Efficient Top-k Algorithms for Approximate Substring Matching

Younghoon Kim
Seoul National University
Seoul, Korea
yhkim@kdd.snu.ac.kr

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

ABSTRACT

There is a wide range of applications that require to query a large database of texts to search for similar strings or substrings. Traditional approximate substring matching requests a user to specify a similarity threshold. Without top-k approximate substring matching, users have to try repeatedly different maximum distance threshold values when the proper threshold is unknown in advance.

In our paper, we first propose the efficient algorithms for finding the top-k approximate substring matches with a given query string in a set of data strings. To reduce the number of expensive distance computations, the proposed algorithms utilize our novel filtering techniques which take advantages of q-grams and inverted q-gram indexes available. We conduct extensive experiments with real-life data sets. Our experimental results confirm the effectiveness and scalability of our proposed algorithms.

Categories and Subject Descriptors

H.2 [**Database Management**]: Systems—*query processing, textual databases*

Keywords

Top-k approximate substring matching; edit distance; inverted q-gram index

1. INTRODUCTION

There is a wide range of applications that require to query a large database of texts to search for similar strings or substrings. A list of possible applications includes substring matching or short snippet suggestions for web search [23], named entity recognition [27], bio-informatics for finding DNA subsequences [20], music data retrieval [18] and spell checking [26]. These applications generally require high performance for real-time query processing to find similar substrings from text databases. While these applications are not all new, as there is an increasing trend of applications to deal with vast amounts of data, retrieving similar strings becomes a more challenging problem today.

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, 10, 11, 19]. Among them, the edit distance is one of the most widely accepted distance measures for database applications where domain specific knowledge is not really available [10, 12, 20, 27].

Based on the edit distance, the approximate string matching problem is to find every string from a string database whose edit distance to the query string is not larger than a given maximum distance threshold. There are diverse applications requiring approximate string matching. For example, it is useful to check whether the names or addresses provided by users as input are correct by querying databases to search for the strings within a maximum distance threshold [29]. However, for databases with long strings, the containment search of a query string called *approximate substring matching* is more reasonable.

Id	String	Id	String
s_1	Jackson Pollock	s_4	Jacksomville
s_2	Jakob Pollack	s_5	Jakson Pollack
s_3	Jason Polock	s_6	Mackson Polock

Figure 1: An example of a string database D

Two different problem definitions have been introduced for approximate substring matching. In the first definition, the problem is to discover all substrings of each string in a database, whose edit distances to the query string are within the given maximum threshold [4, 17, 27]. Consider a query string 'Jackson' and a database shown in Figure 1. If the maximum edit distance threshold is 2, with 'Jackson Pollock' alone, the substrings of 'Jacks', 'Jackso', 'Jackson', 'ackson' and 'ckson' are all included in the query result.

Another popular definition of the approximate substring matching problem is to retrieve every string s whose *substring edit distance* to the query string σ is at most the given maximum threshold, where the substring edit distance between σ and s is the smallest edit distance among the ones between σ and all substrings in s [13, 25]. For example, let us reconsider the string database in Figure 1 and the query string 'Jackson'. If the maximum threshold is 2, the approximate substring matches are {'Jackson Pollock', 'Jason Polock', 'Jacksomville', 'Jakson Pollack', 'Mackson Polock'}.

All previous studies of approximate string or substring matching mentioned so far require each user to provide a maximum distance threshold. However, in many applications, it is very difficult for each user to know a proper threshold in advance since it depends on the application scenarios. An appealing alternative method is to compute the most similar k strings or substrings without the need to specify a maximum distance threshold. Without such top-k approximate matching, users have to try different distance threshold values, which may lead to empty results (if the threshold chosen is too low) or too many results with a long running time (if the threshold is too high). It also supports interactive applications, where users are presented with top-k most similar strings or substrings progressively.

In our paper, we propose the efficient algorithms for the top-k approximate substring matching problem which find the top-k strings whose substring edit distances to the query string are the smallest among all strings in a given database. Contrast to traditional approximate substring matching, top-k approximate substring matching (1) does not require each user to provide a maximum distance threshold, and (2) finds the top-k strings in data whose 'substring edit distances' to a given query string are the k smallest values. To the best of our knowledge, no existing work has addressed the top-k approximate substring matching problem and our algorithms presented in this paper are the first work for the problem.

EXAMPLE 1.1**:** *Consider the string data in Figure 1. Suppose that the query string σ is 'Jackson' and $k = 3$. Since s_1 has the substring which is exactly σ, its substring edit distance to σ is 0. For s_4 and s_5, the substring edit distances to σ are 1 since s_4 and s_5 have 'Jacksom' and 'Jakson' respectively. For other strings, the substring edit distances to σ are at least* 1. *Thus, the top-*3 *approximate substring matches are {'Jackson Pollock', 'Jacksomville', 'Jakson Pollack'}.* ∎

Given a set of strings D and a query string σ, a naive algorithm of top-k approximate substring matching examines every string s in D and compute the substring edit distance $d_{sub}(s, \sigma)$ one by one. We refer to the brute-force algorithm as *TopK-NAIVE*. Since computation of substring edit distance $d_{sub}(s, \sigma)$ is very expensive, we develop more efficient algorithms by utilizing q-grams in the query strings and available inverted q-gram indexes adopted widely to find approximate string matches [9]. This paper makes the following contributions:

- We first propose the algorithm *TopK-LB*, which reduces the number of substring edit distance computations in *TopK-NAIVE*, by checking each string s with a lower bound of $d_{sub}(s, \sigma)$ first based on dynamic programming.
- We next present the algorithm *TopK-SPLIT* which allows not to calculate the lower bounds of $d_{sub}(s, \sigma)$ for some strings by dividing data D into partitions based on q-grams and utilizing our proposed lower bound of $d_{sub}(s, \sigma)$ between σ and all strings in each partition.
- We also develop the algorithm *TopK-INDEX* that can speed up *TopK-SPLIT* by utilizing the posting lists of some q-grams in the query string, which can be obtained from the existing inverted positional q-gram index, whenever it is available by DBMSs.
- We conducted an extensive performance study of our proposed algorithms with real-life data sets. When there is no inverted q-gram index, we found that *TopK-SPLIT* is the best performer and faster than *TopK-NAIVE* by an order of magnitude. Whenever the inverted indexes are available, *TopK-INDEX* is the best and much faster than *TopK-SPLIT*. Experimental results also confirm that our algorithms outperform the extended traditional algorithms significantly and scale up well with large data sizes.

2. PRELIMINARIES

In this section, we provide definitions used in our paper and present a precise statement of the problem for the top-k approximate substring matching.

2.1 Notations and our Problem Definition

Let Σ be a finite alphabet of size $|\Sigma|$. For a string s of Σ^*, we denote the length of s with $|s|$. We use $s[i, j]$ with $1 \leq i \leq j \leq |s|$ to denote the substring of s which starts from position i and ends at position j. Similarly, $s[i]$ with $1 \leq i \leq |s|$ denotes the character at the i-th position of s.

Edit distance: For two strings s_i and s_j, s_i can be transformed to s_j by applying repeatedly the three operations: insertion, deletion and substitution. The *edit distance* between s_i and s_j, which is denoted by $d(s_i, s_j)$, is defined as the minimum number of operations needed to transform s_i to s_j (or s_j to s_i). The dynamic programming algorithm to compute the edit distance between two strings s_i and s_j takes $O(|\sigma| \cdot |s|)$ time [14].

Substring edit distance: The substring edit distance from a string s to another string σ, which is represented by $d_{sub}(s, \sigma)$, is the minimum among the edit distance between σ and every substring of s. The $O(|\sigma| \cdot |s|)$-time algorithm based on dynamic programming to compute the substring edit distance is presented in [24]. (For more details, read [24, 25].)

We next present the definition of our *top-k approximate substring matching* problem.

DEFINITION 2.1**:** (Top-k Approximate Substring Matching) *Given a set of strings D and a query string σ, the top-k approximate substring matching problem is to find the k approximate strings $TOP_k(\sigma) = \langle s_1, s_2, ..., s_k \rangle$ in D which is an ordered sequence of k strings satisfying the following conditions:*

- *$d_{sub}(s_1, \sigma) \leq d_{sub}(s_2, \sigma) \leq \cdots \leq d_{sub}(s_k, \sigma)$ holds.*
- *For every $s_j \in D$ such that $s_j \notin TOP_k(\sigma)$, we have $d_{sub}(s_k, \sigma) \leq d_{sub}(s_j, \sigma)$.*

EXAMPLE 2.2**:** *Consider the strings shown in Figure 1. Suppose that the query string σ is 'Jackson' and we are interested in the top-*2 *approximate substring matches $TOP_2(\sigma)$. Since s_1 includes σ as a substring, the substring edit distance $d_{sub}(s_1, \sigma)$ is 0. For the string s_4, $d_{sub}(s_4, \sigma)$ is 1 because s_4 has a substring 'Jacksom'. For other strings, the substring edit distances to σ are at least* 1. *Thus, we get $TOP_2(\sigma)$={s_1,s_4}.* ∎

2.2 The q-grams and Inverted q-gram Indexes

We define q-grams and positional q-grams as follows.

DEFINITION 2.3**:** *The* q-gram*s of a string s are $s[i, i + q - 1]$s with all $1 \leq i \leq (|s| - q + 1)$. The* positional q-grams *of a string s are all pairs of each q-gram in s and its position (i.e., ($s[i, i + q - 1]$, i) with $1 \leq i \leq (|s| - q + 1)$).*

EXAMPLE 2.4**:** *For the string s='Jackson' and q=3, ('Jac', 1), ('ack',2), ('cks',3), ('kso',4) and ('son',5) are the positional* 3*-grams of the string s.* ∎

Given a q-gram and a string containing the q-gram, a pair of its string id and the position where the q-gram occurs in the string is called a *posting*. With a set of strings D, the *posting list* of a q-gram is the list of postings for every occurrence of the q-gram in all strings in D. Note that every posting list is primarily sorted in increasing order of string

q-gram	posting list	q-gram	posting list	q-gram	posting list
Jac	(s_1,1), (s_4,1)	llo	(s_1,11)	ack	(s_2,11), (s_5,12)
ack	(s_1,2), (s_4,2), (s_6,2)	loc	(s_1,12), (s_3,10), (s_6,11)	Jas	(s_3,1)
cks	(s_1,3), (s_4,3), (s_6,3)	ock	(s_1,13), (s_3,11), (s_6,12)	aso	(s_3,2)
kso	(s_1,4), (s_4,4), (s_5,3),(s_6,4)	Jak	(s_2,1), (s_5,1)	omv	(s_4,6)
son	(s_1,5), (s_3,3), (s_4,5), (s_5,4), (s_6,5)	ako	(s_2,2)	mvi	(s_4,7)
on_	(s_1,6), (s_3,4), (s_5,5), (s_6,6)	kob	(s_2,3)	vil	(s_4,8)
n_P	(s_1,7), (s_3,5), (s_5,6), (s_6,7)	ob_	(s_2,4)	ill	(s_4,9)
_Po	(s_1,8), (s_2,6), (s_3,6), (s_5,7), (s_6,8)	b_P	(s_2,5)	lle	(s_4,10)
Pol	(s_1,9), (s_2,7), (s_3,7), (s_5,8), (s_6,9)	lla	(s_2,9), (s_5,10)	aks	(s_5,2)
oll	(s_1,10), (s_2,8), (s_5,9)	lac	(s_2,10), (s_5,11)	kso	(s_5,3)
olo	(s_3,8), (s_6,10)	Mac	(s_6,1)	som	(s_4,6)

Figure 2: An inverted index for the database D

ids and secondarily sorted in ascending order of positions. With an *inverted q-gram index* of D, we can access the posting lists with q-grams as keys. In Figure 2, we show the inverted 3-gram index of the strings in Figure 1.

3. THE LOWER BOUNDS OF SUBSTRING EDIT DISTANCES

In this section, to identify the strings in D which do not need to compute actual substring edit distances, we present how to compute the lower bounds of substring edit distances by utilizing q-gram properties. For correctness and efficiency, we develop our techniques to guarantee *no false dismissals* and *few false positives*, respectively. To achieve these goals, we devise several filtering methods.

3.1 A Lower Bound of $\mathbf{d_{sub}}(s, \sigma)$

We denote the lower bound of the edit distance $d(s, \sigma)$ between a string s and a query string σ by $\ell o(d(s, \sigma))$. The following lemma borrowed from [9] allows us to compute $\ell o(d(s, \sigma))$.

LEMMA 3.1.: *[9] Given a query string σ and a string s, if s has at most c common q-grams with σ, the edit distance between $d(s, \sigma)$ satisfies the following inequality.*

$$d(s, \sigma) \geq \lceil (\max(|\sigma|, |s|) - q + 1 - c)/q \rceil$$

In other words, $\ell o(d(s, \sigma)) = \lceil (\max(|\sigma|, |s|) - q + 1 - c)/q \rceil$.

We now present the lemma below which allows us to compute a lower bound of the substring edit distance $d_{sub}(s, \sigma)$ between a string s and a query string σ. Remember that $s[i, j]$ denotes the substring of s which starts from position i and ends at position j.

LEMMA 3.2: *Consider a query string σ and a string s. Let c be the number of common q-grams between σ and s. Furthermore, let c_i be the number of common q-grams between σ and $s[i, i+|\sigma|$-1]. Then, the lower bound of $d_{sub}(s, \sigma)$, represented by $\ell o(d_{sub}(s, \sigma))$, is*

$$\ell o(d_{sub}(s, \sigma)) =$$
$$\begin{cases} \lceil (|\sigma| - q + 1 - c)/q \rceil, & \text{if } |s| < |\sigma|, \\ \min_{1 \leq i \leq |s|-|\sigma|+1} \lceil (|\sigma| - q + 1 - c_i)/q \rceil, & \text{otherwise.} \end{cases} \quad (1)$$

Proof: If $|s| < |\sigma|$, σ cannot be a substring of s. Thus, the lower bound of $d_{sub}(s, \sigma)$ is the same as the lower bound of $d(s, \sigma)$ which is actually $\lceil (|\sigma|-q+1-c)/q \rceil$ by Lemma 3.1.

When $|s| \geq |\sigma|$, in order to compute the lower bound of $d_{sub}(s, \sigma)$, we can enumerate every substring s' of s and compute the lower bound of edit distance between σ and every substring s'.

Consider a substring $s[i, i+\ell-1]$ of length ℓ. If ℓ is larger than $|\sigma|$, the lower bound of $d(s[i,i+\ell-1], \sigma)$, denoted by $\ell o(d(s[i,i+\ell-1), \sigma))$, is at least $\ell o(d(s[i,i+|\sigma|-1], \sigma))$. It is because the lower bound derived by Lemma 3.1 monotonically grows with increasing the length of string s. Since computing a substring edit distance requires to find the substring with the minimum edit distance to σ, we can ignore the substrings of s whose lengths are larger than $|\sigma|$. Thus, $\ell o(d_{sub}(s, \sigma))$ is the minimum one among $\ell o(d(s[i,i+|\sigma|-1],$ $\sigma))$s with $1 \leq i \leq |s|-|\sigma|+1$. ∎

To compute $\ell o(d_{sub}(s, \sigma))$ based on the above Lemma, if we count the common q-grams between σ and every substring $s[i, i+|\sigma|-1]$ with $1 \leq i \leq |s|-|\sigma|+1$, it takes $O(|\sigma| \cdot |s|)$ time. We now present the algorithm called *DYN-LB* which computes a tighter lower bound $\ell o(d_{sub}(s, \sigma))$ based on dynamic programming by utilizing the positions of matching q-grams with $O(\ell^2)$ time, where ℓ represents the number of matching q-gram pairs between s and σ. Note that we generally have $\ell \ll \min(|s|, |\sigma|)$.

Applying dynamic programming: We use the following notations for a query string σ and a string s in our dynamic programming formulation.

- Let $X_\sigma = \langle (x_1, p_1), \ldots, (x_{|X_\sigma|}, p_{|X_\sigma|}) \rangle$ be the sequence of positional q-grams from σ, each q-gram of which appears in the string s, satisfying $p_i < p_{i+1}$ for i=1,...,$|X_\sigma|-1$. Similarly, let $Y_s = \langle (y_1, r_1), \ldots, (y_{|Y_s|}, r_{|Y_s|}) \rangle$ be the sequence of positional q-grams from s, each q-gram of which occurs in σ, satisfying $r_i < r_{i+1}$ for i=1,...,$|Y_s|-1$.
- For a positional q-gram pair $\langle (x_i, p_i), (y_j, r_j) \rangle$ with $(x_i, p_i) \in X_\sigma$, $(y_j, r_j) \in Y_s$ and $x_i = y_j$, let $m[i, j]$ represent the lower bound of all edit distances $d(\sigma[1, p_i+q-1], s')$ where s' is every substring of s ending at position r_j+q-1.
- For two positional q-gram pairs $\{(x_u, p_u), (x_i, p_i)\} \subseteq X_\sigma$ and $\{(y_v, r_v), (y_j, r_j)\} \subseteq Y_s$ satisfying $x_u = y_v$, $x_i = y_j$, $p_u < p_i$ and $r_v < r_j$, let $\delta(p_u, p_i, r_v, r_j)$ denote the lower bound of the edit distance $d(\sigma[p_u, p_i+q-1], s[r_v, r_j+q-1])$.

Computing $m[i, j]$: For a positional q-gram pair $\langle (x_i, p_i),$ $(y_j, r_j) \rangle$ such that $(x_i, p_i) \in X_\sigma$, $(y_j, r_j) \in Y_s$ and $x_i = y_j$, depending on whether x_i $(=y_j)$ is the only common q-gram between $\sigma[1, p_i+q-1]$ and every substring of s ending at position r_j+q-1 or not, we split into two cases.

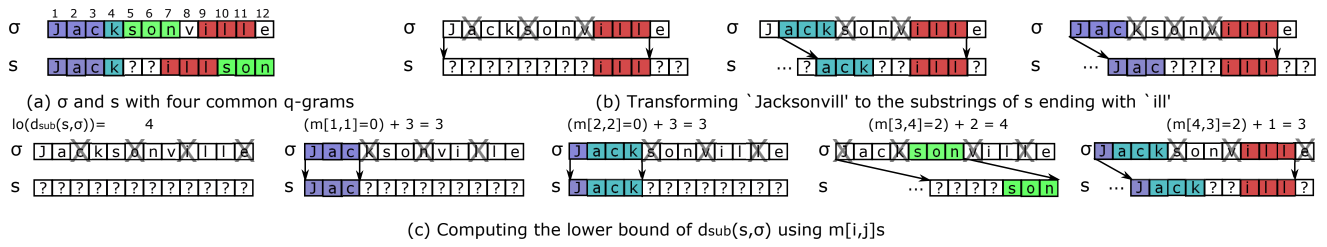
(1) When x_i $(=y_j)$ is the only common q-gram between $\sigma[1, p_i+q-1]$ and every substring of s ending at position r_j+q-1, the edit distance between them is at least $\lceil (p_i-1)/q \rceil$ due to to Lemma 3.2.

(2) Otherwise, suppose that $\langle (x_u, p_u), (y_v, r_v) \rangle$ is the last positional q-gram pair satisfying $p_u < p_i$, $r_v < r_j$ and $x_u = y_v$, which minimizes $m[i, j]$. Then, for every such a matching q-gram pair $\langle (x_u, p_u), (y_v, r_v) \rangle$, $m[i, j]$ (i.e., the lower bound of the edit distance between $\sigma[1, p_i+q-1]$ and every substring of s ending at position r_j+q-1) is the sum of $m[u, v]$ and $\delta(p_u, p_i, r_v, r_j)$ (i.e., the lower bound of the edit distance $d(\sigma[p_u, p_i+q-1], s[r_v, r_j+q-1])$). Since showing the optimal substructure of this problem is easy, we omit the proof.

Since we do not know the positional q-gram pair $\langle (x_u, p_u),$ $(y_v, r_v) \rangle$ satisfying $p_u < p_i$, $r_v < r_j$ and $x_u = y_v$ between $\sigma[1, p_i+$ $q-1]$ and every substring of s ending at position r_j+q-1, we consider every $\langle (x_u, p_u), (y_v, r_v) \rangle \in X \times Y$ as the last matching pair followed by $\langle (x_i, p_i), (y_j, r_j) \rangle$ to compute $m[i, j]$. Thus, the recursive solution to compute $m[i, j]$ is

$$m[i, j] = \min\{\lceil (p_i - 1)/q \rceil,$$
$$\min_{\substack{\forall \langle (x_u, p_u), (y_v, r_v) \rangle \in X \times Y: \\ (x_u = y_v) \wedge (p_u < p_i) \wedge (r_v < r_j)}} \{m[u, v] + \delta(p_u, p_i, r_v, r_j)\}\}. \quad (2)$$

For every pair $\langle (x_1, p_1), (y_j, r_j) \rangle$ with $x_1 = y_j$, there is only a single matching q-gram between $\sigma[1, p_1+q-1]$ and every substring of s ending at position r_j+q-1, and thus we set $m[1, j]$ to $\lceil (p_1-1)/q \rceil$ according to Lemma 3.2.



(a) σ and s with four common q-grams
(b) Transforming `Jacksonvill' to the substrings of s ending with `ill'
lo(dsub(s,σ))= 4
(m[1,1]=0) + 3 = 3
(m[2,2]=0) + 3 = 3
(m[3,4]=2) + 2 = 4
(m[4,3]=2) + 1 = 3
(c) Computing the lower bound of dsub(s,σ) using m[i,j]s

Figure 3: Computing the lower bound of $d_{sub}(s,\sigma)$ with DYN-LB

Computing $\ell o(d_{sub}(s,\sigma))$: Now we show how to compute $\ell o(d_{sub}(s,\sigma))$ using $m[i,j]$. When every q-gram in σ is modified, the lower bound of $d_{sub}(s,\sigma)$ is $\lceil(|\sigma|-q+1)/q\rceil$ due to Lemma 3.2. Otherwise, suppose that the last matching q-gram pair is $\langle(x_i,p_i),\ (y_j,r_j)\rangle$ with $(x_i,p_i)\in X_\sigma$, $(y_j,r_j)\in Y_s$ and $x_i{=}y_j$. The lower bound of $d_{sub}(s,\sigma)$ is the sum of $m[i,j]$ and the minimum number of edit operations to modify every remaining q-gram in $\sigma[p_i{+}1,|\sigma|]$ so that there is no matching q-gram between $\sigma[p_i{+}1,|\sigma|]$ and $s[r_j{+}1,|s|]$, which is $\lceil(|\sigma|{-}p_i{-}q{+}1)/q\rceil$ according to Lemma 3.2. Since we do not know the actual last matching positional q-gram pair, we consider every $\langle(x_i,p_i),(y_j,r_j)\rangle\in X\times Y$ such that $x_i{=}y_j$ as the last matching pair to compute $\ell o(d_{sub}(s,\sigma))$. Thus, the lower bound of $d_{sub}(s,\sigma)$ is

$$\min\{\lceil(|\sigma|-q+1)/q\rceil,\ \min_{1\le i\le|X_\sigma|,1\le j\le|Y_s|}\{m[i,j]+\lceil(|\sigma|-p_i-q+1)/q\rceil\}. \quad (3)$$

Computing $\delta(\mathbf{p_u},\mathbf{p_i},\mathbf{r_v},\mathbf{r_j})$: Given two positional q-gram pairs $\{(x_u,p_u),(x_i,p_i)\}\subseteq X_\sigma$ and $\{(y_v,r_v),(y_j,r_j)\}\subseteq Y_s$ satisfying $x_u{=}y_v$, $x_i{=}y_j$, $p_u{<}p_i$ and $r_v{<}r_j$, the transformation from $\sigma[p_u,p_i{+}q{-}1]$ to $s[r_v,r_j{+}q{-}1]$ by performing at least an edit operation on every q-gram in $\sigma[p_u,p_i{+}q{-}1]$ except the two q-grams (x_u,p_u) and (x_i,p_i) is the same as converting $\sigma[p_u{+}1,p_i{+}q{-}2]$ to $s[r_v{+}1,r_j{+}q{-}2]$ by modifying every q-gram in $\sigma[p_u{+}1,\ p_i{+}q{-}2]$ without exception.

The lower bound of the edit distance for the transformation performing at least an edit operation on every q-gram in $\sigma[p_u+1,p_i{+}q{-}2]$ is $\lceil(p_i{-}p_u{-}1)/q\rceil$ $(=\lceil(((p_i{+}q{-}2){-}(p_u{+}1)+1){-}q{+}1{-}0)/q\rceil)$ due to Lemma 3.2. Furthermore, the difference between the lengths of two strings is a definite lower bound of the edit distance [9]. Thus, $\delta(p_u,p_i,r_v,r_j)$ becomes $\max\{\lceil(p_i-p_u-1)/q\rceil,|(p_i-p_u)-(r_j-r_v)|\}$.

Time complexity: Let ℓ be the number of matching q-gram pairs $\langle(x_i,p_i),(y_j,r_j)\rangle\in X\times Y$ such that $x_i{=}y_j$. Since the number of entries $m[i,j]$ is ℓ and it takes $O(\ell)$ time to compute each $m[i,j]$ in Equation (2), the time complexity of *DYN-LB* is $O(\ell^2)$.

Example 3.3**:** *Suppose we have σ='Jacksonville' of length 12 and s='Jack Willson'. The matching positional q-gram sets in σ and s are X_σ={(Jac,1), (ack,2), (son,5), (ill,9)} and Y_s={(Jac,1), (ack,2), (ill,7), (son,10)} respectively as illustrated in Figure 3(a). With the first matching q-gram 'Jac' in both X_σ and Y_s, m[1,1] becomes 0. With the second matching positional q-gram pair (x_2,p_2)=(ack,2) in X_σ and (y_2,p_2)=(ack,2) in Y_s, m[2,2] becomes 0. With the next matching q-grams (x_4,p_4)=(ill,9) in X_σ and (y_3,p_3)=(ill,7) in Y_s, we consider the following transformations, as shown in Figure 3(b):*

- *The transformation that modifies every q-gram except the last q-gram (ill,9) in σ, which takes at least 3 edit operations (=$\lceil(9-1)/3\rceil$).*
- *Modifying $\sigma[1,4]$='Jack' to a substring of s ending with 'ack' followed by changing 'acksonvill' to 'ack Will', which requires at least 2 edit operations (=$m[2,2]$+max($\lceil(9{-}2{-}1)/3\rceil$, $|10{-}8|$)).*
- *Transforming $\sigma[1,3]$='Jac' to $s[1,3]$='Jac' and converting 'Jacksonvill' to 'Jack Will', which needs at least 3 edit operations (=$m[1,1]$+max($\lceil(9{-}1{-}1)/3\rceil$, $|11-9|$)).*

Now, m[4,3] becomes 2. Similarly, we obtain m[3,4]=2. Finally, to compute the lower bound of $d_{sub}(s,\sigma)$ by Equation (3), we examine the lower bounds by the five transformations in Figure 3(c), such that every matching q-gram in X_σ is modified or each q-gram in X is the right most q-gram which is not changed. Then, we obtain 3 as the lower bound of $d_{sub}(s,\sigma)$. ∎

3.2 A Lower Bound of Substring Edit Distances between a Query and a Set of Strings

Previously, we could compute the lower bound of $d_{sub}(s,\sigma)$ using the common q-grams between a query string σ and the string s. However, we next show how to compute the lower bound of $d_{sub}(s,\sigma)$ using the mismatching q-grams from σ to s (i.e., the q-grams occurring in σ but not in s).

Lemma 3.4**:** *Consider a query string σ and a string s. If the number of mismatching q-grams from σ to s is at least m, we have $d_{sub}(s,\sigma)\ge\lceil m/q\rceil$.*

Proof: When a string s_2 is obtained by applying a single edit operation on a string s_1, we have at most q mismatching q-grams from s_1 to s_2. Since at least m q-grams in σ do not appear in s, we need at least $\lceil m/q\rceil$ edit operations to convert σ to s (i.e., $\ell o(d(s,\sigma))=\lceil m/q\rceil$). Furthermore, for every substring s' of the string s, since the number of mismatching q-grams from σ to s' is at least that of mismatching q-grams from σ to s, we can claim that the number of mismatching q-grams from σ to s' is at least m. Thus, we have $d(s',\sigma)\ge\lceil m/q\rceil$ for every substring s' of s and we can conclude $d_{sub}(s,\sigma)\ge\lceil m/q\rceil$. ∎

Consider a subset $G'\subseteq G$ where G is the set of all possible q-grams from σ. Assume that we partition D into $D^+_{G'}$ and $D^-_{G'}$ so that $D^+_{G'}$ is the set of strings in D which share at least a q-gram in G' and $D^-_{G'}$ is the rest of strings in D (i.e., $D^-_{G'}=D{-}D^+_{G'}$). Let $d_{sub}(D,\sigma)$ represent the smallest substring edit distance between σ and every string s in D. We denote the lower bound of $d_{sub}(D,\sigma)$ by $\ell o(d_{sub}(D,\sigma))$.

The lower bounds of $d_{sub}(D^-_{G'},\sigma)$: We can compute a lower bound of the substring edit distances between σ and every string in $D^-_{G'}$ by the following lemma.

Lemma 3.5**:** *Assume that we have a set of strings D and a query string σ. For each subset $G'\subseteq G$ where G is the q-gram set of σ, we have $d_{sub}(D^-_{G'},\sigma)\ge\lceil|G'|/q\rceil$.*

Proof: Due to Lemma 3.4, for every string s in $D^-_{G'}$ which does not include any q-gram in G', $d_{sub}(s, \sigma) \geq \lceil |G'|/q \rceil$ holds. Thus, we have $d_{sub}(D^-_{G'}, \sigma) \geq \lceil |G'|/q \rceil$. ∎

While the lower bound of obtained by Lemma 3.5 is effective, it does not consider the positions of the q-grams in σ. In other words, we assumed that every q-gram in G' may overlap with another q-gram in G' in at least a position of σ. However, if we choose the q-grams for G' such that every q-gram in G' does not overlap with another q-gram in G', we can obtain a tighter lower bound of $\ell o(d_{sub}(D^-_{G'}, \sigma))$.

LEMMA 3.6.: *Let G be the set of all q-grams in a query string σ. If we select a subset $G' \subseteq G$ in which every q-gram does not overlap with another q-gram in G', we have $d_{sub}(D^-_{G'}, \sigma) \geq |G'|$.*

Proof: For each string s in $D^-_{G'}$, suppose that we perform edit operations on σ to convert it to a substring of s. Since every q-gram in G' does not appear in s, every q-gram in G' should be modified by at least an edit operation. However, every q-gram in G' does not overlap with another q-gram in G' and thus a single edit operation cannot affect multiple q-grams. Thus, for every string s in $D^-_{G'}$, we need at least $|G'|$ edit operations to transform σ to any substring of s. ∎

EXAMPLE 3.7: *Suppose we have a query string σ=‘Jackson’ and G'={‘Jac’,‘kso’} in which every 3-gram does not overlap with another q-gram in G'. The strings in $D^-_{G'}$ are the strings which do not have even one of the 3-grams in G'. Since each 3-gram in G' does not overlap with another one, to transform σ to every substring of each string in $D^-_{G'}$, we need at least 2 edit operations (an edit operation on each of ‘Jac’ and ‘kso’) in σ. Note that we have $d_{sub}(D^-_{G'}, \sigma) \geq 2$ by Lemma 3.6.* ∎

If we fix the size of $G' \subseteq G$, we obtain the tightest lower bound $\ell o(d_{sub}(D^-_{G'}, \sigma))$ when G' is selected without any pair of overlapping q-grams.

COROLLARY 3.8: *Consider a query string σ and let G be the set of q-grams in σ. We obtain the tightest lower bound $\ell o(d_{sub}(D^-_{G'}, \sigma))$ among every possible $G' \subseteq G$ when the size of G' is $\lfloor |\sigma|/q \rfloor$.*

Proof: This is because we cannot select a set of q-grams with a larger size than $\lfloor |\sigma|/q \rfloor$ in which every q-gram does not overlap with another q-gram in the set. ∎

We now introduce the following lemma to use Lemma 3.6 effectively for top-k approximate substring matching.

LEMMA 3.9.: *Consider a set of strings D, a query string σ, the q-gram set G of σ and a subset $G' \subseteq G$. Assume that we partitioned D into $D^+_{G'}$ and $D^-_{G'}$. While examining every string in $D^+_{G'}$, if there are at least k strings whose substring edit distances to σ are at most $\ell o(d_{sub}(D^-_{G'}, \sigma))$, we can find the top-$k$ approximate substring matches in D by examining the remaining strings in $D^+_{G'}$ only by ignoring the remaining strings in $D^-_{G'}$.*

Proof: While examining every string in $D^+_{G'}$ one by one, let s_k denote the string with the k-th smallest substring edit distance $d_{sub}(s_k, \sigma)$ so far. Since $d_{sub}(s_k, \sigma) \leq \ell o(d_{sub}(D^-_{G'}, \sigma))$, for every string $s' \in D^-_{G'}$, $d_{sub}(s', \sigma)$ cannot get smaller than $d_{sub}(s_k, \sigma)$ and we can not have any string of the top-k approximate substring matches in $D^-_{G'}$. ∎

```
Function TopK-LB ($D$, $k$, $\sigma$)
begin
1.  $H_{TopK}$ = an empty max-heap storing $\langle$dist, str$\rangle$;
2.  $G = \{\sigma[i,i+q-1] | 1 \leq i \leq |\sigma|-q+1\}$;
3.  for each string $s$ in $D$ do
4.      $R = \{(s[i,i+q-1], i) \mid \forall i$ s.t.$1 \leq i \leq |s|-q+1,\ s[i,i+q-1] \in G\}$;
5.      $LB$ = DYN-LB($R$, $|\sigma|$);
6.      if $|H_{TopK}| < k$ or $LB < H_{TopK}$.getMax().dist then
7.          $d = d_{sub}(s, \sigma)$;
8.          if $|H_{TopK}| < k$ then $H_{TopK}$.insert($\langle d,s \rangle$);
9.          else if $H_{TopK}$.getMax().dist $> d$ then
10.             $H_{TopK}$.insert($\langle d,s \rangle$);
11.             $H_{TopK}$.deleteMax();
12. end for
13. return $H_{TopK}$;
end
```

Figure 4: The TopK-LB algorithm

4. TOP-K ALGORITHMS FOR APPROXIMATE SUBSTRING MATCHING

We first introduce the brute-force algorithm *TopK-NAIVE* which blindly examines every string s in a set of strings D and computes substring edit distances $d_{sub}(s, \sigma)$ one by one. Since distance computations are very expensive, to reduce the number of substring edit distances to compute, we propose the algorithm *TopK-LB* which computes $d_{sub}(s, \sigma)$ only when the lower bound $\ell o(d_{sub}(s, \sigma))$ is smaller than the current k-th smallest substring edit distance. The lower bound $\ell o(d_{sub}(s, \sigma))$ is computed by our novel dynamic programming algorithm in *TopK-LB*.

Since *TopK-LB* utilizes the algorithm based on dynamic programming to compute the lower bound $\ell o(d_{sub}(s, \sigma))$ with *every* string s in D, we next propose the algorithm *TopK-SPLIT* which can even skip the expensive computation of $\ell o(d_{sub}(s, \sigma))$ for some strings s in D. *TopK-SPLIT* divides D into partitions for a given q-gram set G' and utilizes our new lower bound of $d_{sub}(s, \sigma)$. We also develop a novel dynamic programming algorithm to find the best q-gram set G' to be used at the beginning of *TopK-SPLIT*. Furthermore, whenever the k-th substring edit distance becomes smaller, *TopK-SPLIT* adaptively adjusts G' so that we can skip even more strings for computing the lower bounds of $d_{sub}(s, \sigma)$.

TopK-SPLIT still examines every string in D and thus its running time increases proportionally to the size of D. To speed up *TopK-SPLIT* further, we develop *TopK-INDEX* that does not need to check every string in D by utilizing the inverted q-gram indexes, whenever they are available.

We next present the above algorithms for finding the top-k approximate substring matches in more details.

TopK-NAIVE: The naive algorithm examines every string s in D and computes the substring edit distance $d_{sub}(s, \sigma)$. To find the top-k approximate substring matches in D, we maintain a max-heap H_{TopK} storing the k strings s' with the smallest $d_{sub}(s', \sigma)$s which are used as the keys in the max-heap H_{TopK}. If the size of H_{TopK} is less than k, we just insert the string s to H_{TopK}. Otherwise, we check whether $d_{sub}(s, \sigma)$ is smaller than $d_{sub}(s_R, \sigma)$ of the string s_R at the root of H_{TopK} (i.e., whether $d_{sub}(s, \sigma)$ is smaller than the k-th smallest substring edit distance so far). If it is satisfied, we delete the string at the root s_R of H_{TopK} and insert the string s to H_{TopK}. If not, we move to the next string and repeat the above step until we encounter the last string in D. We refer to the naive algorithm as *TopK-NAIVE*.

TopK-LB: Similar to *TopK-NAIVE*, we scan all strings in D. However, for each string s in D, we generate the list of common positional q-grams $R = \langle (g_1, p_1), \ldots (g_n, p_n) \rangle$ between σ and s in increasing order of positions where the position p_i of each (g_i, p_i) in R is the position in s at which the q-gram g_i appears. Then, we can compute the lower bound $\ell o(d_{sub}(s, \sigma))$ by invoking *DYN-LB* presented in Section 3.1. If $\ell o(d_{sub}(s, \sigma))$ is not smaller than the k-th smallest substring edit distance so far, we do not need to calculate the actual value of $d_{sub}(s, \sigma)$ since s cannot be a string of the top-k approximate substring matches. We call the algorithm *TopK-LB* and present its pseudocode in Figure 4.

EXAMPLE 4.1.: *Consider the set of strings D in Figure 1. Suppose the query string σ='Jacksen' of length 7 and we are interested in the top-2 approximate matches. The first two strings s_1 and s_2, whose substring edit distances to σ are 1 and 4 respectively, are inserted into the max-heap H_{TopK}. For the next string s_3='Jason Polock', the lower bound $\ell o(d_{sub}(s_3, \sigma))$ is 2 by* DYN-LB. *Since the second smallest substring edit distance in H_{TopK} is 4 and $\ell o(d_{sub}(s_3, \sigma))$ is smaller than 4, we have to compute $d_{sub}(s_3, \sigma)$, which turns out to be 3, and insert $(s_3, 3)$ into H_{TopK}. Now we have $\{(s_1, 1), (s_3, 3)\}$ in H_{TopK} and the second smallest substring edit distance so far becomes 3.*

With s_4='Jacksomville', $\ell o(d_{sub}(s_4, \sigma))$ becomes 1. Since $\ell o(d_{sub}(s_4, \sigma))$ is smaller than the second smallest substring edit distance in H_{TopK}, we have to compute $d_{sub}(s_4, \sigma)$ which is 2 and $(s_4, 2)$ is added to H_{TopK}. Then, the second smallest substring edit distance so far becomes 2. For s_5='Jakson Pollack', $\ell o(d_{sub}(s_5, \sigma))$ is 2 and thus we can skip computing expensive $d_{sub}(s_5, \sigma)$. Finally, with s_6='Mackson Polock', $\ell o(d_{sub}(s_5, \sigma))$ is 1 and we should compute $d_{sub}(s_5, \sigma)$. In summary, we calculated the substring edit distances with 5 strings out of 6 strings. ∎

TopK-SPLIT: Let G be the set of all q-grams in σ and let G' be a subset of G such that every q-gram in G' does not overlap with another q-gram in the set. We conceptually divide the strings in D into $D^+_{G'}$ and $D^-_{G'}$ where $D^+_{G'}$ is the set of strings in D each of which has at least a q-gram in G' and $D^-_{G'}$ is $(D - D^+_{G'})$. While scanning each string s, we can check easily whether the string s belongs to $D^+_{G'}$ or $D^-_{G'}$ by checking the q-grams in s.

Similar to *TopK-LB*, when examining every string s in D one by one, *TopK-SPLIT* generates the list of common positional q-grams R and skips computing $d_{sub}(s, \sigma)$ if $\ell o(d_{sub}(s, \sigma))$ is at least the k-th smallest substring edit distance so far. However, unlike *TopK-LB*, *TopK-SPLIT* can even skip computing $\ell o(d_{sub}(s, \sigma))$ by using Lemma 3.9 for each string as follows.

Let s_R denote the string whose substring edit distance to σ (i.e., $d_{sub}(s_R, \sigma)$) is the k-th smallest value among the strings examined so far. Among the unseen strings, we can skip distance computations for the strings whose substring edit distances to σ are at least $d_{sub}(s_R, \sigma)$. Since we split D into $D^+_{G'}$ and $D^-_{G'}$, after $d_{sub}(s_R, \sigma)$ becomes at most $\ell o(d_{sub}(D^-_{G'}, \sigma))$, we can skip every unseen string s belonging to $D^-_{G'}$. In this case, we have

$$d_{sub}(s_R, \sigma) \le |G'| \le d_{sub}(D^-_{G'}, \sigma) \qquad (4)$$

since the lower bound of $d_{sub}(D^-_{G'}, \sigma)$ is $|G'|$ due to Lemma 3.6 regardless of the q-grams in G'. However, for the unseen strings $s \in D^+_{G'}$, we still have to compute $d_{sub}(s, \sigma)$.

Intuitively, the larger $|G'|$ is, the earlier the condition of $d_{sub}(s_R, \sigma) \le |G'|$ in Inequality (4) can be satisfied. Thus,

```
Function TopK-SPLIT (D, k, σ)
begin
1.  $H_{TopK}$ = an empty max-heap storing ⟨dist, str⟩;
2.  $G = \{\sigma[i,i+q-1] | 1 \le i \le |\sigma|-q+1\}$;
3.  $\tau = \lfloor |\sigma|/q \rfloor$;
4.  $cond = false$;
5.  for each string s in D do
6.    $R = \{(s[i,i+q-1], i) \mid \forall i \text{ s.t.} 1 \le i \le |s|-q+1,\ s[i,i+q-1] \in G\}$;
7.    if $cond$=$true$ and $\exists (g_i, p_i) \in R,\ g_i \in G'$ then $part$ = '+';
8.    else $part$ = '−';
9.    if $cond$=$false$ or $part$ = '+' then
10.       $LB$ = DYN-LB($R$, $|\sigma|$);
11.       if $|H_{TopK}| < k$ or $LB < H_{TopK}$.getMax().dist then
12.          $d = d_{sub}(s, \sigma)$;
13.          if $|H_{TopK}| < k$ then $H_{TopK}$.insert(⟨d,s⟩);
14.          else if $H_{TopK}$.getMax().dist > d then
15.             $H_{TopK}$.insert(⟨d,s⟩);
16.             $H_{TopK}$.deleteMax();
17.             if $H_{TopK}$.getMax().dist ≤ τ then
18.                $cond = true$;
19.                $\tau = H_{TopK}$.getMax().dist;
20.                $G'$ = BEST-$G'$(τ, G);
21.          if $cond$=$true$ and $H_{TopK}$.getMax().dist < τ then
22.             $\tau = H_{TopK}$.getMax().dist;
23.             $G'$ = BEST-$G'$(τ, G);
24. end for
25. return $H_{TopK}$;
end
```

Figure 5: The TopK-SPLIT algorithm

when $|G'|$ is large, we have a smaller number of strings in D to examine before $d_{sub}(s_R, \sigma) \le |G'|$ holds. However, once the inequality is satisfied, with a large $|G'|$, the number of the unseen strings belonging to $D^+_{G'}$ also becomes generally large.

To satisfy $d_{sub}(s_R, \sigma) \le |G'|$ as soon as possible, we initially set $|G'|$ with $\lfloor |\sigma|/q \rfloor$. Note that the lower bound of $d_{sub}(D^-_{G'}, \sigma)$ is $|G'|$ and cannot be larger than $\lfloor |\sigma|/q \rfloor$ due to Corollary 3.8. Furthermore, note that we need G' to determine $D^+_{G'}$ to prune unseen strings only when $d_{sub}(s_R, \sigma) \le |G'|$ is satisfied, since the inequality holds in the same location in D regardless of G' with the same size of G'. Thus, once $d_{sub}(s_R, \sigma) \le \lfloor |\sigma|/q \rfloor$ holds, while scanning strings in D, we compute G' of size $d_{sub}(s_R, \sigma)$ and next examine the unseen strings in $D^+_{G'}$ only. For a given size ρ of G', we will discuss how to select such a subset G' minimizing the number of strings in $D^+_{G'}$ in Section 5. Thus, for the time being, we will assume that such a G' is provided by the function *BEST-G'(ρ)*.

Let τ be the k-th substring edit distance when $d_{sub}(s_R, \sigma) \le \lfloor |\sigma|/q \rfloor$ is first satisfied. While we examine the strings in D, the k-the smallest substring edit distance monotonically decreases. When the k-the smallest substring edit distance is changed from τ to τ'', we can still test the unseen strings in $D^+_{G'}$ only. However, since any G'' satisfying $|G''| < |G'|$ and $d_{sub}(s_R, \sigma) = \tau'' \le |G''|$ allows us not to examine $D^-_{G''}$ and we have $|D^-_{G''}| \le |D^-_{G'}|$ generally, we can select such a G'' and check the unseen strings in $D^+_{G''}$ only. We can perform the above step whenever the k-th substring edit distance decreases.

The pseudocode of *TopK-SPLIT* is provided in Figure 5. The lower bound of $d_{sub}(D^-_{G'}, \sigma)$ (i.e., τ) is set to $\lfloor |\sigma|/q \rfloor$ initially which is the largest one according to Corollary 3.8. The variable *cond* has $false$ initially representing that the k-th smallest substring edit distance so far exceeds τ. Until we set *cond* as $true$, while scanning each string s in D, if $LB = \ell o(d_{sub}(s, \sigma))$ computed by *DYN-LB* is smaller than the k-th smallest substring edit distance so far, we compute the actual value of $d_{sub}(s, \sigma)$ similar to *TopK-LB*. However, once the k-th smallest substring edit distance (i.e.,

```
Function TopK-INDEX ($D$, $I$, $k$, $\sigma$)
begin
1.  $H_{TopK}$ = an empty max-heap storing $\langle$dist, str$\rangle$;
2.  $G = \{\sigma[i,i+q-1] | 1 \le i \le |\sigma|-q+1\}$;
3.  $\tau = |G'|$;
4.  $cond = false$;
5.  while $D$.end() = $false$ do
6.      $sid_D$ = $D$.getCurrentId();
7.      $sid_I$ = $I$.getFrontierId($G$);
8.      $s$ = null;
9.      if $cond = false$ and $sid_I > sid_D$ then
10.         $s$ = $D$.getString($sid_D$);
11.         $R = \{(s[i,i+q-1], i) \mid \forall i$ s.t.$1 \le i \le |s|-q+1, s[i,i+q-1] \in G\}$;
12.         if $\exists (g_i, p_i) \in R, g_i \in G'$ then $part$ = '+';
13.         else $part$ = '−';
14.     else
15.         ($R$, $part$) = $I$.getPosQgrams($sid_I$);
16.     if $cond = false$ or $part$ = '+' then
17.         $LB$ = DYN-LB($R$, $|\sigma|$);
18.         if $|H_{TopK}| < k$ or $LB < H_{TopK}$.getMax().dist then
19.             if $s$ = null then $s$ = $D$.getString($sid_I$);
20.             $d = d_{sub}(s, \sigma)$;
21.             if $|H_{TopK}| < k$ then $H_{TopK}$.insert($\langle d,s \rangle$);
22.             else if $H_{TopK}$.getMax().dist $> d$ then
23.                 $H_{TopK}$.insert($\langle d,s \rangle$);
24.                 $H_{TopK}$.deleteMax();
25.                 if $H_{TopK}$.getMax().dist$\le \tau$ then
26.                     $cond = true$;
27.                     $\tau = H_{TopK}$.getMax().dist;
28.                     $G'$ = BEST-$G'$($\tau$, $G$);
29.         if $cond$=$true$ and $H_{TopK}$.getMax().dist$<\tau$ then
30.             $\tau = H_{TopK}$.getMax().dist;
31.             $G'$ = BEST-$G'$($\tau$, $G$);
32. end while
33. return $H_{TopK}$;
end
```

Figure 6: The TopK-INDEX algorithm

H_{TopK}.getMax().dist) becomes at most τ, we set *cond* to *true* and select G' by calling *BEST-G'* in lines 17–21 so that we can skip the computation of $d_{sub}(s, \sigma)$ for the unseen strings $s \in D^-_{G'}$ by Lemma 3.9. Furthermore, whenever the k-th smallest substring edit distance decreases, we select G' with size of H_{TopK}.getMax().dist again in lines 23–26.

EXAMPLE 4.2**:** *Consider the strings D in Figure 1. Suppose we want to find the top-2 approximate substring matches the query string σ = 'Jacksen' using 3-grams. The lower bound of $d_{sub}(D^-_{G'}, \sigma)$ (i.e., τ) τ is initially set to 2(=$\lfloor 7/3 \rfloor$).*

First, the strings s_1 and s_2 are inserted into H_{TopK} whose substring edit distances to σ are 1 and 4 respectively. With s_3 and s_4, we compute $d_{sub}(s_3, \sigma)$(=3) and $d_{sub}(s_4, \sigma)$(=2), and then the second smallest substring edit distance in H_{TopK} becomes 2.

We now have 2 strings whose substring edit distances are at most τ. After selecting the q-grams set $G' \subseteq G$ by BEST-G'*, we can skip computing $d_{sub}(s, \sigma)$ with the strings $s \in D^-_{G'}$ due to Lemma 3.9. Suppose that the selected 3-gram set G' is {'Jac', 'kse'}. Then, for the strings s_5 and s_6, we can ignore them because they do not have any common 3-gram with G'. In summary, we could find the top-2 approximate substring matches by calculating the substring edit distances with 4 strings only.* ∎

TopK-INDEX: When the k-th smallest substring edit distance so far becomes at most $|G'|$ in *TopK-SPLIT*, we can even skip computing $\ell o(d_{sub}(s, \sigma))$ with the strings s $\in D^-_{G'}$. However, we still have to read unseen string s in D to determine whether s belongs to $D^-_{G'}$ by checking if s has at least a q-gram in G'. We will next improve the inefficiency by retrieving the unseen strings belonging to $D^+_{G'}$ only with utilizing inverted q-gram indexes when they are available.

While scanning each string in D, our new algorithm *TopK-INDEX* also scans the posting lists of the q-grams in G together. We assume that the strings in D are sorted with increasing order of string ids and we are also able to retrieve the strings in D with string ids as keys by random I/O access. We also assume that the posting list of every q-gram in G is stored with the increasing order of string id first and its position next in the inverted q-gram index. By reading the posting lists of the q-grams in G only, we can obtain the common q-gram list R for the strings which share at least a q-gram with σ. Whenever we need to compute the actual substring distance $d_{sub}(s, \sigma)$, we actually access the string s stored in D. Since the size of each string in D is generally much larger than that of the query string σ, using the posting lists of the q-grams in G only speeds up the generation of R and computation of $\ell o(d_{sub}(s, \sigma))$ significantly.

After the condition to apply Lemma 3.9 is satisfied (i.e., $d_{sub}(s_R, \sigma) \le \ell o(d_{sub}(D^-_{G'}, \sigma)))$, instead of sequentially reading the unseen strings in D, we access the remaining strings $s \in D^+_{G'}$ only by random I/O access with utilizing the inverted q-gram index to skip the remaining strings $s \in D^-_{G'}$. Similar to *TopK-SPLIT*, whenever $d_{sub}(s_R, \sigma)$ decreases, we select a smaller G' with $|G'| = d_{sub}(s_R, \sigma)$ to reduce the size of $D^-_{G'}$.

The pseudocode of *TopK-INDEX* is presented in Figure 6. The difference from *TopK-SPLIT* is the lines 6–16 in which the common positional q-gram list R is produced by reading the posting lists of q-grams in G sequentially in increasing order of string ids.

Note that every posting list in the inverted index is sorted by string ids and positions. To obtain the list R in increasing order of string ids, we initially set the frontier of each posting list of a q-gram in G as the location of the first posting in the posting list. Then, we read the smallest id among the postings in all frontiers and move the frontier to the next posting in its posting list. Let I denote the given inverted index. The call of *I.getFrontierId(G)* returns the smallest string id sid_I among the ids in all frontiers, and *I.getPosQgrams(sid_I)* returns the common positional q-gram list R between the string of sid_I and the query string σ by reading all postings (g_i, p_i) from the posting lists of the q-grams in G such that g_i=sid_I. The invocation of *I.getPosQgrams* also informs whether the string belongs to $D^+_{G'}$ or $D^-_{G'}$.

If we just follow the posting lists of the q-grams in G only to generate the list R, there may exist a string s in D which may not have any common q-gram with the query string σ and we may miss all of such strings. Thus, while we move a cursor on D together in the increasing order of string ids, if the smallest frontier id sid_I is larger than the id sid_D of D which the current cursor indicates, we read the string of sid_D from D, generate R from the actual string in D, and then perform the computations of lower bound and actual distance if necessary.

After *cond* becomes *true*, we always go to line 15 to read the posting lists to obtain the common q-gram list R without reading strings from D. The reason is that, after we find at least k strings whose substring edit distances are at most $\ell o(d_{sub}(D^-_{G'}, \sigma))$, if the query string σ and a string s in D do not share at least a q-gram, the string s belongs to $D^-_{G'}$ and thus we do not need to check such strings in D by Lemma 3.9. Furthermore, if the string s of sid_I shares at least a common q-gram with G' (i.e., *part* is '+'), we compute $\ell o(d_{sub}(s, \sigma))$ in line 17. If the lower bound

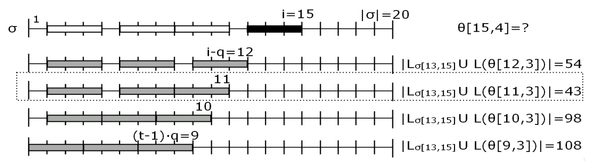

σ
1
i=15
|σ|=20
θ[15,4]=?
i-q=12
|Lσ[13,15] U L(θ[12,3])|=54
11
|Lσ[13,15] U L(θ[11,3])|=43
10
|Lσ[13,15] U L(θ[10,3])|=98
(t-1)·q=9
|Lσ[13,15] U L(θ[9,3])|=108

Figure 7: Computing $u[i,t]$ and $\theta[i,t]$

$\ell o(d_{sub}(s,\sigma))$ is smaller than the k-th smallest substring edit distance so far, we retrieve the string s from D by random I/O access to compute $d_{sub}(s,\sigma)$ as in line 20.

EXAMPLE 4.3*: Consider the strings D in Figure 1. Suppose that the query string σ is 'Jacksen' and we want to find the top-2 approximate substring matches from D using 3-grams. The 3-gram set G of σ is {'Jac', 'ack', 'cks', 'kse', 'sen'}. Initially, the lower bound τ of $d_{sub}(D^-_{G'},\sigma)$ is set to 2 ($=\lfloor 7/3 \rfloor$).*

At the beginning, the minimum string id among the frontiers of the posting lists of q-rams in G is s_1. The current cursor of D is also at s_1. Since we can obtain the common q-gram list R of s_1 from the posting lists, in each posting list, we read the frontier posting and move the frontier to the next. We also move the cursor of D to the next string without examining s_1. Since the current H_{TopK} is empty, we compute $d_{sub}(s_1,\sigma)$=1 and insert $(s_1, 1)$ into H_{TopK}.

Now, the minimum string id among the frontiers is s_4. However, since the current cursor of D indicates s_2, we read the s_2 and compute $d_{sub}(s_2,\sigma)$=4. Similarly, we read s_3 from D to calculate $d_{sub}(s_3,\sigma)$=3 and update H_{TopK}. Now the second smallest substring edit distance so far becomes 3.

We next read the frontier postings to obtain the common 3-gram list R between s_4 and σ which is {('Jac',1),('ack',2), ('cks',3)}. The current cursor of D moves to the next string s_5. After we compute $d_{sub}(s_4,\sigma)$=2, the second smallest substring edit distance so far becomes 2. From now, since we already have 2 strings whose substring edit distances are at most τ=2, with G' of size 2 obtained by calling BEST-G'*, we can find the top-2 approximate substring matches by considering the strings sharing at least a 3-gram in G' only due to Lemma 3.9. Let us assume that* BEST-G' *selects G'={'Jac', 'kse'}. Then, for the strings s_5 and s_6, we can ignore them because they do not appear in any posting list of 'Jac' or 'kse'. In summary, we can find the top-2 approximate substring matches by calculating the actual substring edit distances with 4 strings in D only.* ▮

5. SELECTING THE BEST G′ OF SIZE ρ

In the previous section, we assumed that $G' \subseteq G$ of size ρ is provided by the function *BEST-G'* for *TopK-SPLIT* and *TopK-INDEX*. Now, we present the algorithm *BEST-G'* which finds G' of size ρ such that the number of strings with at least a q-gram in G' in D (i.e., the size of $D^+_{G'}$) is the smallest. We use the following notations to describe our dynamic programming formulation to select such a G'.

- Let L_{g_i} be the set of the ids of strings containing the q-gram $g_i{\in}G$.
- For a q-gram set $Q \subseteq G$, let $L(Q) = \bigcup_{g_i \in Q} L_{g_i}$, which is the union of L_{g_i}s for all q-grams $g_i \in Q$, and we will use $|L(Q)|$ to represent the number of elements in $L(Q)$.
- Let $Q(i,t)$ be the set of all possible subsets Q of q-grams appearing in $\sigma[1,i]$ such that (1) $|Q|$=t, (2) the q-gram $\sigma[i{-}q{+}1,i]$ (i.e., the q-gram ending at the position i) always appears in Q and (3) all q-grams in Q does not overlap to each other.
- Let $\theta[i,t]$ denote the q-gram set Q in $Q(i,t)$ such that $|L(Q)|$ is the minimum among all q-gram sets in $Q(i,t)$.
- Let $u[i,t]$ represent $|L(\theta[i,t])|$.

Given the size ρ of G', we select Q with the minimum $|L(Q)|$ as G' among all subsets $Q \subseteq G$ consisting of ρ non-overlapping q-grams in σ. In every possible such Q, the ending position of the rightmost q-gram $\sigma[i{-}q{+}1,i]$ can be located in the positions i of σ with $\rho{\cdot}q{\le}i{\le}|\sigma|$. Thus, for every substring $\sigma[1,i]$ with $\rho{\cdot}q{\le}i{\le}|\sigma|$, we will enumerate the best ρ non-overlapping q-gram set Q which not only contains $\sigma[i{-}q{+}1,i]$ but also has the minimum $|L(Q)|$. The reason why we do not consider every i which is smaller than $(\rho{\cdot}q)$ is because the substring $\sigma[1,i]$ with $i < \rho \cdot q$ cannot have ρ non-overlapping q-grams. Since we use $\theta[i,\rho]$ to store the best Q with ρ non-overlapping q-grams which not only contains $\sigma[i{-}q{+}1,i]$ but also has the minimum $|L(Q)|$, we can find G' by selecting the $\theta[i,\rho]$ with the smallest $|L(\theta[i,\rho])|$ among $\theta[i,\rho]$s with $\rho{\cdot}q{\le}i{\le}|\sigma|$. In other words, if we compute $\theta[i,t]$s for every $1 \le t \le \lfloor|\sigma|/q\rfloor$ and $t \cdot q \le i \le |\sigma|$, we can select the q-gram set $\theta[i^*,\rho]$ as G' where i^* is chosen as follows:

$$i^* = \arg \min_{t \cdot q \le i \le |\sigma|} \{u[i,\rho]\} = \arg \min_{t \cdot q \le i \le |\sigma|} \{|L(\theta[i,\rho])|\}. \quad (5)$$

We next present a dynamic programming algorithm to compute $\theta[i,t]$ for $t = 1, \ldots, \lfloor|\sigma|/q\rfloor$ and $i = t \cdot q, \ldots, |\sigma|$.

Dynamic programming formulation: Since $\theta[i,t]$ contains t number of non-overlapping q-grams in the substring $\sigma[1,i]$ including $\sigma[i{-}q{+}1,i]$, we consider $\theta[j,t{-}1]{\cup}\{\sigma[i{-}q{+}1,i]\}$ with $1{\le}j{\le}(i-q)$ to compute $\theta[i,t]$. However, $\theta[j,t{-}1]$ does not exist for j=1,...,$(t{-}1){\cdot}q - 1$ because it is impossible to have $(t{-}1)$ non-overlapping q-grams in the substring $\sigma[1,j]$. Thus, we enumerate $\theta[j,t{-}1] \cup \{\sigma[i{-}q{+}1,i]\}$ for $(t{-}1) \cdot q{\le}j{\le}(i{-}q)$ only and select $\theta[j,t{-}1] \cup \{\sigma[i{-}q{+}1,i]\}$ with the smallest $|L(\theta[j,t{-}1]) \cup L_{\sigma[i-q+1,i]}|$ as $\theta[i,t]$. Our recursive definition for $\theta[i,t]$ is

$$\theta[i,t] = \theta[j^*, t-1] \cup \{\sigma[i-q+1,i]\} \quad (6)$$

where

$$j^* = \arg \min_{(t-1) \cdot q \le j \le i-q} \left\{\left|L_{\sigma[i-q+1,i]} \cup L(\theta[j,t-1])\right|\right\}. \quad (7)$$

In Figure 7, we show an example to illustrate how we compute $\theta[15,4]$. We enumerate $|L_{\sigma[13,15]} \cup \theta[j,3]|$ with $9{\le}j{\le}12$ and select $\{\sigma[13,15]\} \cup \theta[11,15]$ as $\theta[15,4]$ which minimizes $|L(\theta[15,4])|$.

If we compute $\theta[i,t]$ for every t from 1 to $\lfloor|\sigma|/q\rfloor$ and every i from $t{\cdot}q$ to $|\sigma|$ with the above dynamic programming algorithm, we can choose G' with a size ρ by Equation (5). We refer to this algorithm which finds G' with a given size ρ as *BEST-G'(ρ)*.

Note that *BEST-G'(ρ)* cannot find the optimal q-gram set G' because the optimal substructure property of our dynamic programming formulation is not satisfied. Suppose $\{\sigma[i{-}q{+}1,i]\} \cup Q$ is set to $\theta[i,t]$ in Equation (6) where Q is the optimal set with $(t-1)$ non-overlapping q-grams for $\sigma[1,j^*]$. Let Q' be a non-optimal q-gram set with $(t{-}1)$ non-overlapping q-grams in $\sigma[1,j^*]$ including the q-gram ending at the position j^*. Even though Q' is not an optimal q-gram set for $\sigma[1,j^*]$, if $L_{\sigma[i-q+1,i]}$ and $L(Q')$ share many common string ids so that $|L_{\sigma[i-q+1,i]} \cup L(Q')|$ becomes

g_1 \ g_2	-	Jac	ack	cks
Jac	100	-	-	-
ack	32	-	-	-
cks	16	-	-	-
kso	10	110	-	-
son	120	130	125	-
onv	40	120	43	50

(a) $|L_{g_1} \cup L_{g_2}|$ (=$|L(\{g_1,g_2\})|$)

	t=1		t=2	
i	θ[i,1]	u[i,1]	θ[i,2]	u[i,2]
3	{Jac}	100	-	-
4	{ack}	32	-	-
5	{cks}	16	-	-
6	{kso}	10	{Jac,kso}	110
7	{son}	120	{ack,son}	125
8	{onv}	40	{ack,onv}	43

(b) θ[i,t] and u[i,t]

Figure 8: Computations in *BEST*-$G'(\rho)$

smaller than $|L_{\sigma[i-q+1,i]} \cup L(Q)|$, we have $|L(Q')| > |L(Q)|$ and thus the computed $\theta[i,t]$ is not an optimal q-gram set for $\sigma[1,i]$. Thus, *BEST-$G'(\rho)$* finds an approximate q-gram set for G'. However, our performance study confirms that *BEST-$G'(\rho)$* obtains a good G' close to the optimal sets.

When *BEST-$G'(\rho)$* is invoked in *TopK-SPLIT* or *TopK-INDEX*, we need to know the actual size of $L(\theta[i,t])$ which is the union of the sets of string ids whose strings contain a q-gram in $\theta[i,t]$. We will use the MinHash technique [5, 7] to estimate the sizes of unions of sets in *BEST-$G'(\rho)$*.

Assume that for every q-gram g_i appearing in D, there is a MinHash signature of the set of string ids in which the q-gram g_i occurs. In *BEST-$G'(\rho)$*, we maintain the MinHash signature of $L(\theta[i,t])$ to compute $L_{\sigma[i-q+1,i]} \cup L(\theta[j,t-1])$ in Equation (7) with constant time.

Time complexity: In *BEST-$G'(\rho)$*, we have to compute $\theta[i,t]$ and $u[i,t]$ for every $1 \leq t \leq \lfloor|\sigma|/q\rfloor$ and every $t \cdot q \leq i \leq |\sigma|$. To compute each $\theta[i,t]$, it takes $O(|\sigma|)$ time. For each $u[i,t]$, it takes constant time since $u[i,t]$ is simply $|L(\theta[i,t])|$. Thus, the time complexity of *BEST-$G'(\rho)$* is $O(|\sigma|^3/q)$.

EXAMPLE 5.1.: *Consider the query string σ='Jacksonv' with $|\sigma| = 8$. Suppose that we should select the non-overlapping 3-gram set G' with size 2. For all possible non-overlapping 3-gram sets Q whose sizes are 1 or 2, $|L(Q)|$s are shown in Figure 8(a). We show all $\theta[i,t]$s and $u[i,t]$s computed by* BEST-$G'(\rho)$ *in Figure 8(b).*

Let us assume that we want to compute $\theta[8,2]$ that always includes the 3-gram $\sigma[6,8]$='onv'. We enumerate the following three cases of $|L_{onv} \cup L_{cks}|$(=50), $|L_{onv} \cup L_{ack}|$(=43) and $|L_{onv} \cup L_{Jac}|$(=120) by Equation (7), and select $\theta[8,2]$ = {'ack','onv'} with $u[8,2]$ = 43. Finally, to select the best G' with size 2 by Equation (5), we choose $\theta[8,2]$ = {'ack', 'onv'} for G' since $u[8,2]$ is the minimum among $u[6,2]$, $u[7,2]$ and $u[8,2]$. ∎

6. EXPERIMENTS

We empirically compared the performance of our proposed algorithms. All experiments reported in this section were performed on the machines with Intel Core^2 Duo 2.66GHz of processor and 2GB of main memory running Linux operating systems. All algorithms were implemented using C++ and compiled with GCC Compiler of version 4.1.3.

6.1 Implemented Algorithms

We implemented the following algorithms for our performance study.

- **TopK-NAIVE:** This represents the brute-force algorithm which computes the substring edit distance with every string in D to find the top-k approximate substring matches.
- **TopK-LB:** It is the implementation of *TopK-LB* which computes $d_{sub}(s,\sigma)$ only for the strings s in D whose lower bound $\ell o(d_{sub}(s,\sigma))$, obtained by calling *DYN-LB*, is smaller than the k-th smallest substring edit distance found so far.
- **TopK-SPLIT:** This is the algorithm which improves *TopK-LB* further by skipping even the computation of $\ell o(d_{sub}(s,\sigma))$ using the lower bound of substring edit distance between a query and a set of strings in Lemma 3.9.
- **TopK-INDEX:** This is the implementation of *TopK-INDEX* that utilizes the inverted q-gram indexes available to speed up *TopK-SPLIT*.
- **TopK-NGPP:** This is the modified version of *NGPP* in [27] to obtain the top-k approximate substring matches. The algorithm *NGPP* is the state-of-the-art algorithm to find all substrings of each string in D whose edit distances to a query string are at most a given maximum threshold τ. To adapt *NGPP* to top-k approximate substring matching, we first read the first k strings in D and set the k-th smallest one among their substring edit distances as the initial threshold τ. Then, as examining every string in D, if the k-th smallest substring edit distance becomes smaller than τ, we update the threshold τ to the k-th smallest substring edit distance.
- **TopK-FSS:** Similar to *TopK-NGPP*, we also extended *FSS* in [4] to compute the top-k approximate substring matches in the same manner.

Note that we used our own buffer management for reading the data strings in disk and accessing inverted indexes in all of our implementations without utilizing OS buffers to see the buffering effects for the tested algorithms more carefully. We report the native execution times (i.e., wall clock times) of the tested algorithms in this section.

6.2 Data Sets

To study the performance our proposed algorithms, we utilize the following two real-life data sets.

DBLP: For a short string data set, we used DBLP titles collected from `http://dblp.uni-trier.de/xml/`. However, since the original data is small, we increased its size by duplicating the original data 5 times. While we duplicate each string, we randomly performed an edit operation such as insert, delete and substitute on each position of the string with the probability of 0.1. The size of generated data is 635 MB with 13,966,030 strings whose average size is 48 bytes.

Wikipedia: This is the data set consisting of 106,185 web pages obtained from `http://en.wikipedia.org/wiki/Wikipedia:Database_download` for a long string data set. The data size is 1.1 GB and the average size is 11,027 bytes.

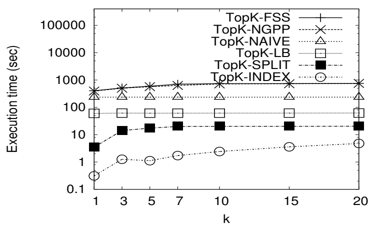
6.3 Queries Used

For DBLP, we randomly selected between 1 and 4 adjacent words appearing in DBLP titles to generate test queries. The range of query lengths is from 6 to 25 with average length 13.2. For Wikipedia, we sampled 50 entities from CoNLL data, which is a name entity data collected for Name Entity Recognition available for download at `http://www.cnts.ua.ac.be/conll2003/ner/`, as test queries. The lengths of selected queries are from 5 to 27 with average length 12.3.

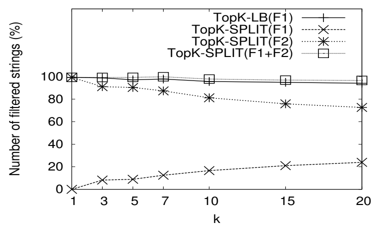
Similarly, for our experiments with varying lengths of query strings, we also selected the query strings of lengths from 5 to 25 with DBLP titles and CoNLL data respectively. For each query length, 50 query strings were sampled.

6.4 Performance Results

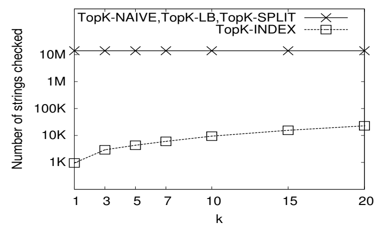
We measured the performance of the algorithms for both DBLP and Wikipedia with varying k, the number of strings



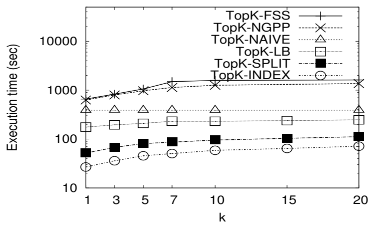
(a) Execution times (DBLP)



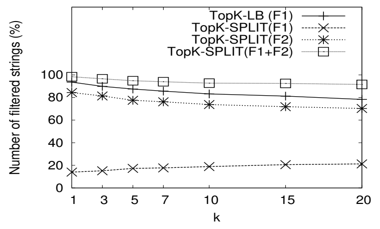
(b) Filtering effects (DBLP)



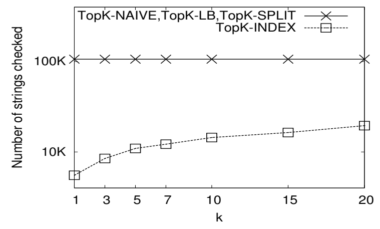
(c) Checked strings (DBLP)



(d) Execution times (Wikipedia)



(e) Filtering effects (Wikipedia)



(f) Checked strings (Wikipedia)

Figure 9: Varying k using DBLP and Wikipedia

n and the length of query strings L. Furthermore, we varied the length of q-grams q, MinHash signature size ℓ and buffer size B used. The default parameters are: k=5, q=3, ℓ=50 and B=512MB.

Varying k: We first report the execution times, the number of filtered strings and the percentage of strings read from D with varying k from 1 to 20 in Figure 9(a)–(f).

(1) Execution times: We show the execution times for DBLP and Wikipedia in Figure 9(a) and Figure 9(d) respectively. The log scale was used on the y-axises. In both data, *TopK-NGPP* and *TopK-FSS* show even worse performance than our naive algorithm *TopK-NAIVE*. The reason is that both of *TopK-NGPP* and *TopK-FSS* have to enumerate all substrings of every string in D to compute the substring edit distances. As expected from the experiments in [27], *TopK-NGPP* performs better than *TopK-FSS*.

As k is increased, the execution times for *TopK-LB*, *TopK-SPLIT* and *TopK-INDEX* grow gradually. Since the k-th smallest substring edit distance so far becomes larger with growing k, less number of strings are skipped for computing substring edit distances.

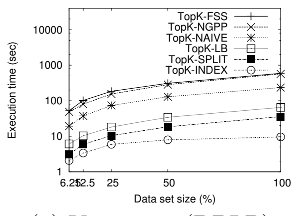
For every range of k, *TopK-INDEX* shows the best performance. *TopK-INDEX* is faster than *TopK-NAIVE* by at least 49.4 times and 5.5 times for DBLP and Wikipedia respectively. Furthermore, *TopK-SPLIT* is also faster than *TopK-NAIVE* by at least 11.6 and 3.5 times for DBLP and Wikipedia respectively.

(2) Filtering effects: To show the effectiveness of our filtering methods, we plotted the percentage of strings skipped for computing their substring edit distances in Figure 9(b) and Figure 9(e) with DBLP and Wikipedia respectively. Note that *TopK-LB(F1)* in the graphs represents the percentage of strings filtered with *TopK-LB* using the lower bounds $\ell o(d_{sub}(s, \sigma))$ obtained by *DYN-LB*. We also use *TopK-SPLIT(F1)* similarly in the graphs. *TopK-SPLIT(F2)* in the graphs represents the percentage of strings filtered by *TopK-SPLIT* using the lower bound of substring edit distances between a query string and a set of strings presented in Lemma 3.9. Furthermore, *TopK-SPLIT(F1+F2)* is the total ratio of strings filtered with *TopK-SPLIT*. Since the filtering effect of *TopK-INDEX* is exactly the same with that of *TopK-SPLIT*, we report the result of *TopK-SPLIT* only.

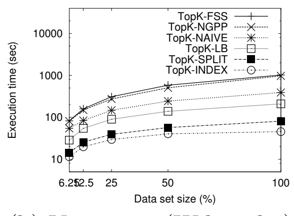
With every range of k, *TopK-SPLIT(F1+F2)* is always larger than *TopK-LB(F1)* in both data sets. For k=20 with Wikipedia, Figure 9(e) shows that *TopK-SPLIT* skips computing $d_{sub}(s, \sigma)$ with 17% more strings than *TopK-LB* does. Considering that *TopK-SPLIT* was 2.2 times faster than *TopK-LB* in Figure 9(d), we can conclude that using the lower bound by Lemma 3.9 improves the speed of query processing more than using the lower bound by *DYN-LB* does because the lower bound by Lemma 3.9 requires only to check whether each string contains at least a common q-gram with the query string in a constant time. With increasing k, both *TopK-LB(F1)* and *TopK-SPLIT(F1)* decrease slowly since the k-th smallest distance so far also becomes larger together with k. However, since the k-th substring edit distance found so far decreases slower as k grows, the chances that *TopK-SPLIT* skips the strings using the lower bound by *DYN-LB* increase on the contrary and thus *TopK-SPLIT(F2)* grows gradually.

(3) Number of strings read from D: We also plotted the number of data strings examined in disk with random I/O access by *TopK-INDEX* for DBLP and Wikipedia in Figure 9(c) and Figure 9(f) respectively. The y-axises are in a log scale. The graph confirms that *TopK-INDEX* effectively reduces the cost for reading the data strings in disk.

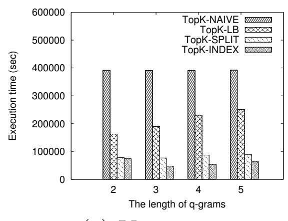
Varying n: With each of DBLP and Wikipedia, we selected the strings from the original data set to produce smaller data with varying the sampling rate from 6.25% to 100%. With varying the size of data, we plotted the execution times in Figure 10(a) for DBLP and in Figure 10(b) for Wikipedia. The graphs show that *TopK-INDEX* is the fastest in every range of data sizes. Both *TopK-NGPP* and *TopK-FSS* were even slower than *TopK-NAIVE* and their performance degrades linearly as data size grows because they have to examine every substring exhaustively. As the data size increases, the relative speedup of *TopK-INDEX* to *TopK-NAIVE* improves from 4.6 times to 8.6 times for Wikipedia. With DBLP, the speedup increases from 9.3 times to 24.5 times. Thus, we conclude that the performance of *TopK-INDEX*



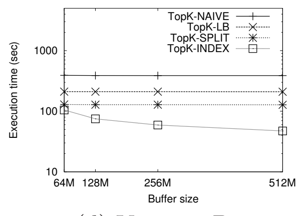
(a) Varying n (DBLP)



(b) Varying n (Wikipedia)

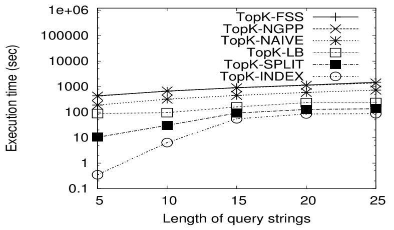


(c) Varying q

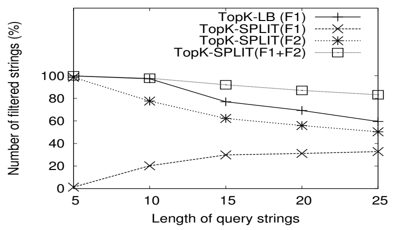


(d) Varying B

Figure 10: Varying n, q and B



(a) Execution times (Wikipedia)



(b) Filtering effects (Wikipedia)

Figure 11: Experiments with varying L

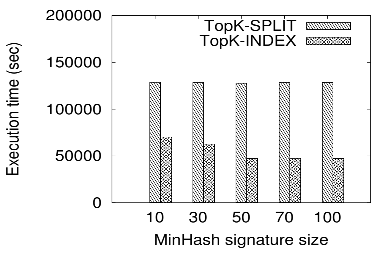


Figure 12: Varying ℓ (Wikipedia)

does not decline linearly with increasing data size and *TopK-INDEX* scales well to large data.

Varying L: With varying query lengths L from 5 to 25, we plotted the execution times for Wikipedia in Figure 11(a). The log scale was used on the y-axises. With every range of L, *TopK-INDEX* shows the best performance and *TopK-FSS* is the worst. When the lengths of query strings are 5 and 25, *TopK-INDEX* is 540 times and 8.44 times faster than *TopK-NAIVE* respectively. As L is increased, the execution times for all algorithms grow gradually. This is because there is a less chance that the strings in D have similar substrings to a longer query string and thus the k-th smallest substring edit distance so far becomes larger with growing L.

We also plotted the percentage of strings skipped for computing their substring edit distances $d_{sub}(s,\sigma)$ in Figure 11(b). With every range of L, *TopK-SPLIT* skips more strings for computing $d_{sub}(s,\sigma)$ than *TopK-LB* does. Furthermore, while *TopK-LB* computes $\ell o(d_{sub}(s,\sigma))$ based on dynamic programming algorithm *DYN-LB* for every string in D, *TopK-SPLIT* skips even computing $\ell o(d_{sub}(s,\sigma))$ for many strings. With increasing L, the gaps between the ratios of strings filtered by *TopK-SPLIT* and *TopK-LB* become larger because the query strings have more chance to have very selective q-grams resulting that more strings are filtered using the lower bound by Lemma 3.9.

Varying ℓ: We varied the MinHash signature size ℓ from 10 to 100 and plotted the running times of *TopK-SPLIT* and *TopK-INDEX* in Figure 12(a). Since the other algorithms do not estimate the computational cost to select G', they are not affected by the MinHash signature size. The graph shows that the performances of both algorithms are the worst when ℓ=10 and do not degrade that much with $\ell \leq 50$. Thus, we used ℓ=50 as the default value in all our experiments.

Varying q: Figure 10(c) shows the graph of execution time as the length q of the q-grams used is varied from 3 to 5. Since *TopK-NAIVE* is not affected by the length of q-grams, its execution times are always constant. *TopK-LB* and *TopK-SPLIT* show the best performances when q=2. This is because, with a large q, the lower bound $\lceil(|\sigma|-q+1-n)/q\rceil$ in Lemma 3.2 decreases and thus, less strings are skipped for computing $d_{sub}(s,\sigma)$. Furthermore, since the maximum of $\ell o(d_{sub}(D^-_{G'},\sigma))$, which is $\lfloor|\sigma|/q\rfloor$, due to Corollary 3.8 also becomes smaller with a larger q, the critical point string where we meet the k-th string whose substring edit distance is at most $\ell o(d_{sub}(D^-_{G'},\sigma))$ appears later. However, *TopK-INDEX* shows the best performance when q=3 because the posting lists become very large with 2-grams.

Varying B: With varying the size of buffer B from 64 MB to 512 MB, we show the execution times in Figure 10(d). With increasing B, the speed of *TopK-INDEX* is improved gradually since we have more chance to hit the cached pages of inverted indexes with larger buffer sizes.

7. RELATED WORK

We present the previous work on approximate string matching first and then approximate substring matching.

Approximate string matching: For approximate *string* matching with a maximum distance threshold τ, the count filtering technique proposed in [9] utilizes the fact that if the *edit distance* between s and σ is at most τ, they must share at least $(\max(|s|,|\sigma|)-q+1-\tau\cdot q)$ q-grams. In [16], another algorithm is proposed to speed up approximate *string* matching by reducing the average size of inverted lists using variable length q-grams. However, this algorithm utilizes the length filtering or prefix filtering which cannot be used for approximate *substring* matching. In [1] and [22], the algorithms to find similar strings *probabilistically* were proposed, but we focused on the problem of finding the exact top-k approximate substring matches in this paper.

In [3] and [15], the algorithms using inverted q-gram indexes are presented. However, as mentioned in [27], to utilize the inverted q-gram indexes, the length of the q-grams to be used should be smaller than $(|\sigma|+1)/(\tau+1)$. Thus, depending on the value of τ and the q-grams used for accessing existing inverted indexes, it is not always possible to use existing inverted q-gram indexes for approximate string matching [27]. In [26] and [28], the top-k approximate *string* matching algorithms using inverted q-gram indexes are presented. However, as pointed out in [27], we cannot determine the q-gram size in advance to build indexes and thus these algorithms may not find the top-k approximate *string* matches correctly when using preexisting inverted q-gram indexes. One of our proposed algorithms also utilizes existing inverted indexes, but we still guarantee the correctness.

The algorithms in [6, 29] utilize the suffix tries which index a small portion of suffixes only in the data. However, to use suffix tries for approximate *substring* matching, we have to index every suffix appearing in the data, resulting large index sizes. Because of the high space requirements and poor locality of suffix tries, it is mentioned in [21] that the suffix trie based approaches are proper only when both of the data and index fit in main memory. Furthermore, since the approximate *string* matching algorithm with a maximum threshold τ using suffix tries in [8] has exponential space and time complexity to τ, it is hard to be extended for top-k approximate *substring* matching.

Approximate substring matching: Approximate substring matching has been studied in the context of approximate entity extraction in [4, 17, 27]. With a given threshold and a entity dictionary, these algorithms are actually join algorithms which find every substring in a database such that the edit distance between the substring and one of the strings in the entity dictionary is at most the threshold. The algorithms in [4, 27] can be extended for top-k approximate substring matching and thus we compared our proposed algorithms to our extended algorithms of [4, 27]. The algorithm in [17] utilizes inverted q-gram indexes but to use inverted indexes for top-k approximate substring matching, we have to ensure that the length of the q-grams used must be smaller than $(|\sigma|+1)/(\tau_k+1)$ where τ_k is the k-th smallest substring edit distance in D. Since we cannot know τ_k in advance to build indexes, we could not adapt the algorithm in [17] to our top-k approximate substring matching problem.

To the best of our knowledge, no previous work addresses the top-k approximate substring matching problem and our algorithms presented here are the first work for the problem.

8. CONCLUSION

In this paper, we studied the problem of top-k approximate substring matching. We first proposed efficient filtering methods using q-grams which may enable us to skip strings without computing the actual substring edit distances to the query string. We next presented two algorithms *TopK-LB* and *TopK-SPLIT* which efficiently find top-k approximate substring matches by utilizing the filtering techniques. Furthermore, we developed an improved algorithm *TopK-INDEX* which utilizes inverted q-gram indexes to speed up *TopK-SPLIT*. By experiment results, we show the effectiveness and scalability of our algorithms with real-life data sets.

Acknowledgment

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MEST) (No. NRF-2009-0078828). It was also supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology (No. NRF-2012M3C4A7033342).

9. REFERENCES

[1] A. Andoni and K. Onak. Approximating edit distance in near-linear time. In *STOC*, 2009.

[2] R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval - the concepts and technology behind search, Second edition*. Pearson Education Ltd., Harlow, England, 2011.

[3] A. Behm, S. Ji, C. Li, and J. Lu. Space-constrained gram-based indexing for efficient approximate string search. In *ICDE*, 2009.

[4] T. Bocek, E. Hunt, and B. Stiller. Fast similarity search in large dictionaries. In *Technical Report*, 2007.

[5] A. Z. Broder. On the resemblance and containment of documents. In *Proceedings of Compression and Complexity of SEQUENCES*, 1997.

[6] S. Chaudhuri and R. Kaushik. Extending autocompletion to tolerate errors. In *SIGMOD*, 2009.

[7] Z. Chen, F. Korn, N. Koudas, and S. Muithukrishnan. Selectivity estimation for boolean queries. In *PODS*, 2000.

[8] R. Cole, L.-A. Gottlieb, and M. Lewenstein. Dictionary matching and indexing with errors and don't cares. In *STOC*, 2004.

[9] 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*, 2001.

[10] L. Jin and C. Li. Selectivity estimation for fuzzy string predicates in large data sets. In *VLDB*, 2005.

[11] N. Koudas, A. Marathe, and D. Srivastava. Flexible string matching against large databases in practice. In *VLDB*, 2004.

[12] H. Lee, R. T. Ng, and K. Shim. Extending q-grams to estimate selectivity of string matching with low edit distance. In *VLDB*, 2007.

[13] H. Lee, R. T. Ng, and K. Shim. Approximate substring selectivity estimation. In *EDBT*, 2009.

[14] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Phys. Dokl.*, 1985.

[15] C. Li, J. Lu, and Y. Lu. Efficient merging and filtering algorithms for approximate string searches. In *ICDE*, 2008.

[16] C. Li, B. Wang, and X. Yang. VGRAM: Improving performance of approximate queries on string collections using variable-length grams. In *VLDB*, 2007.

[17] G. Li, D. Deng, and J. Feng. Faerie: efficient filtering algorithms for approximate dictionary-based entity extraction. In *SIGMOD*, 2011.

[18] C.-C. Liu, J.-L. Hsu, and A. L. P. Chen. An approximate string matching algorithm for content-based music data retrieval. In *ICMCS*, 1999.

[19] A. Mazeika, M. H. Böhlen, N. Koudas, and D. Srivastava. Estimating the selectivity of approximate string queries. *ACM Trans. Database Syst.*, 32(2), 2007.

[20] G. Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, 2001.

[21] G. Navarro, E. Sutinen, and J. Tarhio. Indexing text with approximate q-grams. *J. Discrete Algorithms*, 2005.

[22] R. Ostrovsky and Y. Rabani. Low distortion embeddings for edit distance. *J. ACM*, 54(5), 2007.

[23] M. Sahami and T. D. Heilman. A web-based kernel function for measuring the similarity of short text snippets. In *WWW*, 2006.

[24] P. H. Sellers. The theory and computation of evolutionary distances: Pattern recognition. *J. Algorithms*, 1(4), 1980.

[25] E. Ukkonen. Finding approximate patterns in strings. *J. Algorithms*, 6(1):132–137, 1985.

[26] R. Vernica and C. Li. Efficient top-k algorithms for fuzzy search in string collections. In *KEYS*, 2009.

[27] W. Wang, C. Xiao, X. Lin, and C. Zhang. Efficient approximate entity extraction with edit distance constraints. In *SIGMOD*, 2009.

[28] Z. Yang, J. Yu, and M. Kitsuregawa. Fast algorithms for top-k approximate string matching. In *AAAI*, 2010.

[29] Z. Zhang, M. Hadjieleftheriou, B. C. Ooi, and D. Srivastava. Bed-tree: an all-purpose index structure for string similarity search based on edit distance. In *SIGMOD*, 2010.