Efficient String Similarity Search: A Cross Pivotal Based Approach

Fei Bi[1(✉)], Lijun Chang[1], Wenjie Zhang[1], and Xuemin Lin[1,2]

[1] University of New South Wales, Sydney, Australia
f.bi@student.unsw.edu.au, {ljchang,zhangw,lxue}@cse.unsw.edu.au
[2] East China Normal University, Shanghai, China

Abstract. In this paper, we study the problem of string similarity search with edit distance constraint; it retrieves all strings in a string database that are similar to a query string. The state-of-the-art approaches employ the concept of pivotal set, which is a set of non-overlapping signatures, for indexing and query processing. However, they do not fully exploit the pruning power potential of the pivotal sets by using only the pivotal set of the query string or the data strings. To remedy this issue, in this paper we propose a cross pivotal based approach to fully exploiting the pruning power of multiple pivotal sets. We prove theoretically that our cross pivotal filter has stronger pruning power than state-of-the-art filters. We also propose a more efficient algorithm with better time complexity for pivotal selection. Moreover, we further develop two advanced filters to prune unpromising single-match candidates which are the set of candidates introduced by one and only one of the probing signatures. Our experimental results on real datasets demonstrate that our cross pivotal based approach significantly outperforms the state-of-the-art approaches.

1 Introduction

The problem of string similarity search that finds similar strings in a string database to a query string has attracted a great deal of attentions recently. It is an important operation in data cleaning and data integration, and has many applications, such as duplicate detecting [1,6], spelling checking [8,21], and sequence alignment comparison in bioinformatics [9,16]. Edit distance [13] is a well-known metric to measure the similarity between two strings; two strings are similar if and only if their edit distance is no larger than a user-given threshold.

The existing studies for string similarity search with edit distance constraint in [2,4,5,8,11,12,14,15,21] have adopted the signature-based filter-verification framework. The framework consists of two phases: indexing phase and query processing phase. In the indexing phase, a set of signature based inverted indexes are constructed offline. During the query processing phase, firstly, a set of candidate strings are obtained by probing the inverted indexes with filtering conditions, and then each candidate string is verified to compute the similarity to the query string.

© Springer International Publishing Switzerland 2015
M. Renz et al. (Eds.): DASFAA 2015, Part I, LNCS 9049, pp. 545–564, 2015.
DOI: 10.1007/978-3-319-18120-2_32

In the literature, a variety of signature-based filters, such as the counter filter [11] and the prefix filter [3], are proposed to extract candidates. The number of signatures plays an important role on both the pruning power and the filtering cost of a filter, because it directly links to the number of probings to the inverted index as well as the number of candidates retrieved. It has been proven in [14] that the minimum number of signatures is $\tau + 1$, where τ is the user-given similarity threshold. Based on this result, the concept of pivotal set has been proposed for indexing, which indexes a set of $\tau + 1$ non-overlapping signatures. Several pivotal set based techniques [4,14,15] have recently been proposed to improve the pruning power and reduce the filtering cost and shown to outperform other existing works. Nevertheless, these techniques only exploit the single-side pivotal set for query processing; that is, they either use pivotal set of data string or query string [14,15], or choose dynamically during query processing [4].

In this paper, we propose a cross pivotal set based approach to further exploit the pruning power of multiple pivotal sets. Thus, we can achieve significantly less number of candidates than the state-of-the-art approaches; we also prove theoretically that the pruning power of the proposed cross pivotal filter is stronger than the state-of-the-art pivotal based filters. Through our empirical studies, we observe that a large portion of the candidates obtained by the pivotal based approaches are introduced by only one probing signature, and they are very unlikely to be the results; we call these candidates as single-match candidates. Due to the unique feature of our cross pivotal filter, these single-match candidates can be identified, and furthermore we propose two advanced filters to reduce the number of non-results in the set of single-match candidates. Our experimental results demonstrate that the proposed filters can significantly reduce the number of candidates with very little extra filtering cost.

Contributions. The contributions of this paper are summarized as follow:

- We propose a novel cross pivotal filter, which achieves much smaller number of candidates compared to the state-of-the-art approaches. We also prove the superiority of the proposed filter theoretically.
- We propose an efficient dynamic programming approach for pivotal selection, which runs in $O(q\tau^2)$ time in contrast to the $O(q^2\tau^3)$ time of the existing approach (where q is the length of signatures(q-grams)).
- We develop two advanced filters to further reduce the number of candidates.

We conduct extensive experimental studies and demonstrate that our cross pivotal based approach significantly outperforms the state-of-the-art approaches in terms of both candidate number and query processing time.

Organization. The rest of the paper is organized as follows. A brief overview of related work is given below. We formally define the problem in Section 2 and present our cross pivotal filter and the framework in Section 3. The two advanced filters are developed in Section 4. We show the extension of our approach in Section 5, and present the experimental results in Section 6. Finally, we conclude the paper in Section 7.

Related Work. The problem of string similarity search with edit distance constraint has been extensively studied [2,4,5,8,10–12,14,15,17,20,21], and most of the existing studies employ the signature-based filter-verification framework.

There are various representations of signatures, among which q-gram [7] is a very popular and effective one. Based on q-grams, the count filter [11] was first proposed as an effective method of pruning unpromising candidates, which requires that two similar strings must share a certain amount of q-grams. By ordering q-grams of each string according to an universal order, the prefix filter [3] ensures that two similar strings must share one common q-gram in their prefixes. Due to the simplicity of prefix filter, several effective methods such as the position prefix filter [19] and the mismatch filter [18], were designed to further improve the pruning power of the prefix filter.

Recently, several pivotal based filters [4,14,15] were proposed to reduce the signature size and gain more pruning power. The Q-Chunk method [14] employs the fixed-position q-chunks as signatures to perform filtering, and it devises two filters, IndexGram and IndexChunk, which utilize $\tau + 1$ q-chunks in the querying phase and in the indexing phase, respectively. IndexGram-Turbo and IndexChunk-Turbo [15] are designed to optimize Q-Chunk with floating q-chunks and to reduce filtering cost. Pivotal prefix filter [4] improves the prefix filter by probing the inverted index with a pivotal set. In this paper, we propose a new filter named cross pivotal filter which outperforms all the existing signature-based approaches.

2 Problem Definition

In this paper, we focus on a string database S which consists of a set of strings. For a string $s \in S$, we let $|s|$ denote the length of s and let $s.id$ denote the unique identifier of s in S.

Definition 2.1. (**Edit Distance**) The edit distance between two strings s and r, is the minimum number of edit operations required to transform one string to the other, denoted as $\mathsf{ed}(s, r)$. There exists three types of edit operations: insertion, deletion, and substitution. □

Example 2.1. Given two strings $s =$ "*datalearningx*" and $r =$ "*datacleanings*", the edit distance between s and r is $\mathsf{ed}(s, r) = 3$, since the optimal sequence of edit operations to transform s to r is: 1) insert 'c' before 'l'; 2) delete 'r'; 3) substitute 'x' with 's'. □

Problem Statement. Given a string database S, a query string r, and a threshold τ, we study the problem of *string similarity search*; that is, finding all strings $s \in S$ such that $\mathsf{ed}(s, r) \leq \tau$.

Notations. Frequently used notations are summarized in Table 1.

Table 1. Notation Table

Notation	Description
s / S / r	the data string / string database / query string
τ	the query threshold
$\mathsf{ed}(s, r)$	the edit distance between s and r
$\mathsf{q}(s)$ / $\mathsf{q}(r)$	the q-gram set of s / r
$\mathsf{pre}_x(s)$ / $\mathsf{pre}_x(r)$	the x-prefix set of s / r
$\mathsf{piv}(s)$ / $\mathsf{piv}(r)$	the pivotal set of s / r
$\mathsf{piv}_{\tau+1}(s)$ / $\mathsf{piv}_{\tau+1}(r)$	the $(\tau+1)$-th pivotal q-gram in $\mathsf{piv}(s)$ / $\mathsf{piv}(r)$, according to the universal order
lp_s / lp_r	the position of $\mathsf{piv}_{\tau+1}(s)$ / $\mathsf{piv}_{\tau+1}(r)$ in ordered $\mathsf{q}(s)$ / $\mathsf{q}(r)$
g / $g.pos$ / $g.order$	a q-gram / its start position / its universal order number

3 A Cross Pivotal Based Approach

In this section, we develop a new filter called cross pivotal filter, based on which we propose efficient query processing algorithms for string similarity search. In the following, we first present our cross pivotal filter in Section 3.1, then give our algorithm in Section 3.2, while an efficient algorithm for pivotal set selection is presented in Section 3.3.

3.1 Cross Pivotal Filter

Given two strings s and r, let $\mathsf{q}(s)$ and $\mathsf{q}(r)$ denote the sets of *q-grams* of s and r, respectively. We sort all q-grams in $\mathsf{q}(s)$ and $\mathsf{q}(r)$ by an universal order (e.g., in q-gram frequency ascending order), and denote the $(q\tau+1)$-*prefix* sets of $\mathsf{q}(s)$ and $\mathsf{q}(r)$ by $\mathsf{pre}_{\mathsf{q}\tau+1}(s)$ and $\mathsf{pre}_{\mathsf{q}\tau+1}(r)$, respectively. From $\mathsf{q}(s)$ and $\mathsf{q}(r)$, we respectively choose sets of $\tau+1$ disjoint q-grams as the *pivotal* sets of s and r, denoted as $\mathsf{piv}(s)$ and $\mathsf{piv}(r)$; two q-grams are defined to be disjoint if and only if they have no overlap (i.e., the difference between their start positions is not smaller than q). Then, we have the theorem below.

Theorem 3.1. *If two strings s and r are similar (i.e., $\mathsf{ed}(s, r) \leq \tau$), then $\mathsf{piv}(s) \cap \mathsf{q}(r) \neq \emptyset$ and $\mathsf{piv}(r) \cap \mathsf{q}(s) \neq \emptyset$.* □

Proof Sketch: We first prove by contradiction that if s and r are similar, then $\mathsf{piv}(s) \cap \mathsf{q}(r) \neq \emptyset$. Suppose $\mathsf{piv}(s) \cap \mathsf{q}(r) = \emptyset$, then none of the $\tau+1$ disjoint q-grams in $\mathsf{piv}(s)$ appears in $\mathsf{q}(r)$. Consequently, at least $\tau+1$ edit operations are required to transform $\mathsf{q}(s)$ into $\mathsf{q}(r)$ (i.e., one for each q-gram in $\mathsf{piv}(s)$); thus, $\mathsf{ed}(s, r) \geq \tau+1$ which is a contradiction. Therefore, $\mathsf{piv}(s) \cap \mathsf{q}(r) \neq \emptyset$. Similarly, we can prove that $\mathsf{piv}(r) \cap \mathsf{q}(s) \neq \emptyset$. □

Let $\mathsf{piv}_{\tau+1}(s)$ denote the $(\tau+1)$-th q-gram in $\mathsf{piv}(s)$ according to the universal order, and lp_s denote its *position* in the ordered $\mathsf{q}(s)$ (i.e., $\mathsf{piv}_{\tau+1}(s)$ is the lp_s-th q-gram in $\mathsf{q}(s)$ according to the universal order); lp_r is defined similarly. Then, we can further reduce $q(r)$ and $q(s)$ in Theorem 3.1 to be their prefix sets, as shown in the lemma below.

Lemma 3.1. *If two strings s and r are similar (i.e., $\mathsf{ed}(s,r) \leq \tau$), then we have $\mathsf{piv}(s) \cap \mathsf{pre}_{lp_s+(q-1)\tau}(r) \neq \emptyset$ and $\mathsf{piv}(r) \cap \mathsf{pre}_{lp_r+(q-1)\tau}(s) \neq \emptyset$.* □

Proof Sketch: If s is similar to r, then to transform s to r, at least one q-gram in $\mathsf{piv}(s)$ remains unchanged, and τ edit operations will introduce at most $q\tau$ new q-grams. Let g be the q-gram in $\mathsf{piv}(s)$ that has the smallest position in the ordered $\mathsf{q}(s)$ among all q-grams that remain unchanged, let x be its position in the ordered $\mathsf{q}(s)$, and let y be the number of q-grams before g in $\mathsf{piv}(s)$. Then, g must be in $\mathsf{pre}_{x-y+q\tau}(r)$, since the τ edit operations will introduce at most $q\tau$ new q-grams and also destroy at least y q-grams. Note that $x + (\tau - y) \leq lp_s$, where $(\tau - y)$ is the number of q-grams after g in $\mathsf{piv}(s)$. Therefore, $\mathsf{piv}(s) \cap \mathsf{pre}_{lp_s+(q-1)\tau}(r) \neq \emptyset$. Similarly, we can prove that $\mathsf{piv}(r) \cap \mathsf{pre}_{lp_r+(q-1)\tau}(s) \neq \emptyset$. □

If we select the pivotal set $\mathsf{piv}(s)$ from $\mathsf{pre}_{q\tau+1}(s)$, then $lp_s \leq q\tau + 1$; note that, the existence of such a selection is proven in [4]. Then, following from Lemma 3.1, we have the corollary below.

Corollary 3.1. *Given any two similar strings s and r, if the pivotal set is selected from the $(q\tau + 1)$-prefix, then $\mathsf{piv}(s) \cap \mathsf{pre}_{(2q-1)\tau+1}(r) \neq \emptyset$ and $\mathsf{piv}(r) \cap \mathsf{pre}_{(2q-1)\tau+1}(s) \neq \emptyset$.* □

Cross Pivotal Filter. Given any two strings s and r, if $\mathsf{piv}(s) \cap \mathsf{pre}_{lp_s+(q-1)\tau}(r) = \emptyset$ or $\mathsf{piv}(r) \cap \mathsf{pre}_{lp_r+(q-1)\tau}(s) = \emptyset$, then s and r cannot be similar.

Lemma 3.1 above proves the correctness of this filter.

Table 2. Dataset S and query string r $(q = 2)$

id	string	q-gram set ordered by global order
s_1	*datamining*	$\{\langle am,4\rangle\langle mi,5\rangle\langle da,1\rangle\langle ni,7\rangle\langle ng,9\rangle\langle ta,3\rangle\langle at,2\rangle\langle in,6\rangle\langle in,8\rangle\}$
s_2	*datalearning*	$\{\langle ar,7\rangle\langle rn,8\rangle\langle le,5\rangle\langle al,4\rangle\langle da,1\rangle\langle ea,6\rangle\langle ni,9\rangle\langle ng,11\rangle\langle ta,3\rangle\langle at,2\rangle\langle in,10\rangle\}$
s_3	*dutaleatings*	$\{\langle du,1\rangle\langle ut,2\rangle\langle ti,8\rangle\langle le,5\rangle\langle gs,11\rangle\langle al,4\rangle\langle ea,6\rangle\langle ng,10\rangle\langle ta,3\rangle\langle at,7\rangle\langle in,9\rangle\}$
s_4	*datalearnings*	$\{\langle ar,7\rangle\langle rn,8\rangle\langle le,5\rangle\langle gs,12\rangle\langle al,4\rangle\langle da,1\rangle\langle ea,6\rangle\langle ni,9\rangle\langle ng,11\rangle\langle ta,3\rangle\langle at,2\rangle\langle in,10\rangle\}$
s_5	*datalweatings*	$\{\langle lw,5\rangle\langle we,6\rangle\langle ti,9\rangle\langle gs,12\rangle\langle al,4\rangle\langle da,1\rangle\langle ea,7\rangle\langle at,2\rangle\langle ng,11\rangle\langle ta,3\rangle\langle at,8\rangle\langle in,10\rangle\}$
r	*datacleaning*	$\{\langle ac,4\rangle\langle cl,5\rangle\langle an,8\rangle\langle le,6\rangle\langle da,1\rangle\langle ea,7\rangle\langle ni,9\rangle\langle ng,11\rangle\langle ta,3\rangle\langle at,2\rangle\langle in,10\rangle\}$

Table 3. Universal order of q-grams (increasing frequency order)

Frequency	q-grams
1	$\langle 1:am\rangle\langle 2:an\rangle\langle 3:aw\rangle\langle 4:li\rangle\langle 5:lw\rangle\langle 6:mi\rangle\langle 7:we\rangle$
2	$\langle 8:ar\rangle\langle 9:du\rangle\langle 10:ut\rangle\langle 11:rn\rangle\langle 12:ti\rangle$
3	$\langle 13:le\rangle\langle 14:gs\rangle$
4	$\langle 15:al\rangle\langle 16:da\rangle\langle 17:ea\rangle\langle 18:ni\rangle$
5	$\langle 19:ng\rangle\langle 20:ta\rangle$
6	$\langle 21:at\rangle\langle 22:in\rangle$

Example 3.2. Consider string s_1 and string r in Table 2. Here, $q = 2$, $\tau = 2$, and the universal order of q-grams is shown in Table 3; for example, $\langle 1 : am\rangle$

in Table 3 indicates that am is the first q-gram in the universal order, and $\langle am, 4\rangle$ in Table 2 denotes that the start position of am in s_1 is 4. Assume that $\{am, da, ni\}$ is selected as $\mathsf{piv}(s_1)$ and $\{ac, an, le\}$ is selected as $\mathsf{piv}(r)$; then $lp_s = 4$, $lp_r = 4$. The prefix lengths are $lp_s + (q-1)\tau = 6$, and $lp_r + (q-1)\tau = 6$. Therefore, $\mathsf{pre}_{lp_r+(q-1)\tau}(s_1) = \{am, mi, da, ni, ng, ta\}$ and $\mathsf{pre}_{lp_s+(q-1)\tau}(r) = \{ac, cl, an, le, da, ea\}$. Then, we have $\mathsf{piv}(s_1) \cap \mathsf{pre}_{lp_s+(q-1)\tau}(r) = \{da\}$ and $\mathsf{piv}(r) \cap \mathsf{pre}_{lp_r+(q-1)\tau}(s_1) = \emptyset$. Thus, s_1 and r cannot be similar according to the cross pivotal filter. □

Compared with Existing Filters. In the literature, there are other filters studied, such as IndexChunk-Turbo [15], IndexGram-Turbo [15], and pivotal prefix filter [4]. Given a data string s and a query string r, the pruning condition (i.e., the condition that s and r cannot be similar) for IndexChunk-Turbo is $\mathsf{piv}(s) \cap \mathsf{pre}_{(2q-1)\tau+1}(r) = \emptyset$, and for IndexGram-Turbo is $\mathsf{piv}(r) \cap \mathsf{pre}_{(2q-1)\tau+1}(s) = \emptyset$; while for pivotal prefix filter it is $\mathsf{piv}(r) \cap \mathsf{pre}_{q\tau+1}(s) = \emptyset$ if $last(\mathsf{pre}_{q\tau+1}(r)) < last(\mathsf{pre}_{q\tau+1}(s))$, otherwise it is $\mathsf{piv}(s) \cap \mathsf{pre}_{q\tau+1}(r) = \emptyset$, where $last(\mathsf{pre}_{q\tau+1}(s))$ and $last(\mathsf{pre}_{q\tau+1}(r))$ denote the universal order of the last q-gram in $\mathsf{pre}_{q\tau+1}(s)$ and in $\mathsf{pre}_{q\tau+1}(r)$, respectively. We summarize the pruning conditions of these filters in Table 4.

Table 4. Pruning Conditions of Filters

Filter	Pruning condition
IndexChunk-Turbo	$\mathsf{piv}(s) \cap \mathsf{pre}_{(2q-1)\tau+1}(r) = \emptyset$
IndexGram-Turbo	$\mathsf{piv}(r) \cap \mathsf{pre}_{(2q-1)\tau+1}(s) = \emptyset$
Pivotal Prefix filter	$\mathsf{piv}(r) \cap \mathsf{pre}_{q\tau+1}(s) = \emptyset$, if $last(\mathsf{pre}_{q\tau+1}(r)) < last(\mathsf{pre}_{q\tau+1}(s))$ $\mathsf{piv}(s) \cap \mathsf{pre}_{q\tau+1}(r) = \emptyset$, if $last(\mathsf{pre}_{q\tau+1}(r)) \geq last(\mathsf{pre}_{q\tau+1}(s))$
Cross Pivotal filter	$\mathsf{piv}(r) \cap \mathsf{pre}_{lp_r+(q-1)\tau}(s) = \emptyset$ or $\mathsf{piv}(s) \cap \mathsf{pre}_{lp_s+(q-1)\tau}(r) = \emptyset$

We prove that our cross pivotal filter has the best pruning power among all filters in Table 4 in the theorem below.

Theorem 3.2. *Our cross pivotal filter has the best pruning power among all filters in Table 4.* □

Proof Sketch: Firstly, we have $lp_s \leq q\tau + 1$ and $lp_r \leq q\tau + 1$ if the pivotal sets are selected from the $(q\tau + 1)$-prefix sets. Thus, given a query string r, any data string s that is pruned by IndexChunk-Turbo or pruned by IndexGram-Turbo must also be pruned by our cross pivotal filter.

Secondly, for pivotal prefix filter, if $last(\mathsf{pre}_{q\tau+1}(r)) < last(\mathsf{pre}_{q\tau+1}(s))$, then $\mathsf{piv}(r) \cap \mathsf{pre}_{q\tau+1}(s) = \emptyset$ is equivalent to $\mathsf{piv}(r) \cap \mathsf{q}(s) = \emptyset$, which can be transferred into $\mathsf{piv}(r) \cap \mathsf{pre}_{lp_r+(q-1)\tau}(s) = \emptyset$ according to Corollary 3.1. Similarly, if $last(\mathsf{pre}_{q\tau+1}(r)) \geq last(\mathsf{pre}_{q\tau+1}(s))$, then $\mathsf{piv}(s) \cap \mathsf{pre}_{q\tau+1}(r) = \emptyset$ is equivalent to $\mathsf{piv}(s) \cap \mathsf{pre}_{lp_s+(q-1)\tau}(r) = \emptyset$. Thus, given a query string r, any data string s that is pruned by pivotal prefix filter must also be pruned by our cross pivotal filter.

Thus, the theorem holds. □

3.2 Cross Pivotal Based Approach

In this subsection, based on the proposed cross pivotal filter, we present our approach for string similarity search, which consists of two phases: Phase-I, indexing q-grams; and Phase-II, query processing.

Indexing. Given a query string r, every data string $s \in S$ that survives the cross pivotal filter has $\mathsf{piv}(s) \cap \mathsf{pre}_{\mathsf{lp}_s+(\mathsf{q}-1)\tau}(r) \neq \emptyset$ and $\mathsf{piv}(r) \cap \mathsf{pre}_{\mathsf{lp}_r+(\mathsf{q}-1)\tau}(s) \neq \emptyset$. Therefore, in order to efficiently retrieve all the candidate strings in S that pass the cross pivotal filter, we construct inverted index on the pivotal set and the prefix set for each data string in S. However, the query string r is not given at the time of indexing, so lp_r is unknown; moreover, lp_s may vary for different data strings $s \in S$. Therefore, we select $\mathsf{piv}(s)$ and $\mathsf{piv}(r)$ from $\mathsf{pre}_{\mathsf{q}\tau+1}(s)$ and $\mathsf{pre}_{\mathsf{q}\tau+1}(r)$, respectively, and set lp_s and lp_r to be $q\tau + 1$ which is their upper bound.

We denote the inverted index constructed for the pivotal sets of all data strings as I_{piv}, and denote the inverted index constructed for the $((2q-1)\tau+1)$-prefix sets of all data strings as I_{pre}. In the inverted indexes, for each q-gram g, we store not only the ids of strings that contain g but also the start positions of g in the corresponding strings; the start positions are stored to enable position filtering during query processing.

The pseudocode is shown in Algorithm 1, denoted Indexing. We first generate all the q-grams $\mathsf{Q}(s)$ for each string $s \in S$ and count the frequency of the generated q-grams as well (Line 1). Then, we sort the set of all generated q-grams for all strings in S in increasing frequency order, which defines the universal order of q-grams (Line 2). The inverted indexes I_{piv} and I_{pre} are initialized to be empty (Line 3). Then, we process each string s in S (Lines 4-13). For string s, we first obtain and store the length $|s|$ of s (Line 5), and then sort $\mathsf{q}(s)$ according to the universal order (Line 6). The $((2q-1)\tau+1)$-prefix of s is selected and indexed by I_{pre} (Lines 7-9). The pivotal set $\mathsf{piv}(s)$ of s is chosen from the $(q\tau+1)$-prefix of $\mathsf{q}(s)$ by algorithm PivotalSelection which will be discussed in Section 3.3 (Lines 10-11), and is indexed by I_{piv} (Lines 12-13).

Example 3.3. Consider the string database $S = \{s_1, \ldots, s_5\}$ in Table 2 with $q = 2$ and $\tau = 2$; then $(2q-1)\tau + 1 = 7$. For string s_1, $\mathsf{pre}_{(2q-1)\tau+1}(s_1) = \{am, mi, da, ni, ng, ta, at\}$ and assume $\mathsf{piv}(s_1) = \{am, da, ni\}$. Then, the ⟨id, start position⟩ pairs $\langle s_1, 4\rangle$, $\langle s_1, 5\rangle$, $\langle s_1, 1\rangle$, $\langle s_1, 7\rangle$, $\langle s_1, 9\rangle$, $\langle s_1, 3\rangle$, and $\langle s_1, 2\rangle$ are put into $I_{pre}[am]$, $I_{pre}[mi]$, $I_{pre}[da]$, $I_{pre}[ni]$, $I_{pre}[ng]$, $I_{pre}[ta]$, and $I_{pre}[at]$, respectively. Similarly, for the inverted index of pivotal sets, $\langle s_1, 4\rangle$, $\langle s_1, 1\rangle$ and $\langle s_1, 7\rangle$ are put into $I_{piv}[am]$, $I_{piv}[da]$ and $I_{piv}[ni]$, respectively. The final inverted index I_{pre} and I_{piv} are shown in Table 5. □

Time Complexity. The time complexity of Algorithm 1 is $O(\sum_{s\in S}|s| \log \sum_{s\in S}|s|)$, if we exclude the pivotal selecting time consumed by PivotalSelection. The reason is that putting a q-gram into I_{pre} or I_{piv} takes constant time, and Line 2 of Algorithm 1 is the dominating cost.

Algorithm 1. INDEXING

Input: String database S
Output: Inverted index I_{piv} for pivotal sets, and I_{pre} for $((2q-1)\tau+1)$-prefix sets

```
1  Generate q-grams q(s) for all strings s ∈ S;
2  Sort all q-grams in S in increasing frequency order, which defines the universal order;
3  I_piv ← ∅, I_pre ← ∅;
4  for each s ∈ S do
5      Obtain and store the length of s;
6      Sort q(s) according to the universal order;
       /* Lines 7-9: Index prefix set                                   */
7      pre_{(2q-1)τ+1}(s) ← {the first (2q − 1)τ + 1 q-grams in q(s)};
8      for each q-gram g ∈ pre_{(2q-1)τ+1}(s) do
9          I_pre[g] ← I_pre[g] ∪ ⟨s.id, g.pos⟩;
       /* Lines 10-13: Index pivotal set                                */
10     pre_{qτ+1}(s) ← {the first qτ + 1 q-grams in q(s)};
11     piv(s) ← PivotalSelection(pre_{qτ+1}(s));
12     for each q-gram g ∈ piv(s) do
13         I_piv[g] ← I_piv[g] ∪ ⟨s.id, g.pos⟩;
```

Table 5. Example of Inverted Index for String Database S

I_{pre} for dataset S	I_{piv} for dataset S
$am \rightarrow \langle s_1,4\rangle$; $at \rightarrow \langle s_1,2\rangle$; $du \rightarrow \langle s_3,1\rangle$; $lw \rightarrow \langle s_5,5\rangle$; $mi \rightarrow \langle s_1,5\rangle$;	$am \rightarrow \langle s_1,4\rangle$; $du \rightarrow \langle s_3,1\rangle$;
$ng \rightarrow \langle s_1,9\rangle$; $ta \rightarrow \langle s_1,3\rangle$; $ut \rightarrow \langle s_3,2\rangle$; $we \rightarrow \langle s_5,6\rangle$;	$lw \rightarrow \langle s_5,5\rangle$; $ni \rightarrow \langle s_1,7\rangle$;
$ar \rightarrow \langle s_2,7\rangle, \langle s_4,7\rangle$; $ni \rightarrow \langle s_1,7\rangle, \langle s_2,9\rangle$; $rn \rightarrow \langle s_2,8\rangle, \langle s_4,8\rangle$;	$ar \rightarrow \langle s_2,7\rangle, \langle s_4,7\rangle$;
$ti \rightarrow \langle s_3,8\rangle, \langle s_5,9\rangle$;	$da \rightarrow \langle s_1,1\rangle, \langle s_2,1\rangle$;
$gs \rightarrow \langle s_3,11\rangle, \langle s_4,12\rangle, \langle s_5,12\rangle$; $le \rightarrow \langle s_2,5\rangle, \langle s_3,5\rangle, \langle s_4,5\rangle$;	$gs \rightarrow \langle s_4,12\rangle, \langle s_5,12\rangle$;
$al \rightarrow \langle s_2,4\rangle, \langle s_3,4\rangle, \langle s_4,4\rangle, \langle s_5,4\rangle$; $da \rightarrow \langle s_1,1\rangle, \langle s_2,1\rangle, \langle s_4,1\rangle, \langle s_5,1\rangle$;	$ti \rightarrow \langle s_3,8\rangle, \langle s_5,9\rangle$;
$ea \rightarrow \langle s_2,6\rangle, \langle s_3,6\rangle, \langle s_4,6\rangle, \langle s_5,7\rangle$;	$le \rightarrow \langle s_2,5\rangle, \langle s_3,5\rangle, \langle s_4,5\rangle$;

Query Processing. Given the inverted indexes I_{piv} and I_{pre} constructed for a string database S, for any query string r, our query processing algorithm consists of two stages: stage-I, generating candidate sets, and stage-II, verifying each string in the candidate set to find the true similar strings. The candidate set is defined as the set of strings in S that pass the cross pivotal filter as proposed in Section 3.1; that is, $\{s \in S \mid \mathsf{piv}(s) \cap \mathsf{pre}_{(2q-1)\tau+1}(r) \neq \emptyset, \mathsf{piv}(r) \cap \mathsf{pre}_{(2q-1)\tau+1}(s) \neq \emptyset\}$.

To obtain the candidate set, we first use the q-grams in the prefix of $\mathsf{q}(r)$, $\mathsf{pre}_{(2q-1)\tau+1}(r)$, to query the inverted index I_{piv} to generate an initial set of candidates. Here, we also apply the *length filtering* and the *position filtering*. For each q-gram $g \in \mathsf{pre}_{(2q-1)\tau+1}(r)$, and each entry in the inverted list of g (i.e., $\langle sid, pos\rangle \in I_{piv}[g]$), the length filtering requires that the length of the string with id sid must be within the range $[|r|-\tau, |r|+\tau]$, and the position filtering requires that the position difference must satisfy $|g.pos - pos| \leq \tau$. Then, we use the pivotal q-grams in $\mathsf{piv}(r)$ to query the inverted index I_{pre} to further refine the candidate set using the position filtering again.

The pseudocode is shown in Algorithm 2, denoted as SEARCH. Firstly, we generate the q-gram set $\mathsf{q}(r)$ of the query string r, and sort it according to the universal order (Line 1). Then, through Lines 2-8, we generate the initial candidate set by using the prefix of $\mathsf{q}(r)$, $\mathsf{pre}_{(2q-1)\tau+1}(r)$, to probe the inverted index I_{piv}. That is, for each q-gram $g \in \mathsf{pre}_{(2q-1)\tau+1}(r)$ (Line 4), and each pair $\langle sid, pos\rangle \in I_{pre}[g]$ (Line 5), we add sid to the candidate set if its corresponding string passes both the length filtering and the position filtering (Lines 7-8). Next, through Lines 9-15, we refine the candidate set by using the pivotal set $\mathsf{piv}(r)$ to probe the inverted index I_{pre}. Finally, for each string s in the candidate set, we verify it by checking whether $\mathsf{ed}(s, r)$ is larger than τ or not (Lines 16-20); note that, here we use the same verification algorithm as that was used in the existing works [4,15].

Note that, we can also apply the tight cross pivotal filter as presented in Lemma 3.1 at Lines 7,14, by storing the lp_s for each sid in I_{piv} and the position of each q-gram g in the prefix of the string with id sid in I_{pre}. For presentation briefness, we omit the details.

Example 3.4. Consider the inverted indexes I_{piv} and I_{pre} built in Example 3.3 and the query string r in Table 2. $\mathsf{pre}_{(2q-1)\tau+1}(r) = \{\langle ac, 4\rangle, \langle cl, 5\rangle, \langle an, 8\rangle, \langle le, 6\rangle, \langle da, 1\rangle, \langle ea, 7\rangle, \langle ni, 9\rangle\}$ where the numbers indicate the start position of the corresponding q-gram in r, and assume $\mathsf{piv}(r)$ is chosen as $\{\langle ac, 4\rangle, \langle an, 8\rangle, \langle le, 6\rangle\}$. The algorithm starts by using $\mathsf{pre}_{(2q-1)\tau+1}(r)$ to probe the inverted index I_{piv} with both the length filtering and the position filtering. Taking $\langle da, 1\rangle$ in $\mathsf{pre}_{(2q-1)\tau+1}(r)$ as an example, we have $I_{piv}[da] = \{s_1, s_2\}$ and both of them pass the length filtering and the position filtering, thus, the candidates obtained by querying I_{piv} with $\langle da, 1\rangle$, denoted as $cand_{piv}(da, 1)$, are $\{s_1, s_2\}$. Similarly, we have $cand_{piv}(ac, 4) = cand_{piv}(cl, 5) = cand_{piv}(an, 8) = cand_{piv}(ea, 7) = \emptyset$, $cand_{piv}(le, 6) = \{s_2, s_3, s_4\}$ and $cand_{piv}(ni, 9) = \{s_1\}$. Merging them altogether, we obtain the candidate set obtained by querying I_{pre}, denoted as $cand_{piv}$, which is $\{s_1, s_2, s_3, s_4\}$. Then $\mathsf{piv}(r)$ is used to probe I_{pre} to extract candidates. We denote the candidates obtained by $\langle ac, 4\rangle$ as $cand_{pre}(ac, 4)$; then, we have $cand_{pre}(ac, 4) = cand_{pre}(an, 8) = \emptyset$ and $cand_{pre}(le, 6) = \{s_2, s_3, s_4\}$. Therefore, the candidate set obtained by querying I_{Pre}, denoted as $cand_{pre}$, is $\{s_2, s_3, s_4\}$. Finally, we get the candidate set $cand = cand_{piv} \cap cand_{pre} = \{s_2, s_4, s_5\}$, with s_1 in $cand_{piv}$ being pruned. □

Time Complexity. Obviously, the two inverted-index probing processes (i.e., Lines 4-8 and Lines 12-15) dominate the time complexity of Algorithm 2. For each q-gram g in $\mathsf{pre}_{(2q-1)\tau+1}$, it needs $O(|I_{piv}[g]|)$ time to probe the inverted list. Therefore the time complexity for the first probing (i.e., Lines 4-8) is $O(\sum_{g\in\mathsf{pre}_{(2q-1)\tau+1}(r)} |I_{piv}[g]|)$. Similar, we obtain that the time complexity for the second probing (i.e., Lines 12-15) is $O(\sum_{g\in\mathsf{piv}(r)} |I_{pre}[g]|)$. Consequently, the total time complexity of Algorithm 2 (excluding the time for verification at Lines 17-19) is $O(\sum_{g\in\mathsf{pre}_{(2q-1)\tau+1}(r)} |I_{piv}[g]| + \sum_{g\in\mathsf{piv}(r)} |I_{pre}[g]|)$.

Algorithm 2. SEARCH

Input: String dataset S, inverted indexes I_{piv} and I_{pre}, and query string r
Output: All the similar strings $Res \subseteq S$ to r, (i.e., $Res = \{s \in S \mid \mathsf{ed}(s, r) \leq \tau\}$)

```
1  q(r) ← {the q-grams of r sorted according to the universal order};
   /* Lines 2-8: Candidate_temp ← {s ∈ S | piv(s) ∩ pre_{(2q−1)τ+1}(r) ≠ ∅}   */
2  pre_{(2q−1)τ+1}(r) ← {the first (2q − 1)τ + 1 q-grams in q(r)};
3  Candidate_temp ← ∅;
4  for each q-gram g ∈ pre_{(2q−1)τ+1}(r) do
5      for each pair ⟨sid, pos⟩ ∈ I_piv[g] do
6          Let s be the string with id sid;
7          if |s| ∈ [|r| − τ, |r| + τ] and pos ∈ [g.pos − τ, g.pos + τ] then
8              Candidate_temp ← Candidate_temp ∪ {sid};
   /* Lines 9-15: Candidate ← {s ∈ Candidate_temp | piv(r) ∩ pre_{(2q−1)τ+1}(s) ≠ ∅}   */
9  pre_{qτ+1}(r) ← {the first qτ + 1 q-grams in q(r)};
10 piv(r) ← PivotalSelection (pre_{qτ+1}(r));
11 Candidate ← ∅;
12 for each q-gram g ∈ piv(r) do
13     for each pair ⟨sid, pos⟩ ∈ I_pre[g] do
14         if sid ∈ Candidate_temp and pos ∈ [g.pos − τ, g.pos + τ] then
15             Candidate ← Candidate ∪ {sid};
   /* Lines 16-19: Verification                                            */
16 Res ← ∅;
17 for each sid ∈ Candidate do
18     Let s be the string with id sid;
19     if Verify (s, r) then Res ← Res ∪ {s};
20 return Res;
```

3.3 Pivotal Set Selection

In this section, we propose a new efficient dynamic programming algorithm for selecting the pivotal set in $O(q\tau^2)$ time.[1]

Objective Function. In Table 4, we have shown that the pruning condition of cross pivotal filter is $\mathsf{piv}(r) \cap \mathsf{pre}_{lp_r+(q-1)\tau}(s) = \emptyset$ or $\mathsf{piv}(s) \cap \mathsf{pre}_{lp_s+(q-1)\tau}(r) = \emptyset$. Intuitively, the smaller lp_s and lp_r are, the shorter the prefixes of s and r in the cross pivotal filter, and thus the more powerful the pruning condition. Hence, we compute the pivotal set with the aim of minimizing lp_s and lp_r, which is equivalent to minimizing the maximum order of the selected pivotal q-grams. Note that, each q-gram has an universal order number, denoted $g.order$, which is the position of g in the sorted set of all q-grams in the string database (i.e., in the universal order).

[1] Note that, our method is different from the algorithm in [4] which runs in $O(q^2\tau^3)$ time.

Algorithm. We develop a dynamic programming algorithm to find the pivotal set (i.e., $\tau+1$ disjoint q-grams) with the minimum max-order from the q-grams q of a string. It has been proven in [4] that there always exists a pivotal set in the $(q\tau+1)$-prefix of q; thus, it is sufficient for us to take the $(q\tau+1)$-prefix $\mathsf{pre}_{q\tau+1}$ as input. That is, the optimal pivotal set will be in $\mathsf{pre}_{q\tau+1}$.

We sort q-grams in $\mathsf{pre}_{q\tau+1}(r)$ by their start positions, and let g_i denote the i-th q-gram in this order with $1 \leq i \leq q\tau+1$. For each q-gram g_i in $\mathsf{pre}_{q\tau+1}(r)$, we let $ld(i)$ denote the last non-overlapping q-gram that appears before g_i; that is, $ld(i) = \max\{1 \leq j < i \mid g_i.pos - g_j.pos \geq q\}$. If all q-grams before g_i overlap with g_i, then $ld(i)$ is set to 0.

Now, we let $mmo(i,j)$ denote the optimal solution (i.e., with minimum max-order) of selecting j disjoint q-grams from $\{g_1,\ldots,g_i\}$. Then, $mmo(i,j)$ can be obtained from two cases: 1) including g_i, thus the other part of the solution is $mmo(ld(i),j-1)$; and 2) not including g_i, thus, it is the same as $mmo(i-1,j)$ by greedy selection. Therefore, we can compare the two cases, and choose the one with smaller max-order.

Let $mmo(i,j).maxOrder$ denote the max-order of the q-grams in $mmo(i,j)$, let $order_{in}$ denote $\max\{mmo(ld(i),j-1).maxOrder, g_i.order\}$, and let $order_{not\_in}$ denote $mmo(i-1,j).maxOrder$. Then we can compute $mmo(i,j)$ recursively as follows,

$$\begin{cases} mmo(i,j) \leftarrow mmo(i-1,j), & If \quad order_{not\_in} < order_{in} \\ mmo(i,j) \leftarrow mmo(ld(i),j-1) \cup g_i, & If \quad order_{not\_in} \geq order_{in} \end{cases} \tag{1}$$

Lemma 3.2. *Equation (1) correctly computes* $mmo(i,j)$ *for all* $1 \leq i \leq q\tau+1$ *and* $1 \leq j \leq \tau+1$. □

Proof Sketch: Obviously, the optimal solution $mmo(i,j)$ must be one of the two cases: 1) including g_i; and 2) not including g_i. For case 1), since g_i is included in the solution, then the other part of the solution must be $mmo(ld(i),j-1)$ by greedy selection due to the fact that $mmo(i',j') \geq mmo(i'+1,j')$ for all i' and j'. For case 2), as g_i is not selected, the optimal solution must be $mmo(i-1,j)$ by greedy selection. Thus, $mmo(i,j)$ is the better one between the above two cases. Thus, the lemma holds. □

Following from the above, the pseudocode is shown in Algorithm 3, denoted as PivotalSelection. It starts by sorting the set of q-grams by their start positions (Line 1). $ld(i)$s are computed at Lines 3-6, while Lines 7-12 compute $mmo(i,j)$ following Equation (1). Note that, for efficiency concerns, in Algorithm 3, $mmo(i,j)$ actually stores $mmo(i,j).maxOrder$; that is, $mmo(i,j)$ stores the max-order of the set of selected q-grams instead of the actual q-grams. Finally, we obtain the optimal pivotal set by backtracking on $mmo(i,j)$ (Lines 13-16); the observation is that, the q-gram at position ri of p is not selected in the pivotal set if and only if $mmo(ri,i)$ equals $mmo(ri-1,i)$, and we start the construction from $mmo(q\tau+1,\tau)$.

Time Complexity. The time complexity of Algorithm 3 is $O(q\tau^2)$. It is easy to verify that the $ld(i)$s are constructed in $O(q\tau)$ time at Lines 3-6, the $mmo(i,j)$s

Algorithm 3. PivotalSelection

```
   Input: A set of q-grams p
   Output: The pivotal set piv of p with minimum max-order
1  Sort q-grams in p by their start positions;
2  Allocate arrays ld[qτ + 1] and mmo[qτ + 1][τ + 1];
   /* Lines 3-6: Compute ld[1], ..., ld[qτ + 1]                    */
3  ld[1] ← 0;
4  for i ← 2 to qτ + 1 do
5      ld[i] ← ld[i − 1];
6      while g_i.pos − g_{ld[i]+1}.pos ≥ q do ld[i] ← ld[i] + 1;
   /* Lines 7-12: Compute mmo(i, j)                                */
7  mmo[1][1] ← g_1.order;
8  for i ← 2 to qτ + 1 do
9      mmo[i][1] ← min{mmo[i − 1][1], g_i.order};
10     for j ← 2 to τ + 1 do
11         maxOrder ← max{mmo[ld[i]][j − 1], g_i.order};
12         mmo[i][j] ← min{maxOrder, mmo[i − 1][j]};
   /* Lines 13-16: Construct the pivotal set                       */
13 piv ← ∅; ri ← qτ + 1;
14 for i ← τ + 1  to  1 do
15     while mmo[ri][i] = mmo[ri − 1][i] do ri ← ri − 1;
16     piv[i] ← the q-gram at position ri of p; ri ← ld[ri];
17 return piv;
```

are computed in $O(q\tau^2)$ at Lines 7-12, and Lines 13-16 build the pivotal set in $O(q\tau)$ time. Note that, a similar but different dynamic programming approach has been proposed in [4] for selecting weight-based pivotal set in $O(q^2\tau^3)$ time. Therefore, our pivotal selection algorithm is more efficient.

4 Advanced Filters

As it is time-consuming to verify each candidate by checking whether the edit distance between the candidate string and the query string is larger than τ, in this section we propose two advanced filters to further refine the candidate set, thus reduce the number of candidates to be verified. Before that, we first introduce the concept of single-match candidate.

Definition 4.2. (**Single-Match Candidates**) A candidate string s is called a single-match candidate if s contains only one q-gram of the pivotal set of the query string r and moreover, that pivotal q-gram has only one copy in r. □

Through our experiments, we observe that a large portion (i.e., 50% to 95%) of the candidates are single-match candidates; that is, they are introduced by only one pivotal q-gram of r. However, many of the single-match candidates are not included in the final result. For example, in **DNA** dataset with τ being 10, out of 1.4 million candidates, there are 1 million single-match candidates, none of which contributes to the actual result; in **Title** and **URL** with τ being 10

and 3 respectively, the number of all candidates, single-match candidates, and actual results that are single-match candidates are 9.5 millions, 8.1 millions, 0 and 18.8 millions, 17.4 millions, 9030 respectively.

Example 4.5. Consider the three candidates s_2, s_3 and s_4 computed by SEARCH in Example 3.4. Obviously, they are single-match candidates, all of which are solely introduced by the pivotal q-gram $\langle le, 6\rangle$ that has only one matched position (i.e., start position 5) in each of the three candidate strings. □

Motivated by the above, in the following we propose two advanced filters, pivotal substitution filter and position match filter, to refine the single-match candidates.

4.1 Pivotal Substitution Filter

For our cross pivotal filter, a natural extension of Theorem 3.1 is that, let $\mathsf{piv}(r)$ and $\mathsf{piv}'(r)$ denote two different pivotal sets computed from $\mathsf{q}(r)$, if s and r are similar, then $\mathsf{piv}(s) \cap \mathsf{q}(r) \neq \emptyset$ and $\mathsf{piv}(r) \cap \mathsf{q}(s) \neq \emptyset$ and $\mathsf{piv}'(r) \cap \mathsf{q}(s) \neq \emptyset$. Moreover, this can be extended to many more pivotal sets. However, the query with multiple pivotal sets would involve significant filtering cost, as it increases the number of strings required to be processed. To address this issue, based on the above observation of single-match candidates, we propose the pivotal substitution filter to perform multiple pivotal queries efficiently.

Definition 4.3. (**Pivotal Substitution**) Given a pivotal set $\mathsf{piv}(r)$ and a q-gram $g_i \in \mathsf{piv}(r)$, a pivotal substitution of g_i is a q-gram from $\mathsf{q}(r)$ that is disjoint with the other τ pivotal q-grams of $\mathsf{piv}(r)$. □

Pivotal Substitution Filter. For a single-match candidate s introduced by the i-th pivotal q-gram g_i, if there exists such a pivotal substitution of g_i in $\mathsf{q}(r)$ that is not in $\mathsf{q}(s)$, then s and r cannot be similar.

The intuition of the pivotal substitution filter is as follows. Let g' be such a pivotal substitution, and let $\mathsf{piv}'(r)$ be the result of substituting g_i with g' in $\mathsf{piv}(r)$. Then, $\mathsf{piv}'(r) \cap \mathsf{q}(s) = \emptyset$; thus, s and r are not similar.

Example 4.6. Consider, the candidate s_3 in Example 4.5. For s_3, the matched pivotal q-gram is $\langle le, 6\rangle$. Assume we substitute it with $\langle da, 1\rangle$, but $\langle da, 1\rangle$ is not in $\mathsf{q}(s_3)$; thus, s_3 is pruned by the pivotal substitution filter. □

Implementation and Time Complexity. To minimize the filtering cost, when selecting a pivotal substitution for a pivotal q-gram g_i, we select the one with minimum universal order among all q-grams in $\mathsf{q}(r)$ that satisfy the non-overlapping constraint. Once a pivotal substitution g' is selected, the candidate set is refined by conducting an intersection with the inverted list of g'. Note that, we perform the above pivotal substitution filter for each pivotal in $\mathsf{piv}(r)$. Thus, the time complexity of applying each pivotal substitution filter linear to the size of the inverted list of the pivotal substitution (i.e., $O(|I_{pre}[g']|)$).

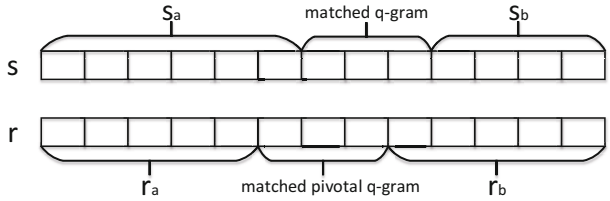


Fig. 1. A Demonstration of Pivotal Match Filter

4.2 Position Match Filter

In this subsection, we propose the position match filter for single-match candidates based on their matched positions.

Position Match Filter. For a single-match candidate s that is introduced by the i-th pivotal q-gram g_i in $\mathsf{piv}(r)$ and has a matched position pos, if $|pos - g_i.pos| > i-1$ or $|(|s|-pos)-(|r|-g_i.pos)| > \tau+1-i$, then s cannot be similar to the query string r.

We prove the correctness of the position match filter by showing that, assuming s and r are similar, then $|pos-g_i.pos| \leq i-1$ and $|(|s|-pos)-(|r|-g_i.pos)| \leq \tau+1-i$. Since s is a single-match candidate and only matches the pivotal q-gram g_i, then g_i must be matched to the q-gram of s with start position pos and remains unchanged in the optimal sequence of edit operations. As illustrated in Figure 1, let s_a and s_b denote the left substring and the right substring in s separated by the matched q-gram respectively, and r_a and r_b are defined similarly. Then, we have $\mathsf{ed}(s,r) = \mathsf{ed}(s_a,r_a) + \mathsf{ed}(s_b,r_b)$.

Moreover, we have $\mathsf{ed}(s_a,r_a) \geq i-1$ and $\mathsf{ed}(s_b,r_b) \geq \tau+1-i$, since there are $i-1$ $(\tau+1-i)$ unmatched pivotal q-grams between s_a and r_a (s_b and r_b), respectively. For s and r being similar, $ed(s,r) \leq \tau$. Thus, $\mathsf{ed}(s_a,r_a) \leq \mathsf{ed}(s,r)-\mathsf{ed}(s_b,r_b) \leq i-1$, and $\mathsf{ed}(s_b,r_b) \leq \tau+1-i$. Therefore, $\mathsf{ed}(s_a,r_a) = i-1$ and $\mathsf{ed}(s_b,r_b) = \tau+1-i$.

Note that $\mathsf{ed}(s_a,r_a) \geq |pos - g_i.pos|$ and $\mathsf{ed}(s_b,r_b) \geq |(|s| - pos - q + 1) - (|r| - g_i.pos - q + 1)| = |(|s| - pos) - (|r| - g_i.pos)|$, due to lower bounding by length difference. Therefore, the position match filter is correct.

Example 4.7. As mentioned in Example 4.5, s_4 is a single-match candidate introduced by $\langle le, 6\rangle$ with matched position $pos = 5$. Now we apply the position match filter to s_4. The length difference of the two substrings on the left is $|pos - g_i.pos| = |5-6| = 1$ and the length difference of the two on the right is $|(|s| - pos) - (|r| - g_i.pos)| = |(13-5)-(12-6)| = 2$. Also, we have $i = 2$. Therefore, $|pos - g_i.pos| = 1 = i-1$ and $|(|s| - pos) - (|r| - g_i.pos - q)| = 2 > \tau+1-i = 1$. Consequently, s_4 is pruned by the position match filter. □

Time Complexity. Obviously, the running time for applying position match filter on each single-match candidate is constant. Thus, this filter can be applied very efficiently.

5 Extension for Dynamic Thresholds

In this section, we extend our techniques to support dynamic thresholds. Given a maximum threshold τ_{max}, we compute the pivotal set piv for τ_{max}, and then for each $1 \leq \tau \leq \tau_{max}$, the pivotal set for τ is selected from piv. In the indexing phase, we construct two sets of inverted indexes $I_{pre} = \{I^0_{pre}, I^1_{pre} \ldots I^{\tau_{max}}_{pre}\}$ and $I_{piv} = \{I^0_{piv}, I^1_{piv} \ldots I^{\tau_{max}}_{piv}\}$ where I^i_{pre} and I^i_{piv} $(0 \leq i \leq \tau_{max})$ are subsidiary indexes. I^i_{pre} is the inverted index built for the q-grams from the $((2q-1)(i-1)+1)$-th position (excluded) to the $((2q-1)i+1)$-th position in the prefix of each string in S, and I^i_{piv} is the inverted index for the $(i+1)$-th pivotal q-gram in the pivotal set for each string in S. In query processing phase, given a query string r and a threshold τ_q, the candidate set is computed as $(\bigcup_{i=0}^{i=\tau_q}(\mathsf{pre}_{(2q-1)*\tau_q+1}(r) \cap I^i_{piv})) \cap (\bigcup_{i=0}^{i=\tau_q}(\mathsf{piv}(r) \cap I^i_{pre}))$.

6 Experiments

In this section, we evaluate the performance of our proposed approach, CrossPivotalSearch. We compare it with three state-of-the-art approaches, PivotalPrefix [4], IndexGram-Turbo [15], and IndexChunk-Turbo [15], and we obtained the source code of the three approaches from the authors. All approaches are implemented in C++ and complied using g++ 4.8.2 with -o3 flag. All experiments are conducted on a machine with an Intel Quad-Core 3.20G CPU and 16 GB memory running 64bit Ubuntu. We use three real datasets in our experiments: the medical publication title dataset **Title**, a DNA sequence dataset **DNA** and a hyperlink dataset **URL**. Statistics of the datasets are shown in Table 6. For each dataset, we generate a set of query strings randomly selected from the dataset.

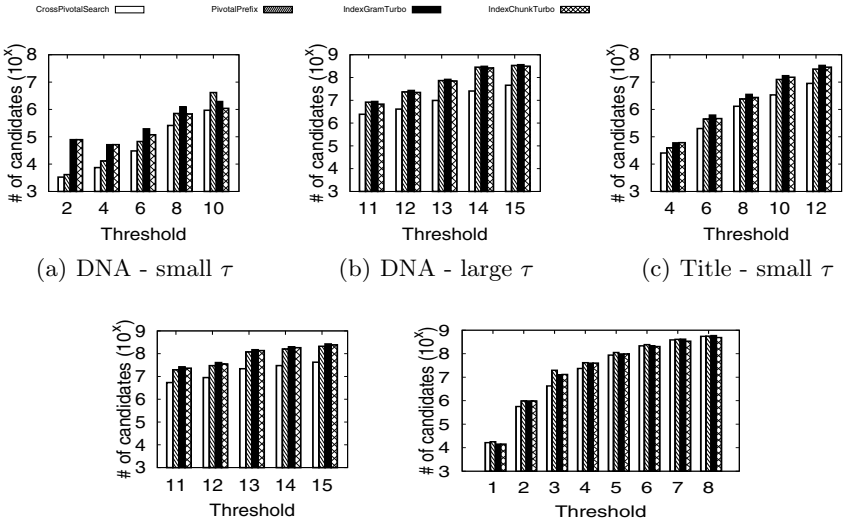
Table 6. Statistics of Datasets

Dataset	# of strings	Avg Length	Size (MB)	Query Size
DNA	2,476,276	108	269	2,305
Title	4,000,000	100.6	402	4,000
URL	1,000,000	28.03	28	1,000

In our approach, CrossPivotalSearch, we first generate candidates by using the cross pivotal filter, and then apply the position match filter and the pivotal substitution filter to further refine the candidate set; for verification, we first use the alignment filter [4] to perform the final pruning for each candidate string, and then verify the candidates with length-aware verification method [5]. The q-gram lengths are tuned for the best performance and shown as follow: for **DNA** dataset, q = 12, 12, 12, 12, 11, 11, 11, 10, 9, 9, 8, 7, 6, 6 for τ varying from 2 to 15; for **Title** dataset, q = 8,8,8,8,6,6,6,5,4,4,4,3,3,3 for τ varying from 2 to 15; and for **URL**, q = 6,3,3,2,2,2,2,2 for τ varying from 1 to 8. Thus, we have both small τ (i.e., $1 \leq \tau \leq 10$) and large τ (i.e., $10 \leq \tau \leq 15$).

6.1 Comparison with State-of-the-art Approaches

In this subsection, we evaluate the approaches by two metrics: candidate number and query processing time.



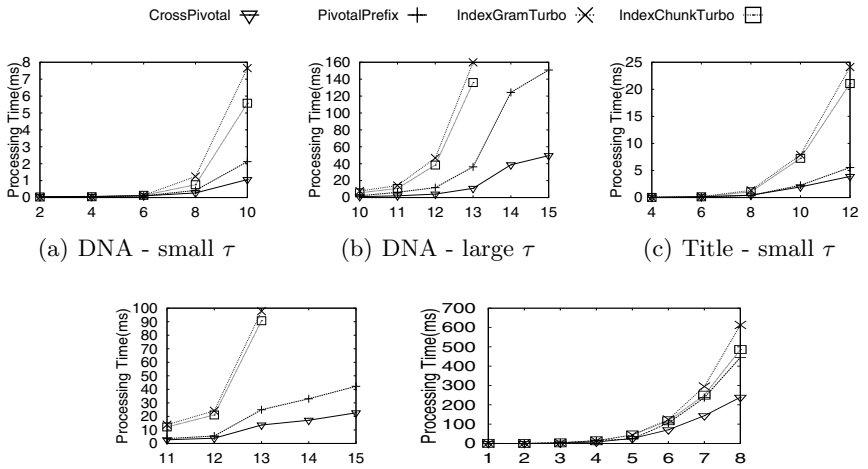
(a) DNA - small τ (b) DNA - large τ (c) Title - small τ

(d) Title - large τ (e) URL

Fig. 2. Comparison with state-of-the-art Approaches - Candidate Number

Candidate Number. The results of candidate number are shown in Figure 2. We can see that CrossPivotalSearch achieves the least candidates among the four algorithms. For example, consider Figure 2(c) for the **Title** dataset with $\tau = 10$, CrossPivotalSearch extracts 3.3 million candidates, while the candidate number for PivotalPrefix, IndexGram-Turbo and IndexChunk-Turbo are 12 millions, 17 millions and 15 millions respectively. CrossPivotalSearch has the smallest number of candidates due to the stronger pruning power of cross pivotal filter and the further pruning power of the two advanced filters. For the best case scenario, CrossPivotalSearch only extracts 13.4%, 11.9% and 14.1% of the candidates of PivotalPrefix, IndexGram-Turbo and IndexChunk-Turbo respectively for the **DNA** dataset; those numbers for the **Title** and **URL** datasets, are 18.3%, 14.7%, 15.9% and 21.5%, 33.5%, 32.4%.

Processing Time. Figure 3 shows the average query processing time for the four approaches. CrossPivotalSearch and PivotalPrefix run significantly faster than IndexGram-Turbo and IndexChunk-Turbo, with CrossPivotalSearch being the best. For instance, in **DNA** dataset with $\tau = 13$, CrossPivotalSearch only takes 10.8 milliseconds, while the average processing time for PivotalPrefix, IndexGram-Turbo and IndexChunk-Turbo are 36.3 milliseconds, 159.9 milliseconds and 135.9 milliseconds, respectively.



(a) DNA - small τ (b) DNA - large τ (c) Title - small τ

(d) Title - large τ (e) URL

Fig. 3. Comparison with state-of-the-art Approaches- Query Time

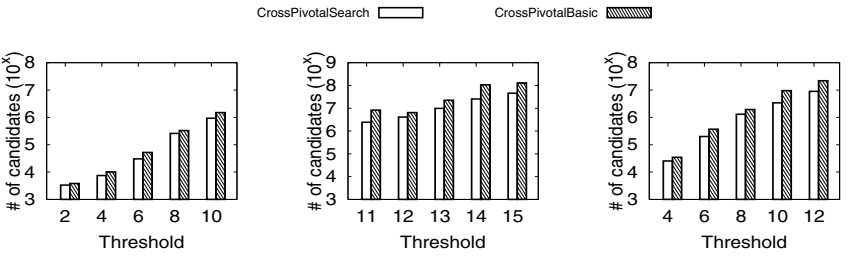
Note that, in Figure 3(b) and Figure 3(d), we omit the result for $\tau = 14$ and $\tau = 15$ for IndexGram-Turbo and IndexChunk-Turbo, due to the excessive processing time compared to that of CrossPivotalSearch and PivotalPrefix. For the best case scenario, CrossPivotalSearch only achieves 31.2%, 5.6% and 5.7% of processing time of PivotalPrefix, IndexGram-Turbo and IndexChunk-Turbo respectively for the **DNA** dataset; those numbers for the **Title** and **URL** datasets, are 51.8%, 10.2%, 10.5% and 53.7%, 38.9%, 49.2%.

We also compare the index size of the approaches and conclude that CrossPivotalSearch has the smallest index size. For example, given a small scale medical title dataset (75.5MB), with $\tau = 8$ and $q = 6$, the index size of CrossPivotalSearch is 1.4GB, while that of CrossPivotalSearch, IndexGram-Turbo and IndexChunk-Turbo are 1.9GB, 3.1GB and 2.3GB, respectively.

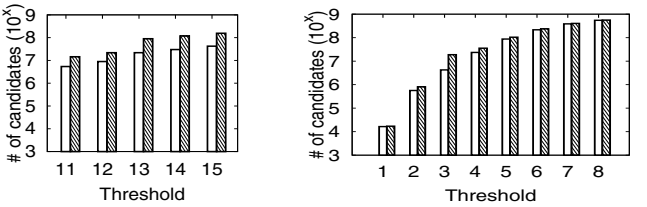
6.2 Evaluation on Advanced Filters

In this subsection, we evaluate the two advanced filters: position match filter and pivotal substitution filter. We compare CrossPivotalSearch with CrossPivotalBasic, which is the same as CrossPivotalSearch except without the two advanced filters.

Figures 4 and 5 depict the candidate numbers and the average query processing time of the two approaches. We can observe that CrossPivotalSearch only has 50% to 80% of the candidates of CrossPivotalBasic, and consequently CrossPivotalSearch runs faster than CrossPivotalBasic. This is due to that single-match candidates account for a large portion of the candidates obtained by the cross pivotal filter, while the two advanced filters can prune out a large amount of single-match candidates.

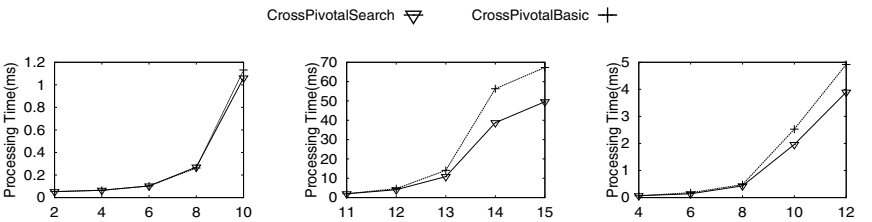


(a) DNA - small τ (b) DNA - large τ (c) Title - small τ

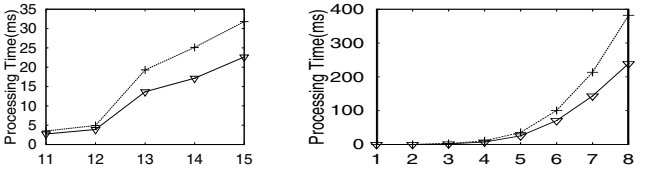


(d) Title - large τ (e) URL

Fig. 4. Evaluation of Advanced Filters - Candidate



(a) DNA - small τ (b) DNA - large τ (c) Title - small τ



(d) Title - large τ (e) URL

Fig. 5. Evaluation of Advanced Filters - Query Time

7 Conclusion

In this paper, we studied the problem of string similarity search with edit distance constraint. We proposed an efficient cross pivotal based approach, and proved its strongest pruning power compared with the state-of-the-art approaches. We then devised two advanced filters, position match filter and pivotal substitution filter

to further reduce the number of candidates. Finally, we compared our cross pivotal based approach with other three state-of-the-art pivotal based approaches by performing a comprehensive experimental study, and empirical evaluations on real datasets demonstrate the superiority of our approach in terms of both candidate number and query processing time.

Acknowledgments. Lijun Chang is supported by ARC DE150100563. Wenjie Zhang is supported by ARC DE120102144, DP120104168, ARC DP150103071 and DP150102728. Xuemin Lin is supported by NSFC61232006, ARC DP120104168, ARC DP140103578, and ARC DP150102728.

References

1. Arasu, A., Ganti, V., Kaushik, R.: Efficient exact set-similarity joins. In: VLDB (2006)
2. Chaudhuri, S., Ganjam, K., Ganti, V., Motwani, R.: Robust and efficient fuzzy match for online data cleaning. In: SIGMOD (2003)
3. Chaudhuri, S., Ganti, V., Kaushik, R.: A primitive operator for similarity joins in data cleaning. In: ICDE (2006)
4. Deng, D., Li, G., Feng, J.: A pivotal prefix based filtering algorithm for string similarity search. In: SIGMOD (2014)
5. Deng, D., Li, G., Feng, J., Li, W.: Top-k string similarity search with edit-distance constraints. In: ICDE (2013)
6. Forman, G., Eshghi, K., Chiocchetti, S.: Finding similar files in large document repositories. In: SIGKDD (2005)
7. Gravano, L., Ipeirotis, P.G., Jagadish, H.V., Koudas, N., Muthukrishnan, S., Srivastava, D.: Approximate string joins in a database (almost) for free. In: VLDB (2001)
8. Hadjieleftheriou, M., Koudas, N., Srivastava, D.: Incremental maintenance of length normalized indexes for approximate string matching. In: SIGMOD (2009)
9. Kahveci, T., Singh, A.K.: Efficient index structures for string databases. In: VLDB (2001)
10. Kim, Y., Shim, K.: Efficient top-k algorithms for approximate substring matching. In: SIGMOD (2013)
11. Li, C., Lu, J., Lu, Y.: Efficient merging and filtering algorithms for approximate string searches. In: ICDE (2008)
12. Li, C., Wang, B., Yang, X.: VGRAM: improving performance of approximate queries on string collections using variable-length grams. In: VLDB (2007)
13. Navarro, G.: A guided tour to approximate string matching. ACM Comput. Surv. **33**(1) (2001)
14. Qin, J., Wang, W., Lu, Y., Xiao, C., Lin, X.: Efficient exact edit similarity query processing with the asymmetric signature scheme. In: SIGMOD (2011)
15. Qin, J., Wang, W., Xiao, C., Lu, Y., Lin, X., Wang, H.: Asymmetric signature schemes for efficient exact edit similarity query processing. ACM Trans. Database Syst. **38**(3) (2013)
16. Sokol, D., Benson, G., Tojeira, J.: Tandem repeats over the edit distance. Bioinformatics **23**(2) (2007)
17. Wang, J., Li, G., Feng, J.: Can we beat the prefix filtering?: an adaptive framework for similarity join and search. In: SIGMOD (2012)

18. Xiao, C., Wang, W., Lin, X.: Ed-join: an efficient algorithm for similarity joins with edit distance constraints. PVLDB **1**(1) (2008)
19. Xiao, C., Wang, W., Lin, X., Yu, J.X.: Efficient similarity joins for near duplicate detection. In: WWW (2008)
20. Yang, X., Wang, B., Li, C.: Cost-based variable-length-gram selection for string collections to support approximate queries efficiently. In: SIGMOD (2008)
21. Zhang, Z., Hadjieleftheriou, M., Ooi, B.C., Srivastava, D.: Bed-tree: an all-purpose index structure for string similarity search based on edit distance. In: SIGMOD (2010)