Approximate Substring Selectivity Estimation *

Hongrae Lee
University of British Columbia
xguy@cs.ubc.ca

Raymond T. Ng
University of British Columbia
rng@cs.ubc.ca

Kyuseok Shim
Seoul National University
shim@ee.snu.ac.kr

ABSTRACT

We study the problem of estimating selectivity of approximate substring queries. Its importance in databases is ever increasing as more and more data are input by users and are integrated with many typographical errors and different spelling conventions. To begin with, we consider edit distance for the similarity between a pair of strings. Based on information stored in an extended N-gram table, we propose two estimation algorithms, MOF and LBS for the task. The latter extends the former with ideas from set hashing signatures. The experimental results show that MOF is a light-weight algorithm that gives fairly accurate estimations. However, if more space is available, LBS can give better accuracy than MOF and other baseline methods. Next, we extend the proposed solution to other similarity predicates, SQL LIKE operator and Jaccard similarity.

1. INTRODUCTION

With the widespread use of the Internet, text-based data sources have become ubiquitous. The demand for effective support of approximate or fuzzy string queries becomes ever increasing because of the presence of unclean data and different spellings. Suppose a user wants to retrieve information on a customer named 'przymusinski' in a decision support system. Using SQL *like* operator, she may express the query as "name like pr%s_nsk%". Alternatively, she may search for the name as 'przimunsinski' but allow a match to be within an edit distance of 2. In query optimization, accurate and efficient selectivity estimation is essential to produce an optimized query execution plan. In the example query, when the selectivity of the predicate is very high, the plan using index seek will be optimal if an appropriate secondary index on the name column is available. However, if its selectivity is low, the cost of scanning the data might be cheaper.

To handle approximate substring matching, various string (dis)similarity measures, such as edit distance, hamming distance, Jaccard coefficient, cosine similarity and Jaro-Winkler distance have been considered [2, 16, 15, 6]. Edit distance is one of most widely accepted measures for general database applications where domain specific knowledge is not assumed [1, 14, 15, 18, 22]. The edit distance between two strings s_1 and s_2, denoted as $ed(s_1, s_2)$, is the minimum number of edit operations of single characters that are needed to transform s_1 to s_2.

In this paper, we first consider edit distance to measure similarity and will extend the solution to other similarity measures in later sections. We distinguish between the two following problems concerning edit distance:

Substring selectivity estimation: *G*iven a query substring s_q and a bag of strings DB, estimate the number of strings $s \in DB$ satisfying $ed(s_q, b) \leq \Delta$, where b is a substring of s and Δ is the edit distance threshold.

Full string selectivity estimation: *G*iven a query string s_q and a bag of strings DB, estimate the number of strings $s \in DB$ satisfying $ed(s_q, s) \leq \Delta$, where Δ is the edit distance threshold.

EXAMPLE 1. *Consider a* $DB = \{$*'kullback', 'bach', 'eisenbach', 'bacchus', 'baeza-yates'*$\}$. *Suppose the query* $s_q \equiv$*'bach' and* $\Delta = 1$. *For substring selectivity, all 5 elements in* DB *except 'baeza-yates' satisfy the edit distance threshold, i.e.,* $|Ans($*'bach'*$, 1)| = 4$. *For full string selectivity, the corresponding answer* $Ans_f($*'bach'*$, 1)$ *is* $\{$*'bach'*$\}$.

The class of applications for substring selectivity estimation is broader than the class of applications for string selectivity estimation (e.g., "name like %pr%sinski%"). Several techniques [15, 18] were proposed to handle string selectivity estimation with edit distance. Direct application of those techniques to the substring problem is not an option since it will almost always under-estimate the true selectivity, which may change the ordering of predicates producing a bad query execution plan.

Furthermore, it is not trivial to adapt those techniques to give estimation of substring selectivity with edit distance. For the substring selectivity estimation problem, the biggest challenge is the estimation of intersection sizes among correlated substrings. Previous studies on string selectivity estimation do not need to consider this complication. To illustrate, let us return to $|Ans_f($*'bach'*$, 1)|$. Estimation methods for string selectivity partition strings into groups (e.g., clusters [15], extended q-gram entries [18]). For $|Ans_f($*'bach'*$, 1)|$,

*This research was supported by the Ministry of Knowledge Economy, Korea, under the Information Technology Research Center support program supervised by the Institute of Information Technology Advancement (grant number IITA-2008-C1090-0801-0031).

EDBT 2009, March 24–26, 2009, Saint Petersburg, Russia.

these groups include 'back', 'bacc', etc. Note that a string cannot simultaneously be the string 'back' and the string 'bacc'. This non-overlapping condition greatly simplifies counting for string selectivity estimation. For substring matching, however, a string contain substrings like 'kull', 'back' and many more substrings simultaneously, which are not necessarily similar (e.g., 'Solomon Kullback'). In other words, the substring condition gives rise to too many possible cases to consider imposing a major challenge. To the best of our knowledge, this is the first study developing algorithms for approximate substring selectivity estimation. As a preview, we make the following contributions.

- The first algorithm we propose is called MOF, which is based on the *M*Ost Frequent minimal base substring. Substring frequencies are estimated from extended q-grams.
- Recall that a key challenge for substring selectivity estimation is to estimate the overlaps among groups of substrings. MOF sidesteps this challenge by basing its estimation on a single substring. It is expected that the estimation may be improved by basing the estimation on multiple substrings. We propose in Section 4 an estimation algorithm called LBS, which uses signatures generated by set hashing techniques. However, standard set hashing is not sufficient. We extend set hashing in two ways. The first one is the approximation of signatures for set intersections. The second one is the use of multiple minimum values to improve accuracy. Depending on the amount of available space, LBS can be tuned to use larger signatures for improved estimation quality.
- The proposed algorithms can be extended to string similarity measures other than the edit distance. We show in Section 5 how to extend the algorithms to deal with SQL LIKE operator and Jaccard similarity.
- We show in Section 6 a comprehensive set of experiments. We compare MOF and LBS with two baseline methods. One is based on random sampling and the other one is a generalization of SEPIA designed for strings [15]. We explore the trade-offs between estimation accuracy, size of the intermediate data structure, and the CPU time taken for the estimation. Our results show that for fast runtime and low intermediate size, MOF is an attractive light-weight algorithm. However, if more space is available for the signatures, so that the overlaps can be more accurately estimated, LBS is the recommended choice.

1.1 Related Work

Jin and Li [15] proposed SEPIA which estimates the selectivity of string predicates with edit distance. SEPIA builds clusters of similar strings and maintains histograms that store distribution information of strings with augmented edit distance information. Given a query, it visits each cluster and estimates how many strings in the cluster are within the threshold using histograms. One of the baseline algorithms used here for experimental comparison, called S-SEPIA, extends SEPIA to handle substrings.

The extension of q-grams with wildcards for estimating selectivity of string matching with edit distance was first proposed by Lee et al. in [18]. Their approach is based on partitioning string hierarchies and exploiting structures in replacement semi-lattices. These concepts are not applicable to substring selectivity estimation. And as discussed earlier, their method does not need to estimate the overlaps among groups, which is the main focus of the methods proposed here for substrings.

Krishnan et al. first proposed to use pruned suffix trees for substring selectivity estimation [17]. The KVI estimate, which assumes independence of substrings, was proposed. Based on the Markov assumption and the concept of maximally overlapping substrings, Jagadish et al. proposed the MO estimate [13]. Chaudhuri et al. observed that MO often under-estimates and introduced the CRT algorithm based on the Short Identifying Substring (SIS) assumption [5]. Chen et al. applied set hashing to handle boolean predicates on substrings [8]. Using a suffix tree and set hashing signatures, they estimate the selectivity of substring boolean predicates. These studies [17, 13, 5] deal with the selectivity estimation of LIKE clauses. However, they cannot handle general forms of LIKE clauses with both '_' and '%'. Moreoever, none of these algorithms deal with selectivity estimation with edit distance or other similarity measures.

Our algorithm LBS adapts from set hashing signatures. A simpler scheme could be based on mapping a value v to a bit position ($v \mod m$). This is used in containment checking of an element with a set, and checking non-overlap between two sets. However, as observed in [1, 20], it performs poorly in estimating e-overlap between two sets for $e > 1$, where e-overlap is satisfied if there are at least e number of overlapping elements between two sets. The nature of our problem of estimating the overlap between two groups concerns a value of e typically much larger than 1.

Approximate string matching has been widely studied across various fields including text retrieval [2, 21], and data cleaning [22]. Jin et al. proposed MAT trees to process fuzzy predicates on strings [14]. Chaudhuri et al. developed an index structure and a fuzzy matching algorithm to effectively filter incoming tuples [7]. Estimating the cardinality of approximate string predicates is essential in all these tasks.

There have been extensive studies on approximate string matching in the computational biology community [12]. However, we believe that the string matching studies in bioinformatics are different in at least two key aspects. First, for efficient selectivity estimation, we rely on the SIS assumption for database applications. It is doubtful whether the SIS assumption holds in a typical bioinformatics application. Second, for database applications, the alphabet size is typically much larger than 4 in DNA sequences. The larger alphabet size presents a harder problem to deal with, particularly in terms of efficiency.

2. PRELIMINARIES

We first introduce two closely related techniques on which the proposed algorithms are based.

2.1 Extended Q-grams with Wildcards

Let Σ be a finite alphabet with size denoted as $|\Sigma|$. Any string s of length $q(> 0)$ in Σ^* is called a q-gram. An N-gram table over DB consists of the frequencies of all q-grams for $1 \leq q \leq N$ [5]. In [18], Lee et al. proposed the *extended N-gram table*, which generalizes the N-gram table with the wildcard symbol '?' for string selectivity estimation. Any

string of length $q(> 0)$ in $(\Sigma \cup \{?\})^*$ is called an *extended q-gram*. Extended q-grams capture various forms of strings with the wildcard symbol. For instance, the 3-gram table for the string "beau" contains 1-gram (i.e., for b, e, a, u individually), 2-grams (i.e., be, ea, au), and 3-grams (i.e., bea, eau). In an extended 3-gram table, the additional 2-grams are ?e, ?a, ?u, b?, e?, a? and ??. The additional 3-grams include (non-exhaustively) ?ea, e?? and etc. The entry ?ea gives the frequency of strings that include a substring of length 3 ending in ea. The entry e?? gives the frequency of strings that contain a substring that begins with e followed by any two characters.

For edit distance, edit operations considered are deletion (D), insertion (I) and replacement (R). A query Q is a pair $Q \equiv (s_q, \Delta)$, where s_q is the query string and Δ is the edit distance threshold. For any substring b such that $ed(b, s_q) \leq \Delta$ and insertion/replacement modeled by '?' (e.g., 'nua', 'nub', etc. are modeled by 'nu?'), b is called a *base substring* of Q. Base substrings represent possible forms of substrings satisfying the query. Then the set of tuple ids in DB that have a substring which can be converted to s_q with at most Δ edit operations is:

$$Ans(s_q, \Delta) = \bigcup_b G_b, \tag{1}$$

for all base substrings b, and G_b denotes the set of tuple ids of string s in DB containing b as a substring. [1] For example, $Ans(sylvie, 1)$ contains strings like "sylvia carbonetto", or "sylvester", but not "cecilia van den berg".

To compute $|Ans(s_q, \Delta)|$, the size of the answer set, all possible base substrings b are enumerated. Specifically, when the length of s_q is l, the base substrings vary in length from $(l - \Delta)$ to $(l + \Delta)$. For each possible length, we consider all $iDjIkR$ combinations, where $iDjIkR$ denotes i deletion, j insertion and k replacement operations, with $i + j + k = \Delta$ and $i, j, k \geq 0$. For example, the base substrings of $Q \equiv (van, 1)$ can be partitioned into 3 groups of length of 2, 3 and 4. Each group is from $1D$, $1R$, or $1I$ edit operation respectively. The group of $1I$ consists of $?van$, $v?an$, $va?n$ and $van?$. The group of $1R$ consists of $va?$, $v?n$, and $?an$. The desired answer $|Ans(van, 1)|$ is $|G_{va} \cup G_{an} \cup G_{vn} \cup \ldots \cup G_{?van} \cup \ldots|$.

The above description considers all possible base strings and combinations for illustration purposes. In practice, particularly for larger edit distance thresholds Δ and long query substrings, a sampling strategy can be applied. However, Equation (1) still requires the accurate estimation of the overlaps among the group G_b's.

2.2 Set Hashing

The estimation of the size of the union $\bigcup_b G_b$, where b are the base substrings, depends on how similar the sets within the union are. This leads to the notion of *resemblance* between two sets A and B, defined as:

$$\gamma = \frac{|A \cap B|}{|A \cup B|}. \tag{2}$$

Min-wise independent permutation is a well-known Monte-Carlo technique that estimates set resemblance. Based on a probabilistic analysis, Cohen proposed unbiased estimators to estimate the size of a set by repeatedly assigning random ranks to the universe and keeping the minimum values from the ranks of a set [9]. Each assignment of random ranks to the universe is a random permutation. The minimum values from the permuted ranks of a set kept for each permutation is called the *signature vector* and can also be used to estimate the set resemblance [9, 3].

[1] Because DB is a bag of strings, DB may contain duplicates of the same string s. We assume that each of those duplicates has its own distinct tuple id. This treatment is consistent with the studies in [5, 15, 18].

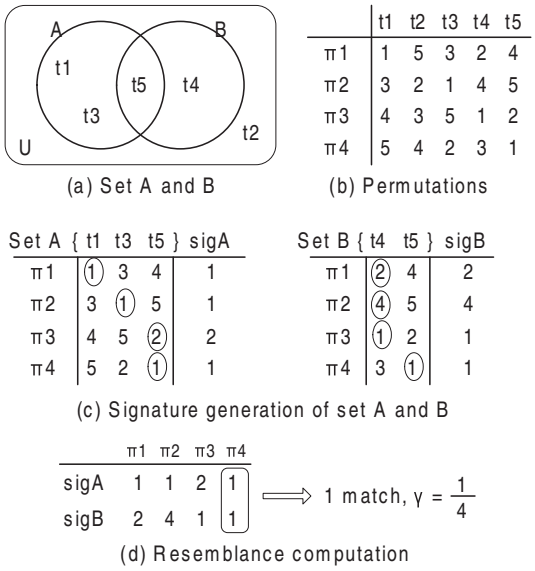


Figure 1: Set Resemblance Example

Figure 1(a) gives an example of two sets $A = \{t_1, t_3, t_5\}$ and $B = \{t_4, t_5\}$. The entire universe of tuple ids is $U = \{1, \ldots, 5\}$. To make the example clearer, we use $t_1, \ldots, t_5$ to denote tuple id $1, \ldots, 5$. Figure 1(b) shows four random permutations $\pi_1, \ldots, \pi_4$. For each of these permutations, Figure 1(c) shows the permuted ranks of the elements of A and B. For instance, for set A, π_1 maps t_1, t_3, t_5 to 1, 3, 4 respectively. The minimum value of the mapped three values for A is 1. Thus, under π_1, the signature value of A is 1. Similarly, the signature values of other permutations are computed. Finally, by doing an equality matching on each dimension of the two signatures in Figure 1(d), the set resemblance is estimated to be 1 out of 4. The true resemblance turns out to be exactly $\frac{|\{t_5\}|}{|\{t_1,t_3,t_4,t_5\}|} = \frac{1}{4}$.

Let us give the formal definitions below. Consider a set of random permutations $\Pi = \{\pi_1, \ldots, \pi_L\}$ on a universe $U \equiv \{1, \ldots, M\}$ and a set $A \subset U$. Let $min(\pi_i(A))$ denote $min(\{\pi_i(x) | x \in A\})$. Π is called *min-wise independent* if for any subset $A \subset U$ and any $x \in A$, when π_i is chosen at random in Π, we have $Pr(min(\pi_i(A)) = \pi_i(x)) = \frac{1}{|A|}$ [4]. Then with respect to two sets A and B, if Π is min-wise independent, $Pr(min(\pi_i(A)) = min(\pi_i(B))) = \gamma$ where γ is the resemblance defined in equation (2).

To estimate γ (denoted as $\hat{\gamma}$), the signature vector of A is constructed as: $sig_A = [min(\pi_1(A)), \ldots, min(\pi_L(A))]$ and similarly for sig_B. The i^{th} entry of vector sig_A is denoted as $sig_A[i]$, i.e., $min(\pi_i(A))$. By matching the signatures sig_A and sig_B per permutation, γ can be approximated as:

$$\hat{\gamma} = \frac{|\{i \mid sig_A[i] = sig_B[i]\}|}{L} \tag{3}$$

The equations above generalize to set resemblance among multiple sets. In practice, L does not need to be big for $\hat{\gamma}$ to be a good estimate of γ. In testing similarity among web documents [3], 100 samples were considered to be good enough. Chen et. al used 50 for L to process boolean predicates [8].

Chen et al. generalized the above idea to estimate the size of boolean queries on sets including intersection, union and negation [8]. The size of the union of m sets $|A_1 \cup \ldots \cup A_m|$ can be estimated as follows. Let $A = A_1 \cup \ldots \cup A_m$ and $B = A_j$ for any $A_j \in \{A_1, \ldots, A_m\}$. By Equation (2), $\gamma = \frac{|A \cap A_j|}{|A \cup A_j|}$. But as $A \cap A_j = A_j$ and $A \cup A_j = A$, we get:

$$|A| = |A_1 \cup \cdots \cup A_m| = \frac{|A_j|}{\gamma}. \quad (4)$$

A_j in Equation (4) can be any $A_j \in \{A_1, \ldots, A_m\}$, but the one whose size is the biggest generally gives the best performance [8]. To estimate γ, the signature of union, sig_A, can be constructed with the individual signatures as follows:

$$sig_A[i] = \min(sig_{A_1}[i], \cdots, sig_{A_m}[i]) \quad (5)$$

Let us return to the example in Figure 1. Based on Figure 1(d) and Equation (5), the constructed signature $sig_{A \cup B}$ is [1,1,1,1]. Now applying Equation (3) to estimate the resemblance between $A \cup B$ and A gives $\hat{\gamma} = 3/4$ ([1,1,1,1] vs. [1,1,2,1]). Finally, by Equation (4), the size of $|A \cup B| = \frac{|A|}{\hat{\gamma}} = \frac{3}{3/4} = 4$. This turns out to be exact, as $A \cup B = \{t_1, t_3, t_4, t_5\}$.

As will be shown later, applying the above set hashing equations is not sufficient for our task. We extend the above scheme in two key ways in Section 4.

3. ESTIMATION WITHOUT SIGNATURES

3.1 MOF: The MOst Frequent Minimal Base String Method

The first method we propose, called MOF, is based on extended q-grams. Recall from Equation (1) and the corresponding discussion that $Ans(Q) = \bigcup_b G_b$ for all the base substrings b of Q. However, it is obvious that given two base substrings b_1 and b_2, it is $G_{b_2} \subseteq G_{b_1}$ if b_1 is a substring of b_2 (e.g., $b_1 = va$, $b_2 = va?$ in the earlier example). We define a base substring b_i as *minimal* if there is not any other base substring b_j with $i \neq j$ that is substring of b_i. Thus, Equation (1) can be simplified to:

$$|Ans(s_q, \Delta)| = |\cup_{b \in MB} G_b| \quad (6)$$

where MB is the set of all the minimal base substrings.

EXAMPLE 2. *Let us consider the example of $Q \equiv (boat, 2)$. Possible base substring length varies from 2 to 6, which correspond to the situations with 2 deletions and 2 insertions respectively. The following table enumerates all the possibilities for illustration purposes.*

Starting from the base substrings of length 2, we eliminate base substrings that contain another base substring. For instance, 'bo' is a substring of 'bo?', 'boa?', 'bo?a', and etc. After removing redundant base substrings, the set of remaining minimal base substrings is MB = {'bo', 'ba', 'bt', 'oa', 'ot', 'at', 'b?a', 'b?t', 'b?a', 'o?t', 'b??t'}. Thus, $|Ans(boat, 2)| = |G_{bo} \cup G_{ba} \cup G_{bt} \cup G_{oa} \cup G_{ot} \cup G_{at} \cup G_{b?a} \cup G_{b?t} \cup G_{b?a} \cup G_{o?t} \cup G_{b??t}|$.

2D	*bo*	*ba*	*bt*	*oa*	*ot*	*at*
1D1R	~~*bo?*~~	*b?a*	~~*?oa*~~		*(from boa)*	
	~~*bo?*~~	*b?t*	~~*?ot*~~		*(from bot)*	
	~~*bt?*~~	*b?a*	~~*?ta*~~		*(from bta)*	
	~~*oa?*~~	*o?t*	~~*?at*~~		*(from oat)*	
1D1I	~~*boa?*~~	~~*bo?a*~~	~~*b?oa*~~	~~*?boa*~~	*(from boa)*	
	~~*bot?*~~	~~*bo?t*~~	~~*b?ot*~~	~~*?bot*~~	*(from bot)*	
	~~*bat?*~~	~~*ba?t*~~	~~*b?at*~~	~~*?bat*~~	*(from bat)*	
	~~*oat?*~~	~~*oa?t*~~	~~*o?at*~~	~~*?oat*~~	*(from oat)*	
2R	~~*bo??*~~	~~*b?a?*~~	*b??t*	~~*?oa?*~~	~~*?o?t*~~	~~*??at*~~
1I1R	~~*boa??*~~	~~*bo?t?*~~	~~*b?at?*~~	~~*?oat?*~~	*(from boat?)*	
	~~*boa??*~~	~~*bo??t*~~	~~*b?a?t*~~	~~*?oa?t*~~	*(from boa?t)*	
			...			
	~~*bo?a?*~~	~~*?bo?t*~~	~~*?b?at*~~	~~*??oat*~~	*(from ?boat)*	
2I	~~*boat??*~~	~~*boat??*~~	~~*boat??*~~	~~*boat??*~~	*(from boat??)*	
			...			
	~~*??boat*~~	~~*??boat*~~	~~*??boat*~~	~~*??boat*~~	*(from ??boat)*	

Algorithm 1 shows an outline to find all the minimal base substrings. The first *for* loop from line (2) to (8) generates all possible base substrings. In the most general case when ℓ and Δ are large, the loop may be computationally expensive. However, as motivated by the Short Identifying Substring (SIS) assumption in [5], $\Delta \leq 3$ can find many database applications. For $\Delta \leq 3$, the following table enumerates all the combinations with deletions, insertions and replacements (ℓ being the length of the query substring). For example, for $\Delta = 3$, there are only two possibilities to obtain a substring of length $(\ell - 1)$, namely either by $2D1I$ or by $1D2R$. Thus, even for $\Delta = 3$, there are only 10 combinations to be considered with a complexity of $O(\ell^3)$.

	ℓ-3	ℓ-2	ℓ-1	ℓ	ℓ+1	ℓ+2	ℓ+3
$\Delta = 1$			1D	1R	1I		
$\Delta = 2$		2D	1D1R	2R, 1D1I	1I1R	2I	
$\Delta = 3$	3D	2D1R	2D1I, 1D2R	3R, 1D1I1R	1D2I, 1I2R	2I1R	3I

The *for* loops in the lines (9) to (14) finds all the minimal base substrings. The loop exploits the simple fact that a string cannot be a substring of something shorter. Finally, the set MB of all the minimal base substrings is returned. For situations when ℓ and Δ are larger, it is too expensive to fully implement line (6), and a sampling strategy is applied. The effectiveness of the sampling is evaluated in Section 5.

With the set MB computed, the MOF ("MOst Frequent") estimation algorithm uses the following heuristic based on the most frequent minimal base substring b_{max} among all the base substrings in MB:

$$MOF(Q) = \frac{|G_{b_{max}}|}{\rho} \quad (7)$$

The parameter ρ is called *coverage*. This simple heuristic is based on a generalization of the SIS assumption stating that: *"a query string s usually has a 'short' substring s' such that if an attribute value contains s', then the attribute value almost always contains s as well"* [5]. Our generalization states that a query substring s usually has an identifying minimal base substring s'. [2] The requirement of *almost always* in the SIS assumption is more realistically relaxed to a fraction ρ in our case; that is, a fraction ρ of the strings in the $Ans(s_q, \Delta)$ also contain the most frequent minimal base substring. As an example, $|Ans('sylvia', 1)|$ is 538 in

[2]Apart from the SIS generalization, the key difference between MOF and the CRT framework in [5] is that the former deals with edit distance.

Algorithm 1 MinimalBaseSubstrings

Input: query string s_q, edit distance threshold Δ
Output: the set MB of minimal base substrings

1: $C = \phi, MB = \phi$
2: **for all** ℓ from $(length(s_q) - \Delta)$ to $(length(s_q) + \Delta)$ {
3: Find all (i, j, k) s.t. $i+j+k = \Delta$ and $length(s_q)-i+j = \ell$
4: **for all** $c = (i, j, k)$ found in the above step {
5: $C = C \cup \{c\}$
6: B_c = the set of all base substrings with $iDjIkR$ operations
7: } /* for all */
8: } /* for all */
9: **for all** $c = (i, j, k) \in C$, $c' = (i', j', k') \in C$ s.t. $j'-i' > j-i$ {
10: **for all** $b \in B_c$ and $b' \in B_{c'}$ {
11: **if** b is a substring of b' {
12: $B_{c'} = B_{c'} - \{b'\}$ }
13: } /* for all */
14: } /* for all */
15: **for all** $c \in C$
16: $MB = MB \cup B_c$
17: **return** MB

Algorithm 2 S-SEPIA

Procedure Construct
Input: DB
Output: Clusters C, Global histogram PPD, Error correction module ECM

1: $SDB = \phi$
2: **for all** tuple $t \in DB$ {
3: Generate $(c * length(t))$ substrings from t; add to SDB
4: } /* for all */
5: Run SEPIA with SDB to construct C and PPD
6: Sample substring queries Q
7: Produce the error correction info ECM from Q using $freq_{est}$ from SDB and $freq_{true}$ from DB

Procedure Estimate
Input: Query string s_q, Edit distance threshold Δ
Output: $|Ans(s_q, \Delta)|$

1: Estimate $freq_{est}$ with s_q and Δ using C and PPD
2: Calculate $freq_{corrected}$ with s_q, Δ and $freq_{est}$ using ECM
3: **return** $freq_{corrected}$

the DBLP author names data set in Section 6, and typical variations of the query 'sylvia' are 'silvia' and 'sylvia' occurring 202 and 100 times repectively. So the base string 's?lvia' alone explains more than half of the answer set size. The validity of our assumption is extensively evaluated in Section 6 by MOF.

In MOF, as shown in Equation (7), a single default value ρ is used for simplicity. This default value can be obtained by sampling on the data set, which can be easily piggybacked when the extended N-gram table is being constructed.

For a base substring b, if the extended N-gram table kept by the system contains an entry for b (e.g., when $|b| \leq N$), then the frequency $|G_b|$ is immediately returned. Otherwise, $|G_b|$ needs to be estimated using a substring selectivity estimation algorithm like MO [13].

3.2 Algorithms not Based on Extended Q-grams

3.2.1 S-SEPIA: a Method based on Clustering

For string selectivity estimation, Jin and Li [15] proposed the SEPIA algorithm. It clusters the strings in the database and use histograms to store distribution information for each cluster. We adapt SEPIA to substring selectivity estimation problem by building clusters of substrings rather than strings. However, if we were to build clusters based on all the substrings contained in the database, it would be infeasible for large databases. Thus, we apply random sampling on substrings. Algorithm 2 shows a skeleton of S-SEPIA which is a simple adaptation of SEPIA to substrings. In the construction of clusters, the loop in lines (2) to (4) randomly extracts $c * \ell$ substrings for each string where ℓ is the length of the string. Line (4) runs the normal SEPIA construction procedure to build the clusters and the corresponding histograms.

Another complication in adapting SEPIA to the substring problem arises from the counting semantics. As discussed in [13], there is the difference between *presence* counting and *occurrence* counting. If the tuple string is "Vancouver Van Rental" and the query substring is "Van", presence counting gives a value 1 to show that the substring is present in the tuple, whereas occurrence counting gives a value of 2 to indicate that the substring occurs twice in the tuple. For substring selectivity estimation, the intended semantics is presence counting, whereas the construction procedure of S-SEPIA essentially conducts occurrence counting.

To address this issue, we adapt the error-correction phase in SEPIA as follows. In lines (6) and (7), queries are randomly sampled to build a table giving a correction factor between presence and occurrence counting, according to query length, threshold, and frequency range. Then, during estimation time, line (2) of the Estimate procedure adjusts the estimated frequency based on this table.

3.2.2 RS: a Method with No Space Overhead

Note that MOF requires space overhead in the form of the extended N-gram table. S-SEPIA also incurs space overhead in the form of the histograms and other auxiliary information associated with the clusters. The algorithm called RS, which stands for "Random Sampling", represents another extreme. It relies on random sampling from DB at query time. As such, it does not create any intermediate "compile-time" data structure and incurs no space overhead. Specifically, given a query substring s_q, all it does is to randomly sample a fixed number of strings s from DB and checks the percentage of them such that $ed(s_q, b) \leq \Delta$, where b is a substring of s. It is conceivable that RS may incur significant query time computation, particularly for larger edit distance threshold Δ and length l. In the experimental results section later, we examine the trade-offs between query time computation, space overhead and estimation accuracy.

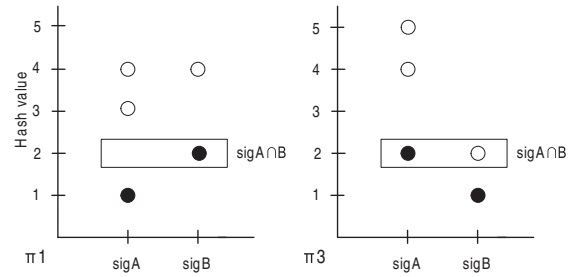
4. ESTIMATION WITH SET HASHING SIGNATURES

4.1 LBS: Lower Bound Estimation

While MOF estimation is simple and efficient, a key weakness is that the estimation is based on a single minimal base substring which is the most frequent. In general, the estimation may be more accurate if many, if not all of, the minimal base substrings are used. However, if multiple minimal base substrings are used, we need to estimate the size of the union $|\cup_{b \in MB} G_b|$ in Equation (6). This can be done by applying Equation (4) to obtain

$$LBS(Q) = |\bigcup_{b \in MB} G_b| \approx \frac{|G_{b_{max}}|}{\gamma} \quad (8)$$

where b_{max} is the most frequent minimal base string in MB.



(a) LBS estimation of the 1st and 3rd signature values by π1 and π3

	π1	π2	π3	π4	
sigA	1	1	2	1	
sigB	2	4	1	1	
sig A∩B	2	4	2	1	By LBS

(b) LBS estimation of the signature of A ∩B

Figure 2: An illustration of LBS

Algorithm 3 LBS Estimation

Input: query s_q, edit distance threshold Δ, maximum extended q-gram length N, default resemblance ρ
Output: $|Ans(s_q, \Delta)|$

```
1: MB = MinimalBaseSubstrings(s_q, Δ) (Algo. 2)
2: freq_max = 0, sig_max = null, sig_union = null
3: for all b ∈ MB {
4:   if length(b) > N {
5:     Decompose b into a set of substrings s_{b,i} of length N
6:     Compute sig_b by applying Equation (9) with sig_{s_{b,i}}
7:     Calculate freq_b (i.e. |G_b|) using MO
8:   }
9:   if freq_b > freq_max {
10:    freq_max = freq_b, sig_max = sig_b
11:  }
12:  sig_union = Union of sig_union and sig_b as in Eqn. (5)
13: } /* for all */
14: Compute γ̂ with sig_max and sig_union as in Eqn. (3)
15: if γ̂ = 0 { γ̂ = ρ }
16: return (freq_max/γ̂) as in Equation (8)
```

Note that $|G_{b_{max}}|$ is exactly the one used in the numerator of the MOF estimation in Equation (7). The only difference is that in MOF, ρ is a single default coverage computed by sampling for each data set. In contrast, the resemblance γ in the above equation is calculated by taking into account all the minimal base strings, making it more specific to the query. It is also possible that the resemblance γ is estimated to be zero for some extreme cases, in which case a default resemblance, like ρ, is used instead since we cannot divide by zero in Equation (8).

To compute $LBS(Q)$ with Equation (8), we need the values of $|G_{b_{max}}|$ and γ. As mentioned earlier, the former can be estimated using MO if it is not kept in the extended N-gram table. However, the difficulty here is to compute the approximation of γ by Equation (3) which requires the signatures to be compared for matching, i.e., we need $sig_b(\equiv sig_{G_b})$ for all $b \in MB$ to compute the approximation of γ. But there may be base substrings that are too long to be kept in the N-gram table. For instance, suppose that a base substring $b \equiv$ 'database' and an extended N-gram table with $N = 5$ is maintained.

Although set hashing technique [8] is able to estimate the size of intersection or union of strings when the query strings are not kept in the summary structure (PST or N-gram table), the algorithm has exponential complexity. It is computationally manageable when the number of terms is expected to be small as in the boolean query problem, but the complexity will be unacceptable in the approximate substring problem where the number of possible forms of substring could be quite large.

Our solution is to rely on the substrings of b stored as extended q-grams in the N-gram table. Specifically, by letting $b_1, \ldots, b_w$ be all the substrings of b of length N stored in the N-gram table with $w = \ell - N + 1$, we can approximate the signature sig_b based on the *individual signatures* $sig_{b_1}, \ldots, sig_{b_w}$. For instance, we calculate the signature $sig_{database}$ based on $sig_{datab}, sig_{ataba}, sig_{tabas}$ and sig_{abase}.

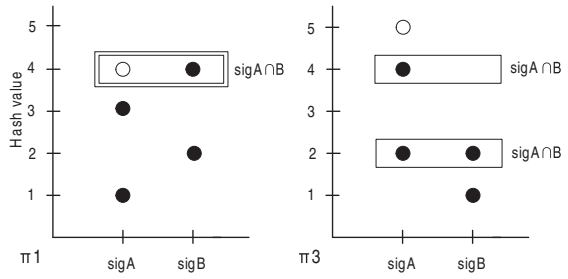
To illustrate our approach, let us consider an example with $w = 2$. Suppose that we have $b \equiv$ 'data', $b_1 \equiv$ 'dat' and $b_2 \equiv$ 'ata', and the tuples containing 'dat' are exactly $\{t_1, t_3, t_5\}$ as captured by set A in Figure 1. Similarly, suppose that the tuples containing 'ata' are exactly $\{t_4, t_5\}$ as modeled by set B in Figure 1. We assume that the 4 random permutations $\pi_1, \pi_2, \pi_3, \pi_4$ in Figure 1 are still used. Now for the signatures, the permuted values of set $A = \{t_1, t_3, t_5\}$ under permutation π_1 is $1, 3, 4$ as shown in Figure 1, which is presented as circles in the left diagram of Figure 2(a). However, because a signature only retains the minimum value for each permutation, only the value 1 is retained (shown as a solid circle, as opposed to the unfilled circles). The situation is similar for set B, as shown by the solid circle in the second column of the left diagram in Figure 2(a).

Let us consider how $sig_{A\cap B}$ can be estimated for π_1 using sig_A and sig_B only. Since each signature keeps the minimum value for each permutation, i.e. 1 for sig_A and 2 for sig_B, it is not possible to figure out the exact minimum value for the permutation of the intersection $A \cap B$ using sig_A and sig_B only. Instead, we try to infer, as tightly as possible, a lower bound of the minimum value for the permutation of the intersection.

The value 1 is not possible for $sig_{A\cap B}$ because if this were the case, the value for sig_B should be 1 as well, instead of 2. On the other hand, 2 is still a possible value for $sig_{A\cap B}$ because all we know from sig_A is that the minimum value is 1. Thus, given just the two solid circles in the left diagram in Figure 2(a), the best inferred signature of $sig_{A\cap B}$ we can get under permutation π_1 is 2. Actually, $A \cap B$ has only t_5 which is mapped to 4. This is shown by the two unfilled circles in Figure 2(a), under the permutation π_1. It is easy to see that the inferred value 2 is a lower bound of the actual value.

The right diagram in Figure 2(a) shows the signatures for permutation π_3 in Figure 1. A similar reasoning determines that the value 2 is the inferred signature value of $sig_{A\cap B}$ under permutation π_3. This turns out to be the correct signature value because of the matched unfilled circle in the second column of the right diagram.

Figure 2(b) shows the inferred $sig_{A\cap B}$, [2, 4, 2, 1], for all the four permutations that are computed by selecting the maximum value of the corresponding sig_A and sig_B values for each permutation. The true signature $sig_{A\cap B}(= sig_{\{5\}})$ is actually [4, 5, 2, 1], as shown in Figure 1. As we mentioned previously, the inferred values are always lower bounds to the true values. This observation can be generalized to compute



(a) LBS estimation of the 1st and 3rd signature values by π1 and π3

	π1	π2	π3	π4	(1st,2nd)
sigA	(1,3)	(1,3)	(2,4)	(1,2)	
sigB	(2,4)	(4,5)	(1,2)	(1,3)	
sig A∩B	(4,4)	(4,4)	(2,4)	(1,3)	By LBS

(b) LBS estimation with two minima

Figure 3: LBS using the First and Second Minima

the intersection of w signatures for any permutation $i, 1 \leq i \leq L$:

$$\hat{sig}_{b_1 \cap \ldots \cap b_w}[i] = max(sig_{b_1}[i], \cdots, sig_{b_w}[i]) \quad (9)$$

To verify, suppose that there is a $sig_{b_i}[k]$ for $1 \leq k \leq L$ such that $sig_{b_i}[k] < sig_{b_j}[k]$. Then the value $sig_{b_i}[k]$ cannot be the true signature value of $sig_{b_1 \cap \ldots \cap b_w}[k]$. Otherwise, the signature value $sig_{b_j}[k]$ must be the same as $sig_{b_i}[k]$. By this argument, Equation (9) forms a lower bound estimation for $sig_{b_1 \cap \ldots \cap b_w}$.

Algorithm 3 sketches the outline of the LBS ("Lower Bound eStimation") estimation algorithm. It first computes all the minimal base substrings. For each such substring, if it is too long to have kept in the extended N-gram table, lines (4) to (8) estimate its frequency and signature. All these signatures are combined together in line (12) for the eventual computation of the resemblance $\hat{\gamma}$ in line (14). Finally, in line (16), the estimate is returned.

4.2 Improving LBS with Extra Minima

Recall from Equation (9) that the estimated signature value is a lower bound to the true signature value. One possibility to improve the lower bound is to keep extra minimum permuted values such as the second minimum, the third minimum, etc. Certainly, each additional minimum value kept increases the size of the signature linearly. Below we describe the mechanics.

So far we have used the notation sig_A to denote the signature of set A under L permutations. To introduce additional minimum values kept in the signature, we use $sig_{A,1}$, $sig_{A,2}, \ldots$ to denote the first minimum, the second minimum and so on in the signatures. (In other words, sig_A used previously is now equivalent to $sig_{A,1}$.) Figure 3 gives an illustration of how to compute the signature with keeping the second minimum value for an intersection of two sets, i.e., $sig_{A \cap B,1}$ and $sig_{A \cap B,2}$ from $sig_{A,1}, sig_{A,2}, sig_{B,1}$ and $sig_{B,2}$. This is a continuation of the situation shown in Figure 2. What is different in Figure 3 is that there are now two solid circles for each set, corresponding to the first and second minima. Recall from Figure 2(a) that with only the first minimum kept, $\hat{sig}_{A \cap B,1}[1] = max(sig_{A,1}[1], sig_{B,1}[1]) = 2$. But with two minima kept, as shown in Figure 3, it is clear that no element in $A \cap B$ can have a hash value of 2 since $sig_{A,1}[1] < 2 < sig_{A,2}[1]$. If there had been an $x \in A$ such that $\pi_1(x) = 2$, then we would have chosen 2 as $sig_{A,2}[1]$ not 3. Similarly, because $sig_{B,1}[1] < 3 < sig_{B,2}[1]$, no element in $A \cap B$ can have a hash value of 3. Thus, we can infer $sig_{A \cap B,1}[1] = 4$, which turns out to be the correct value in this example. As there is no additional minimum values kept in $sig_A[1]$ and $sig_B[1]$, we cannot infer any better lower bound of the second minimum value of $sig_{A \cap B}[1]$ and thus the second minimum value for $A \cap B$ is set to $\hat{sig}_{A \cap B,2}[1] = \hat{sig}_{A \cap B,1}[1] = 4$. This is summarized in Figure 3(b) under the column for the permutation π_1, i.e., (1,3) and (2,4) lead to (4,4).

The right diagram in Figure 3(a) shows the case for permutation π_3. Figure 3(b) also shows the results for permutations π_2 and π_4. The table in Figure 4 enumerates all the possible combinations with the first and second minima kept. To save space, we do not present the general formula which can handle the case when $k_{min} \geq 2$ minima are kept. Hereafter, we use k_{min} to denote the number of minima kept in the signature. In the next section, we will evaluate the effectiveness of LBS with k_{min} varying from 1 to 3.

Condition		Estimation	
$sig_{A,1}[i]$	$sig_{A,2}[i]$	$sig_{A \cap B,1}[i]$	$sig_{A \cap B,2}[i]$
$= sig_{B,1}[i]$	$= sig_{B,2}[i]$	$sig_{A,1}[i]$	$sig_{A,2}[i]$
$\neq sig_{B,1}[i]$	$= sig_{B,2}[i]$	$sig_{A,2}[i]$	$sig_{A,2}[i]$
$= sig_{B,1}[i]$	$\neq sig_{B,2}[i]$	$sig_{A,1}[i]$	$\max(sig_{A,2}[i], sig_{B,2}[i])$
$\geq sig_{B,2}[i]$		$sig_{A,1}[i]$	$sig_{A,2}[i]$
	$\leq sig_{B,1}[i]$	$sig_{B,1}[i]$	$sig_{B,2}[i]$
Otherwise		$\max(sig_{A,2}[i], sig_{B,2}[i])$	$\max(sig_{A,2}[i], sig_{B,2}[i])$

Figure 4: Estimation of Signatures

While the discussion so far is based on the intersection of two sets A and B, the computation can be modeled as a binary operator. When more than two sets are intersected, this binary operator can be applied successively. Note that the LBS estimation algorithm shown earlier does not change, except for line (6). This line is now generalized to compute the first and second minima of the signature. However, only the first minimum values are used to estimate frequency and resemblance in subsequent lines. For our example, if we focus our attention on $\hat{sig}_{A \cap B,1}$ for all the four permutations, the signature becomes [4,4,2,1], which is a closer match to the true signature of [4,5,2,1] than [2,4,2,1] in Figure 2(b) with only the first minimum. This shows that extra minima may help to better approximate signatures for intersections.

5. EXTENSIONS TO OTHER SIMILARITY MEASURES

Although our discussion so far has focused on edit distance, LBS can support various similarity measures.

5.1 SQL LIKE Clauses

We will consider two major constructs of LIKE: '%' and '_'. '%' character matches any substring and '_' character matches any single character.

Suppose a LIKE pattern $\%s_1\%...\%s_m\%$. For example, the predicate "*name* like %silvia%carbonetto%" selects all

the names that contain *silvia* followed by *carbonetto* with any number of characters in between.

Assume for now that each s_i $(1 \leq i \leq m)$ is a plain substring without any special character like '_'. Any string that satisfies the LIKE condition must have all of s_is, $1 \leq i \leq m$ as its substring. Recall that G_b denotes the set of tuple ids in the DB that have b as a substring. Then the size of the set of tuple ids in DB that match the LIKE condition $s_{like} \equiv \%s_1\%...\%s_m\%$ can be estimated as:

$$|\hat{Ans}(s_{like})| = |\bigcap_{1 \leq i \leq m} G_{s_i}|. \quad (10)$$

This is an upper bound of the true selectivity but a tighter bound than $min|G_{s_i}|$ proposed in [5]. Note that LBS utilizes signature and it can handle intersection as well as union. We make use of the next formula for intersection size estimation [8].

$$|G_{s_1} \cap \cdots \cap G_{s_m}| = \gamma_m \cdot |G_{s_1} \cup \cdots \cup G_{s_m}| \approx \hat{\gamma}_m \cdot |\hat{G}|, \quad (11)$$

where γ_m is the resemblance of $G_{s_1}, \ldots, G_{s_m}$ which can be calculated by Equation (3) extended to m sets. $|\hat{G}|$ is the estimated union size and it is actually the estimation of LBS, Equation (8). Thus our estimation of Equation (10) is the output of LBS multiplied by $\hat{\gamma}_m$. We only highlight key modifications due to space restriction.

1. It generates base substrings for each s_i setting $\Delta = 0$ in line (2) to (8) of Algorithm 1. (i.e., each s_i becomes one base substring.)

2. It returns the value in line (16) of Algorithm 3 multiplied by γ_m.

If s_i contains '_', which matches any single character, we only need to substitute '_' with our wildcard '?' when generating base substrings.

One interesting observation is that γ_m can be quite small in Equation (11). To estimate γ_m in the style of Equation (3), we would need a larger L, the number of random permutations in set hashing, for small γ_m. Note that small L is enough in Equation (4). $\gamma = \frac{|A_j|}{|A|}$ is generally not small since $|A_j|$, the frequency of most frequent base substring, is not a small fraction of $|A|$ from the generalized SIS assumption. However, in Equation (11), we cannot make the same assumption on the relative size of $|G_{s_1} \cap \cdots \cap G_{s_m}|$ and $|G_{s_1} \cup \cdots \cup G_{s_m}|$. For example, in the query %silvia%carbonetto%, $freq(silvia)$ may turn out to be very high whereas $freq(carbonetto)$ can be relatively low. If $freq(silvia) = 10,000$ and $freq(carbonetto) = 10$, $\gamma = \frac{|G_{silvia} \cap G_{carbonetto}|}{|G_{silvia} \cup G_{carbonetto}|} \leq \frac{|G_{carbonetto}|}{|G_{silvia}|} = \frac{10}{10,000}$ and L should be at least 1,000 to express the small γ. Let $f_{min} = min|G_{s_i}|$ and $f_{max} = max|G_{s_i}|$. Whenever $|\hat{G}|/f_{min} > L$ or $\hat{\gamma}_m = 0$, we know that our estimation is likely to be inaccurate. Thus, before multiplying the output of LBS and $\hat{\gamma}_m$ we check for the two conditions. If one of them is true, then $\hat{\gamma}_m$ is set to f_{min}/f_{max}. Experimental results on LIKE predicates in Section 6 reflect this heuristic.

5.2 Jaccard Coefficient

The Jaccard similarity [10] of two strings a and b is defined as:

$$\mathcal{J}(a,b) = \mathcal{J}(A,B) = \frac{|A \cap B|}{|A \cup B|},$$

where A and B are sets of q-grams of a and b respectively. For instance, when $q = 3$, $\mathcal{J}$('tyrannosaurus', 'allosaurus') $= \mathcal{J}(\{$ *t*yr, yra, ran, ann, nno, nos, osa, sau, aur, uru, rus $\}, \{$ *a*ll, llo, osa, sau, aur, uru, rus $\}) = \frac{5}{13}$. Our extension starts from the intuition that if two sets are similar, their sizes cannot be too different. If $\mathcal{J}(a,b) = \gamma$, then $\gamma \cdot |A| \leq |B| \leq 1/\gamma \cdot |A|$ [1]. Substituting $|A| = length(a) - q + 1$ and $|B| = length(b) - q + 1$ into the above inequality gives us

$$\lceil \gamma \cdot (length(a) - q + 1) \rceil \leq length(b) - q + 1 \leq \lfloor 1/\gamma \cdot (length(a) - q + 1) \rfloor$$

Using this property, given a query string s_q and a Jaccard similarity threshold γ, we can derive conditions on the length of strings that satisfy the similarity condition. We use this minimum and maximum string length in line (2) of Algorithm 1 and generate base substrings in the same way. However, not all the base substrings generated such have $\mathcal{J} \geq \gamma$, so we filter base substrings treating the wildcard as a separate character. Before line (12) of Algorithm 3, we check every base substring b and see if $\mathcal{J}(s_q, b) \geq \gamma$. If it is smaller than γ, we discard the base substring. So when we compute the union size, we only consider those base substrings that are guaranteed to have $\mathcal{J} \geq \gamma$.

6. EMPIRICAL EVALUATION

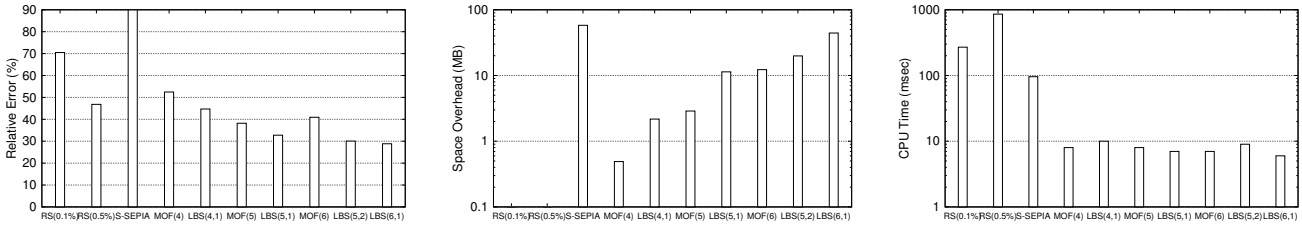
6.1 Experimental Setup

Data sets: We perform a series of experiments using three benchmarks: DBLP author names and titles and IMDB movie Keywords data. There are 699,199 full names in the DBLP authors data set. The average and maximum length are 14.1 and 38 respectively. There are 305,364 titles in the DBLP titles data set. The average and maximum length are 58.6 and 100 respectively. Finally, there are 100,000 keywords in the IMDB Keywords data set, with the average and maximum length being 10.3 and 62. For each data set, we generate random queries $Q \equiv (s_q, \Delta)$. These queries are divided into two overlapping query sets:

- The **short** query set consists of query string s_q which is a substring of a random word of length 5 to 12 in any tuple. When the word length is 7 or less, we use the whole word. There are 200 queries in this set. The edit distance threshold Δ is set to 25% of the query string length (i.e., $\Delta \leq 3$). As motivated by the Short Identifying Substring (SIS) assumption in [5], $\Delta \leq 3$ can find many database applications. The average selectivity is 1.03% for the Keyword data set and 0.55% for the DBLP authors data set.

- The **long** query set consists of query string s_q which is a random word of length 10 to 20 in any tuple. The long set is the most meaningful for the DBLP titles data set. There are 100 queries in this set. The edit distance threshold Δ is again set to 25% of the query string length (i.e., $3 \leq \Delta \leq 5$). The average selectivity is 3.17%.

- The **negative** query set consists of queries whose true frequency is 0. We randomly choose a word in a tuple and replace up to 3 random positions with random



(a) Average Relative Error **(b) Space Overhead** **(c) CPU Query Time**

Figure 5: Short Query Set on DBLP Authors

characters. Out of 200 chosen queries, 52 have true frequency of 0. Notice that there might be tuples that match the newly formed query by chance. This type of query is important especially given the issue of unclean data [17]. If a query optimizer can accurately identify true negative predicates or the estimated frequency is fairly accurate (i.e., the estimation is close to zero), the predicate will remain the most selective condition for query processing.

Evaluation metric: To evaluate the accuracy of an estimation method we rely on three different metrics. The first metric we employ is *relative error* which is defined as $|f_{est} - f_{true}|/f_{true}$, where f_{true} is the true frequency of the query and f_{est} is the estimated frequency. To prevent the accuracy from being distorted by queries with small true frequencies, we exclude queries whose true frequency is less than or equal to 10. For such small queries, the second metric, absolute error $|f_{est} - f_{true}|$ is employed. It is also used to report errors of negative queries to avoid division by 0. The third metric we apply is the *relative error distribution*. We show the distribution by providing a histogram of the relative errors, i.e., [-100%,-75%), [-75%,-50%), etc.

Estimation methods implemented: MOF is parameterized by the extended N-gram table, where N was varied from 4 to 6. As discussed in Section 3.1, the default coverage ρ was acquired by sampling. The following table shows the average value and the standard deviation of ρ using the DBLP Authors data set. The table shows that the average ρ is rather stable with respect to the sample size. Like

Sample Size	10	20	50	100
Average ρ	0.6453	0.6445	0.6523	0.6475
Standard deviation	0.081	0.071	0.034	0.000

MOF, LBS is parameterized by N. It is also parameterized by k_{min}, which controls the number of minimum values for a signature. It was varied from 1 to 3. Thus, the results of LBS are denoted by LBS(N,k_{min}). Strictly speaking, MOF and LBS are tunable by two other parameters PT and L, where PT determines the minimum frequency for a q-gram to be kept in the N-gram table and L controls the number of permutations used, i.e., from π_1 to π_L. For results reported here, we set $PT = 20$ and $L = 10$, unless specified. Finally, in LBS, the default value ρ is used when the estimated resemblance by signature is ≤ 0.2 and 10 most frequent base substrings are used in line (12) of Algorithm 3. We implemented our set hashing based techniques in Java 1.5, and a hash space of 2^{15} was used.

S-SEPIA was implemented in C++ by modifying the SEPIA code downloaded from [11]. We used 2,000 clusters and the CLOSED_RAND [15] method to populate the histograms. To limit the building time to around 48 hours, we restricted the maximum number of sampled substrings per tuple to be 10, and set the sampling ratio according to the data sets. The space consumption was measured by the size of the data structure written on disk.

Finally, RS was also implemented in Java 1.5. Because RS requires sampling from the database DB, the I/O cost may vary depending on the buffering policy. To simplify query time comparisons, we only considered the CPU cost of RS. In other words, we under-estimated the true cost of running RS in practice; it was sufficient for the eventual conclusions.

All experiments were conducted on P4 3GHz machines with 2 GB memory running GNU/Linux with kernel 2.6.

6.2 Short Query Set: Estimation Accuracy vs Space Overhead vs Query Time

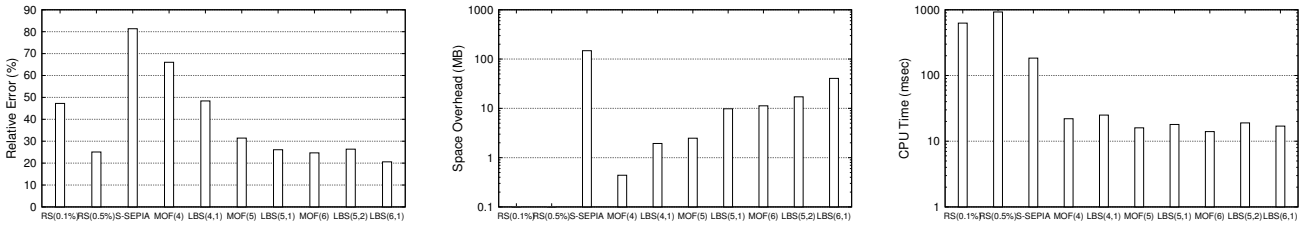
We begin with the short query set, i.e. $\Delta \leq 3$, for the DBLP author data set. Figure 5 shows the average relative error, and the corresponding space overhead and query time in milliseconds. The latter two graphs are drawn in log scale.

MOF vs S-SEPIA: Let us first focus on the comparison between MOF and S-SEPIA. S-SEPIA uses a sampling ratio of 0.25%. For DBLP authors, S-SEPIA gives an average relative error of 64% and uses close to 100 MB of memory, whereas MOF(4) gives a better average error of 53% but using only 0.5 MB! As N increases, MOF(5) and MOF(6) give significantly better relative average error. Yet the amount of space used as shown in Figure 5(b) is still significantly smaller than that used by S-SEPIA.

Recall that S-SEPIA uses the parameter c to control the number $c * \ell$ substrings of s to be used for clustering. The results reported in the figures so far are based on $c = 1$. We suppress the detailed results here, but point out that increasing c beyond 1 does not give clear empirical performance benefit.

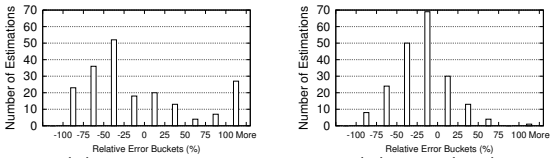
In terms of runtime, it takes around 30 and 120 minutes to build the extended 5-gram and 6-gram tables. But the building time for S-SEPIA is around 35 hours. For query estimation, as shown in Figure 5(c), the processing time of MOF is around 10 milliseconds. S-SEPIA's runtime is around 30 milliseconds. In sum, MOF outperforms S-SEPIA in providing higher average estimation accuracy, lower building time and less space overhead.

MOF vs RS: For method RS, the estimation accuracy and CPU time required are directly proportional to the amount of sampling done. We show in Figure 5 two settings of RS: with 0.1% or 0.5% of the whole database sampled. For estimation accuracy, RS(0.5%) gives an average relative



(a) Average Relative Error **(b) Space Overhead** **(c) CPU Query-Time**

Figure 6: Short Query Set on DBLP Titles



(a) S-SEPIA **(d) LBS(5,2)**

Figure 7: Error Distributions on DBLP Authors

error comparable to MOF(5) and MOF(6). However, all the MOF variants are almost two orders of magnitude faster. In practice, RS may take even longer if I/O cost is incurred. The faster version RS(0.1%) gives significantly higher average relative error. Thus, with a very modest space overhead, MOF dominates RS in providing either superior query time performance or higher estimation accuracy.

A variant of RS is to keep a sampled database around for query time. The hope is that the space overhead would significantly reduce query time. Unfortunately, the sampled database only reduces the time for sampling, not the time to check the edit distance threshold for all the substrings. The latter by far dominates the former.

MOF vs LBS: Next let us compare MOF with LBS on the trade-off between accuracy and space overhead. (The query time of the two algorithms are almost identical.) The ordering in terms of descending average relative error is: MOF(4) > LBS(4,1) > MOF(6) ≈ MOF(5) > LBS(5,1) > LBS(5,2) > LBS(6,1). The corresponding space overhead ordering is almost the reverse. This exactly shows how LBS and MOF can leverage extra space to give lower error. Specifically, the difference in size between the pair MOF(N) and LBS(N,1) is due to the use of signatures by LBS. To formally test whether the differences observed are statistically significant, we use the standard 2-tail Student's t-test to compute a p-value, i.e., the probability of a chance observation. The difference of LBS over MOF is well below 0.01 and is confirmed to be statistically significant. Moreover, even though there does not appear to be a big difference in average relative error between LBS(5,1) and LBS(5,2), the latter's superiority is statistically significant with a p-value below 0.01.

Relative Error Distributions: As an explanation to the average relative error results shown in Figure 5, Figure 7 shows the error distributions for DBLP authors. For space reason, only the distribution of S-SEPIA and LBS(5,2) is shown. The downfall of S-SEPIA can be summarized by the high number of queries that are in the [-100%,-50%] range (i.e., under-estimation) and those in the [100%,∞) bucket (i.e., over-estimation). In contrast, LBS(5,2), perform significantly better in those situations.

Small and Negative Query Sets: The table below summarizes the average absolute errors on the small and negative query sets. There are 4 and 8 queries whose true frequencies are less than or equal to 20 or 50 respectively.

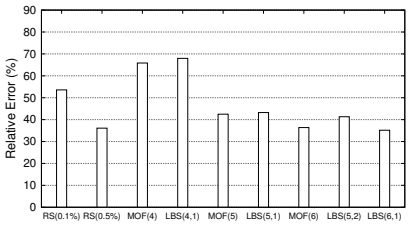
	$f_{true} \leq 20$	$f_{true} \leq 50$	Negative
RS(0.1%)	15	22	0
RS(0.5%)	15	44	0
S-SEPIA	26	45	16
LBS(4,1)	13	13	3
LBS(5,1)	13	13	1
MOF(6)	10	17	0
LBS(5,2)	12	15	1
LBS(6,1)	13	13	0

For the small query sets, MOF and LBS are superior to RS and S-SEPIA. For the negative query set, we can observe clear benefit from increasing N. When $N = 6$, both MOF and LBS exactly estimate the frequency of negative queries as zero. For other settings of LBS, even if the estimated frequencies are not exactly zero (e.g., 1 or 3), the frequencies are sufficiently accurate that it is likely that the predicate will be chosen as the most selective condition. In contrast, for S-SEPIA, because the estimated frequency is inaccurate (e.g., 16), there is a higher chance that incorrectly, another predicate will be selected as the most selective condition, resulting in a more expensive query plan.

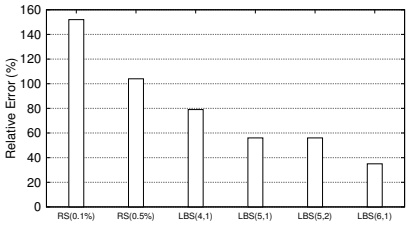
6.3 Query Sets on DBLP Titles

Figure 6 shows the average relative error, the space overhead and CPU query time in log scale for the short query set on the DBLP titles data set. In the context of the earlier discussion on Figure 5, the key highlights for the DBLP title data set are as follow. First, RS(0.5%) gives better average relative error in Figure 6(a) than in Figure 5(a). However, the conclusion remains the same in that MOF and LBS are almost two orders of magnitude faster. Second, S-SEPIA is still dominated by MOF and LBS in both estimation accuracy and space overhead. Finally MOF(5) is dominated by MOF(6), LBS(5,1), LBS(5,2) and LBS(6,1)

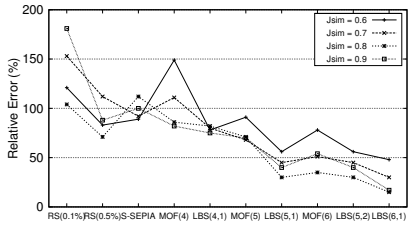
The results reported so far are based on the short query set with $\Delta \leq 3$. Figure 8 shows the average relative error on the long query set on DBLP Titles with $3 \leq \Delta \leq 5$. The results on S-SEPIA are not included as it takes too long to build even with a very low sampling ratio. In long queries with higher edit thresholds, it is not practical for MOF and LBS to enumerate all possible base substrings. One way to cap the computational cost is to generate only up to a specific number of base substrings. In other words, we set a limit for $|B_c|$ in line (6) of Algorithm 1 by sampling. We



(a) Average Relative Error



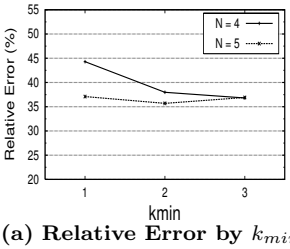
(a) SQL LIKE



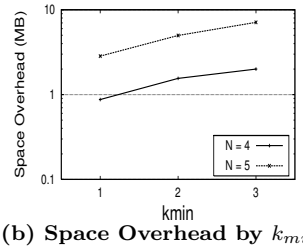
(b) Jaccard Similarity

Figure 8: Long Query Set On DBLP Titles

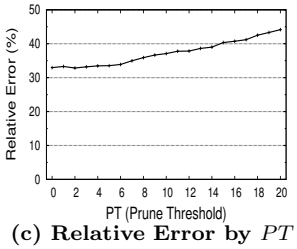
Figure 9: Other Similarity Measures



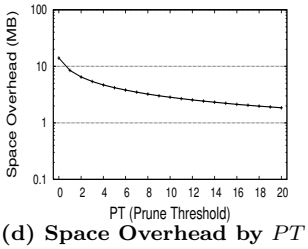
(a) Relative Error by k_{min}



(b) Space Overhead by k_{min}



(c) Relative Error by PT



(d) Space Overhead by PT

Figure 10: Impact of k_{min} and PT on IMDB Keywords

randomly generate up to 200 base substrings, and the results are shown in Figure 8.

In terms of the average relative error comparisons amongst RS, MOF and LBS, all the previous observations on the short query sets remain valid here. As for runtime, for MOF and LBS, the query time increases from an average 20 milliseconds for the short query set to an average 120 milliseconds for the long query set. The RS method scales up poorly with respect to Δ as its query time is well over 2000 milliseconds. In sum, both MOF and LBS are capable of handling long queries and larger Δ thresholds.

The space overhead graph is not included here. With a larger Δ, additional q-grams are needed to handle the larger number of wildcards (e.g., q-grams with 4 or 5 wildcards for our long query set). Compared with the space overhead shown in Figure 6(b), the additional space required turns out to be rather minimal (i.e., between 0.1MB to 0.5MB).

6.4 Other Similarity Measures

For LIKE predicate, we consider two types of conditions, $\%s\%$ and $\%s_1\%s_2\%$ as is typical in TPC-H benchmark. We randomly select one or two words of length between 5 and 12 and introduced 0 to 2 '_' in each word. Figure 9(a) gives the average relative errors on LIKE predicate. MOF and S-SEPIA are not present because MOF does not have signature which is necessary for Equation (11) and S-SEPIA does not support LIKE predicate. We observe that LBS outperforms RS by a large margin. Even with 0.5% of sampling ration, RS's average relative error is greater than 100% while LBS(5,1) is 56%. The effect of inceasing N is also clear. In LBS(6,1), the average relative error is as low as 35%.

For Jaccard similarity, we randomly select a word for similarity threshold J_{sim} of 0.6, 0.7, 0.8 and 0.9. Figure 9(b) plots the average relative error on Jaccard similarity. As in edit distance or LIKE cases, LBS consistently outperforms S-SEPIA, MOF and RS.

6.5 Impact of Parameters: k_{min}, PT

In Figure 5(a), there is the comparison between LBS(5,1) and LBS(5,2) using the DBLP Authors data set. This suggests that as k_{min}, the number of minima kept, increases from 1 to 2, there is a reduction in average relative error by using additional space. Figure 10(a) and (b) analyze the impact of k_{min} in greater details. Specifically, the IMDB Keywords data set is used with $N = 4$ or 5, and $PT = 10$, k_{min} is varied from 1 to 3. When an extended 4-gram table is used (i.e., $N = 4$), keeping the second minimum reduces the error by 14% relative to the first minimum; keeping the second and the third minima reduces the average relative error by 20%. However, this reduction is less in the 5-gram case. In general, when N is small, having $k_{min} = 2$ helps. According to Figure 10(b), it is important to note that the additional space needed by incrementing k_{min} is less than the extra space required by incrementing N. For instance, the space consumption of LBS(4,2) is less than the space consumption of LBS(5,1).

Figure 10(c) and (d) show the impact of changing the pruning threshold PT based on the keywords data set and LBS(5,1). In Figure 10(d), we note a sharp drop in the space overhead, especially in the low prune threshold. This is not so surprising considering that many real world text data follow Zipf's law. The dropping ratio is bigger at higher N. Figure 10(c) shows that LBS experiences only small increase in error by increasing the prune threshold. This is because the true frequency is most likely affected by a small number of base substrings whose frequency is relatively high.

6.6 Impact of Data set Size

Figure 11 compares the average relative errors of N-gram tables for LBS(4,1), LBS(5,1), and LBS(6,1) varying the data set size. The $Title300$ data set is the full Titles data set used throughout the experiments and we randomly select 50,000 and 100,000 titles for the $Title50$ and $Title100$ data

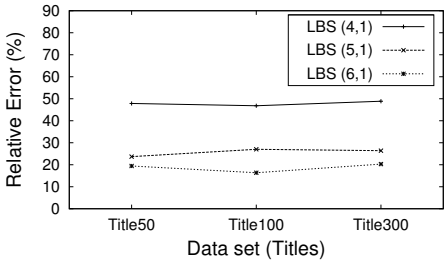


Figure 11: Impact of Data Set Size on Error

set respectively. We can observe that average relative error does not change much as the data set increases in size.

6.7 Recipe: Balancing Space vs. Accuracy

MOF and LBS provide a very tunable setting depending on the space available and CPU time expectations. The table below offers a "recipe" for choosing the method and the parameters.

Available Space	Suggested Method
very low ($< 5\%$)	MOF(4)
low ($< 30\%$)	MOF(5)
medium ($< 70\%$)	LBS(5,1)
abundant ($< 120\%$)	LBS(5,2)
non-issue ($< 400\%$)	LBS(6,1)

When we have very limited amount of space, say $1\% \sim 5\%$ of the original data size, MOF(4) is the recommended choice. If CPU time is not a serious concern, MOF(5) is a good choice. If accuracy is a primary concern, the LBS estimation algorithm is recommended, particularly LBS(5,1) and LBS(5,2). Keeping signatures help to reduce relative error. Increasing k_{min} involves less space than increasing N. Finally, if space is not an issue, then LBS(6,1), and even LBS(7,1), is recommended.

7. CONCLUSIONS

In this paper, we developed algorithms for estimating selectivity of approximate substring matching with edit distance. Two algorithms based on the extended N-gram table, MOF and LBS, show accurate and fast estimation. MOF provides simple yet accurate estimation, and LBS improves MOF capturing more complex correlation among strings by adapting from set hashing signatures. We extended the proposed algorithms to SQL LIKE predicate and Jaccard similarity. As ongoing work, we explore further utilization of the stored signature information in LBS. For instance, we can support boolean queries [8].

8. REFERENCES

[1] A. Arasu, V. Ganti, R. Kaushik. Efficient Exact Set-Similarity Joins. *Proc. VLDB*, pp. 918–929, 2006.

[2] R. Baeza-Yates and B. A. Ribeiro-Neto. Modern Information Retrieval. *ACM Press/Addison-Wesley*, 1999.

[3] Andrei Z. Broder, On the resemblance and containment of documents, *Proc. Compression and Complexity of SEQUENCES*, 1997.

[4] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher, Min-Wise Independent Permutations, *Journal of Computer and System Sciences*, 60(3):630 - 659, 2000.

[5] S. Chaudhuri, V. Ganti and L. Gravano. Selectivity Estimation for String Predicates: Overcoming the Underestimation Problem. *Proc. ICDE*, pp. 227- 238, 2004.

[6] A. Mazeika, M. H. Bohlen, N. Koudas and D. Srivastava. Estimating the Selectivity of Approximate String Queries *TODS*, pp. 227- 238, 2007.

[7] S. Chaudhuri et al. Robust and Efficient Fuzzy Match for Online Data Cleaning. *Proc. SIGMOD*, pp. 313-324, 2003.

[8] Zhiyuan Chen, Flip Korn, Nick Koudas and S. Muithukrishnan, Selectivity Estimation For Boolean Queries, *Proc. PODS*, pp. 216-225, 2000.

[9] E. Cohen, Size-estimation framework with applications to transitive closure and reachability, *Journal of Computer and System Sciences*, 55(3):441 - 453, 1997.

[10] W. Cohen, P. Ravikmar and and S. Fienberg. A Comparison of String Metrics for Matching Names and Records. *Proc. of KDD Workshop on Data Cleaning*, pp. 73-78, 2003.

[11] Flamingo Project, http://www.ics.uci.edu/ flamingo/.

[12] D. Gusfield. Algorithms on Strings, Trees and Sequences. *Cambridge Univ. Press*, 1997.

[13] H. V. Jagadish, Raymond. T. Ng and Divesh Srivastava. Substring Selectivity Estimation, *Proc. PODS*, pp. 249-260, 1999.

[14] L. Jin et al. Indexing Mixed Types for Approximate Retrieval, *Proc. VLDB*, pp. 793-804, 2005.

[15] L. Jin and C. Li. Selectivity Estimation for Fuzzy String Predicates in Large Data Sets, *Proc. VLDB*, pp. 397-408, 2005.

[16] N. Koudas, A. Marathe and D. Srivastava. Flexible String Matching Against Large Databases in Practice, *Proc. VLDB*, pp. 1078-1086, 2004

[17] P. Krishnan, J. S. Vitter and B. Iyer. Estimating Alphanumeric Selectivity in the Presence of Wildcards. *Proc. SIGMOD*, pp. 282-293, 1996.

[18] Hongrae Lee, Raymond T. Ng, and Kyuseok Shim, Extending Q-Grams to Estimate Selectivity of String Matching with Edit Distance, *Proc. VLDB*, pp. 195-206, 2007.

[19] Chen Li, Bin Wang, and Xiaochun Yang. VGRAM: Improving Performance of Approximate Queries on String Collections Using VariableLength Grams *Proc. VLDB*, pp. 303-314, 2007.

[20] N. Mamoulis. Efficient Processing of Joins on Set-valued Attributes. *Proc. SIGMOD*, pp. 157–168, 2003.

[21] G. Navarro and M. Raffinot. Fast and flexible string matching by combining bit-parallelism and suffix automata. *Journal of Experimental Algorithmics*, 5(4), 2000.

[22] S. Sarawagi and A. Bhamidipaty. Interactive Deduplication Using Active Learning. *Proc. VLDB*, pp. 269-278, 2002.

[23] Winkler, W. E. The State of Record Linkage and Current Research Problems. Statistics of Income Division, U.S. Bureau of the Census, 1999.