.70	1	3

(a) k,l settings



(b) USPS dataset

Figure 18: Prefix scheme vs. LSH scheme (with varying threshold values).

our estimator is only $0.81s$ while the filtering time is $83.7s$ and the total running time is $113.4s$.

Summary Finally, we summarize our 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 efficiently, but its F-measure is not as good as Selective-expansion, which makes a good balance between effectiveness and efficiency.

(2) With respect to three similarity join algorithms, SI-join is the winner in all settings over JaccT and SN-join, by using SI-tree to efficiently perform length filtering and signature filtering. We achieve a speedup of up to 50∼300x on three data sets over the state-of-the-art approach.

(3) Finally, 2DHS synopsis offers very good results, and is extremely efficient to estimate the filtering power of different filters, (<1 second , accounting for only 1% of the total running time), which strongly motivates its application in practice.

6. CONCLUSIONS

In this paper, we studied the problem of strings similarity measures and joins with semantic rules (i.e. synonyms). We proposed two expansion-based methods to measure the similarity of strings and a novel index structure called SI-tree to perform the similarity join. We also proposed an estimator to select signatures online to optimize the efficiency of signature filters in join algorithms. The 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 approach over the state-of-the-art methods in three data sets.

7. ACKNOWLEDGEMENTS

This research is partially supported by 973 Program of China (Project No. 2012CB316205), NSF China (No. 60903056), 863 National High-tech Research Plan of China (No. 2012AA011001), Beijing Natural Science Foundation (No. 4112030), and New Century Excellent Talents in University. Wei Wang is supported by ARC Discovery Projects DP130103401 and DP130103405.

8. REFERENCES

[1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *STOC*, pages 20–29, 1996.

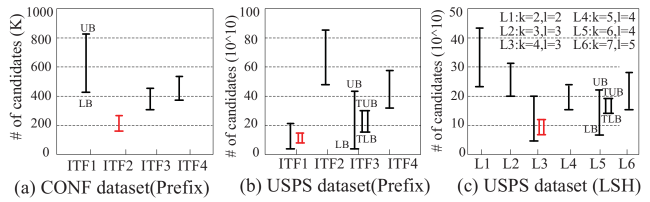


Figure 19: Estimation results of 2DHS synopsis for prefix and LSH schemes (threshold=0.90).

[2] A. Arasu, S. Chaudhuri, and R. Kaushik. Transformation-based framework for record matching. In *ICDE*, pages 40–49, 2008.

[3] A. Arasu, V. Ganti, and R. Kaushik. Efficient exact set-similarity joins. In *VLDB*, pages 918–929, 2006.

[4] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting distinct elements in a data stream. In *RANDOM*, pages 1–10, 2002.

[5] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *WWW*, pages 131–140, 2007.

[6] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *KDD*, pages 39–48, 2003.

[7] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. *Computer Networks*, 29(8-13):1157–1166, 1997.

[8] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *ICDE*, page 5, 2006.

[9] S. Chaudhuri and R. Kaushik. Extending autocompletion to tolerate errors. In *SIGMOD Conference*, pages 707–718, 2009.

[10] W. W. Cohen, P. D. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *IIWeb*, pages 73–78, 2003.

[11] M. Farach. Optimal suffix tree construction with large alphabets. In *FOCS*, pages 137–143, 1997.

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

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

[14] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *VLDB*, pages 491–500, 2001.

[15] K. Iwama and S. Tamaki. Improved upper bounds for 3-sat. In *SODA*, 2004.

[16] G. Kondrak. *N*-gram similarity and distance. In *SPIRE*, pages 115–126, 2005.

[17] H. Lee, R. T. Ng, and K. Shim. Power-law based estimation of set similarity join size. *PVLDB*, 2(1):658–669, 2009.

[18] H. Lee, R. T. Ng, and K. Shim. Similarity join size estimation using locality sensitive hashing. *PVLDB*, 4(6):338–349, 2011.

[19] C. Li, J. Lu, and Y. Lu. Efficient merging and filtering algorithms for approximate string searches. In *ICDE*, pages 257–266, 2008.

[20] G. Li, D. Deng, J. Wang, and J. Feng. Pass-join: A partition-based method for similarity joins. In *VLDB*, 2012.

[21] D. R. H. Miller, T. Leek, and R. M. Schwartz. A hidden markov model information retrieval system. In *SIGIR*, pages 214–221, 1999.

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

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

[24] S. Sarawagi and A. Kirpal. Efficient set joins on similarity predicates. In *SIGMOD Conference*, pages 743–754, 2004.

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

[26] J. Wang, G. Li, and J. Feng. Trie-join: Efficient trie-based string similarity joins with edit-distance constraints. *PVLDB*, 3(1):1219–1230, 2010.

[27] J. Wang, G. Li, and J. Feng. Can we beat the prefix filtering?: an adaptive framework for similarity join and search. In *SIGMOD Conference*, pages 85–96, 2012.

[28] W. Wang, C. Xiao, X. Lin, and C. Zhang. Efficient approximate entity extraction with edit distance constraints. In *SIGMOD Conference*, pages 759–770, 2009.

[29] W. E. Winkler. The state of record linkage and current research problems. Technical report, Statistical Research Division, U.S. Census Bureau, 1999.

[30] C. Xiao, J. Qin, W. Wang, Y. Ishikawa, K. Tsuda, and K. Sadakane. Efficient error-tolerant query autocompletion. *PVLDB*, 2013.

[31] C. Xiao, W. Wang, X. Lin, J. X. Yu, and G. Wang. Efficient similarity joins for near-duplicate detection. *ACM Trans. Database Syst.*, 36(3):15, 2011.