Efficient and Effective KNN Sequence Search with Approximate n-grams

Xiaoli Wang[1] Xiaofeng Ding[2,3] Anthony K.H. Tung[1] Zhenjie Zhang[4]

[1]Dept. of Computer Science
National University of Singapore
{xiaoli,atung}@comp.nus.edu.sg

[2]Dept. of Computer Science
Huazhong University of Sci. & Tech.
xfding@hust.edu.cn

[3]Dept. of Computer Science
University of South Australia

[4]Advanced Digital Sciences Center
zhenjie@adsc.com.sg

ABSTRACT

In this paper, we address the problem of finding k-nearest neighbors (KNN) in sequence databases using the edit distance. Unlike most existing works using short and exact n-gram matchings together with a filter-and-refine framework for KNN sequence search, our new approach allows us to use longer but approximate n-gram matchings as a basis of KNN candidates pruning. Based on this new idea, we devise a pipeline framework over a two-level index for searching KNN in the sequence database. By coupling this framework together with several efficient filtering strategies, i.e. the frequency queue and the well-known Combined Algorithm (CA), our proposal brings various enticing advantages over existing works, including 1) huge reduction on false positive candidates to avoid large overheads on candidate verifications; 2) progressive result update and early termination; and 3) good extensibility to parallel computation. We conduct extensive experiments on three real datasets to verify the superiority of the proposed framework.

1. INTRODUCTION

Given a query sequence, the goal of KNN sequence search is to find k sequences in the database that are most similar to the query sequence. KNN search on sequences have applications in a variety of areas including DNA/protein sequence search [13], approximate keyword search [1], and plagiarism detection [17, 30].

Our study here is also motivated by the real application on ebook social annotation systems. In our systems, a large number of paragraphs are annotated and associated with comments and discussions[1]. For those who own a physical copy of the book, our aim is to allow them to retrieve these annotations into their mobile devices using query by snapping. As shown in Figure 1, queries are generated by users when they use mobile devices to snap a photo of page in the book. The query photo is then processed by an optical character recognition (OCR) program which extracts the text from the photo as a sequence. Since the OCR program might generate errors within the sequence, we need to perform an approximate query against the paragraphs in the server to retrieve those paragraphs that had been annotated. Since the range of error in such cases is hard to determine, a k-nearest neighbor (KNN) search is naturally preferred, avoiding the need to estimate how good the results generated from the OCR are.

Figure 1: An example of the book annotation search

This paper uses edit distance to evaluate the similarity between two sequences. Edit distance is commonly used in similarity search on large sequence databases, due to its robustness to typical errors in sequences like misspelling [13]. Existing edit distance algorithms for sequence search have focused on either approximate searching (e.g., [3, 9, 10, 16, 18, 20, 23]) or KNN similarity search [21, 28, 29]. Although range query has been extensively studied, KNN search remains a challenging issue. Many efforts on answering KNN search utilize the filter-and-refine framework [21, 28, 29]. The main idea is to prune off candidates by utilizing the number of exact matches on a set of n-grams that are generated from the sequences. An n-gram is a contiguous subsequence of a particular sequence (also called q-gram). Although such approaches are effective on short sequence searches, they are less effective if there is a need to process sequences that are longer like a page of text in a book. In this paper, we further investigate the KNN search problem from the viewpoint of enhancing efficiency.

In this paper, we develop a novel search framework which uses approximate n-grams as the filtering signatures. This allows us to use longer n-grams compared to exact matches which in turn gives more accurate pruning since such matching is less likely to be random. We introduce two novel filtering techniques based on approximate n-grams by relaxing the filtering conditions. To ensure efficiency, we employ sev-

[1]`http://readpeer.com`

Proceedings of the VLDB Endowment, Vol. 7, No. 1

eral strategies. First, we use a frequency queue (f-queue) to buffer the frequency of the approximate n-grams to support candidate selection. This can help to avoid frequent candidate verification. Second, we develop a novel search strategy by employing the paradigm of the CA method [6]. By using the summation of gram edit distances as the aggregation function, the CA strategy can enhance the KNN search by avoiding access to sequences with high dissimilarity. Third, we design a pipeline framework to support simple parallel processing. These strategies are implemented over a two-level inverted index. In the upper-level index, n-grams that are derived from the sequence database are stored in an inverted file with their references to the original sequences. In the lower-level index, each distinct n-gram from the upper-level is further decomposed into smaller sub-units, and inverted lists are constructed to store the references to the upper-level grams for each sub-unit. Based on the index, the search framework has two steps.

In the first step, given a query sequence and its n-grams, similar n-grams within a range will be quickly returned using the lower-level index. In the second step, the n-grams returned from the lower level can be automatically used as the input to construct the sorted lists in the upper level. With the sorted lists, our proposed filtering strategies are employed to enhance the search procedure. Our contributions in this paper are summarized as follows:

- We introduce novel bounds for sequence edit distance based on approximate n-grams. These bounds offer new opportunities for improving pruning effectiveness in sequence matching.
- We propose a novel KNN sequence search framework using several efficient strategies. The f-queue supports our proposed filtering techniques with a sequence buffer for candidate selection. The well-known CA strategy has an excellent property of early termination for scanning the inverted lists, and the pipeline strategy can effectively make use of parallel processing to speed up our search.
- We propose a pipeline search framework based on a two-level inverted index. By adopting a carefully staged processing that starts from searching at the lower-level n-gram index to ending at the upper-level sorted list processing, we are able to find KNN for long sequences in an easily parallelizable manner.
- We conduct a series of experiments to compare our proposed filtering strategies with existing methods. The results show that our proposed filtering techniques have better pruning power, and the new filtering strategies can enhance existing filtering techniques.

The rest of this paper is organized as follows. Section 2 discusses related studies. Section 3 provides preliminary concepts and basic principles for the KNN search. Section 4 introduces the proposed filtering techniques. Section 5 illustrates several efficient strategies to support the KNN search. Section 6 presents the pipeline search framework with a two-level inverted index. We evaluate the proposed approaches with experimental results in Section 7 and conclude the paper in Section 8.

2. RELATED WORK

Similarity query based on edit distance is a well-studied problem (e.g., [12, 15, 26]). An extensive survey had been conducted very early in [13]. Early algorithms are based on online sequential search, and mainly focus on speeding up the exact sequence edit distance (SED) computation. Among them, the most efficient algorithm requires $O(|s|^2/log|s|)$ time [12] for computing the SED, and only $O(\tau|s|)$ time for testing if the SED is within some threshold τ [29]. However, online search algorithms still suffer from poor scalability in terms of string length or database size since they need a full scan on the whole database. To overcome this drawback, most recent works follow a filter-and-refine framework. Many indexing techniques have been proposed to prune off most of the sequences before verifying the exact edit distances for a small set of candidates [14]. There are three main indexing ideas: enumerating, backtracking and partitioning.

The first idea is introduced for supporting specific queries when strings are very short or the edit distance threshold is small (e.g., [2, 24]). It is clear that enumeration usually have high space complexity and is often impractical in real query systems.

The second idea is based on branch-and-bound techniques on tree index structures. In [4, 22], a trie is used to index all strings in a dictionary. With a trie, all shared prefixes in the dictionary are collapsed into a single path, so they can process them in the best order for computing the exact SEDs. Sub-trie pruning is employed to enhance the efficiency of computing the edit distance. However, building a trie for all strings is expensive in term of both time and space complexity. In [29], a B^+-tree index structure called B^{ed}-tree is proposed to support similarity queries based on edit distance. Although this index can be implemented on most modern database systems, it suffers from poor query performance since it has a very weak filtering power.

To improve filtering effectiveness, most existing works employ the third idea that splits original strings into several smaller signatures to reduce the approximate search problem to an exact signature match problem (e.g., [3, 7, 9, 10, 11, 16, 18, 20, 23, 27]). We further classify these methods based on their preprocessing methods into the *threshold-aware* approaches and the *threshold-free* approaches. The *threshold-aware* approaches have been developed mainly based on the prefix-filtering framework. Recent work in [23] performed a detailed studies of these methods [11, 16, 23] and conclude that the prefix-filtering framework can be enhanced with an adaptive framework. These methods typically work well only for a fixed similarity threshold. If the threshold is not fixed, two choices exist. First, the index has to be built online for each query with a distinct threshold. This could be time consuming and always be impractical in real systems. Second, multiple indexes are constructed offline for all possible thresholds. This choice has high space complexity especially for databases with long sequences since there can be many distinct edit distance thresholds. The *threshold-free* approaches generally employ various n-gram based signatures. The basic idea is that if two strings are similar they should share sufficient common signatures. Compared to the *threshold-aware* approaches, these methods generally have much less preprocessing time and space overhead for storing indexes. However, if we ignore the preprocessing phrase, these methods have been presented to have the

worse performance for supporting edit distance similarity search [16]. This is because they often suffer from poor filtering effectiveness through the use of loose bounds.

Although most of such approaches had been shown to be efficient for approximate searching with a predefined threshold, limited progress has been made for addressing the KNN search problem. Existing efforts utilize two kinds of index mechanisms [21, 28, 29, 5]. The first index mechanism is adapted from inverted list based index [21, 28]. The KNN search algorithm employs the same intuition by selecting candidates with sufficient number of common n-grams. The difference between them is the list merging technique. In [21], the MergeSkip algorithm is employed to reduce the inverted list processing time. A predefined threshold based algorithm is also proposed by repeating the approximate string queries multiple times to support KNN search. In [28], the basic length filtering is used to improve list processing. Another index mechanism is based on the tree structure [29, 5]. In [29], a B^+-tree based index is proposed to index database sequences based on some sequence orders. The tree nodes are iteratively traversed to update the lower bound of edit distance and the nodes beyond the bound are pruned. In the most recent work [5], an in-memory trie structure is used to index strings and share computations on common prefixes of strings. A range-based method is proposed by grouping the pivotal entries to avoid duplicated computations in the dynamic programming matrix when the edit distance is computed. Although such approaches are effective on the short sequence search, their performances degrade for long sequences since the length of the common prefix are relatively short for long sequences and the large number of long, single branches in the trie bring about large space and computation overhead.

3. PRELIMINARIES

Let Σ be a set of elements, e.g. a finite alphabet of characters in a string database or an infinite set of latitude and longitude in a trajectory database. We use s to denote a sequence in Σ^* of length $|s|$, $s[i]$ to denote the ith element, and $s[i, j]$ to denote a subsequence of s from the ith element to the jth element. The common notations used in the rest of the paper are summarized in Table 1. In this paper, we employ edit distance as the measure on the dissimilarity between two sequences, which is formalized as follows.

DEFINITION 1. *(Sequence Edit Distance)(SED) Given two sequences s_1 and s_2, the edit distance between them, denoted by $\lambda(s_1, s_2)$, is the minimum number of primitive edit operations (i.e., insertion, deletion, and substitution) on s_1 that is necessary for transforming s_1 into s_2.*

We focus on k-nearest neighbor (KNN) search based on the edit distance, following the formal definition as below.

PROBLEM 1. *Given a query sequence q and a sequence database $D = \{s_1, s_2, ..., s_{|D|}\}$, find k sequences $\{a_1, a_2, ..., a_k\}$ in D, which are more similar to q than the other sequences, that is, $\forall s_i \in D \backslash \{a_j (1 \leq j \leq k)\}$, $\lambda(s_i, q) \geq \lambda(a_j, q)$.*

3.1 KNN Sequence Search Using N-grams

In this section, we aim to introduce important concepts and principles of sequence similarity search using n-grams which is a common technique exploited in existing studies.

Table 1: Notations

Notation	Description
D	the sequence database
q	the query sequence
$\|s\|$	the length of sequence s
$s[i]$	the i^{th} element of sequence s
G_s	the n-gram set of a sequence s
$\lambda(s_1, s_2)$	the edit distance between two sequences s_1 and s_2
$\lambda(g_1, g_2)$	the edit distance between two n-grams g_1 and g_2
$\mu(s_1, s_2)$	the gram mapping distance between two sequences s_1 and s_2
ϕ	the frequency threshold value of n-grams
k	the k value for the KNN search
τ	the edit distance threshold
$\tau(t)$	the threshold value computed by the aggregation function in the CA method
$\eta(\tau, t, n)$	the number of n-grams affected by τ edit operations with gram edit distance $> t$

DEFINITION 2. *(n-gram) Given a sequence s and a positive integer n, a positional n-gram of s is a pair (i, g), where g is a subsequence of length n starting at the i^{th} element, i.e., $g = s[i, i + n - 1]$. The set $G(s, n)$ consists of all n-grams of s, obtained by sliding a window of length n over sequence s. In particular, there are $|s| - n + 1$ n-grams in $G(s, n)$.*

In this paper, we skip the positional information of the n-grams. Such a simplified 5-gram set of a sequence *introduction*, for example, is {*intro*, *ntrod*, *trodu*, *roduc*, *oduct*, *ducti*, *uctio*, *ction*}. The n-gram set is useful in edit distance similarity evaluation, based on the following observation: if a sequence s_2 could be transformed to s_1 by τ primitive edit operations, s_1 and s_2 must share at least $\phi = (\max\{|s_1|, |s_2|\} - n + 1) - n \times \tau$ common n-grams [18].

Algorithm 1 A Simple KNN Sequence Search Algorithm

```
Require: The n-gram lists L_G for q, and k
 1: Initialize a max-heap H using first visited k sequences;
 2: for L_i ∈ L_G do
 3:    for all unprocessed s_j ∈ L_i do
 4:       frequency[s_j] + +;
 5:       τ = max{λ_s | s ∈ H};
 6:       φ = max{|s_j|, |q|} − n + 1 − n × τ;
 7:       if frequency[s_j] ≥ φ then
 8:          Compute the edit distance λ(s_j, q);
 9:          if λ(s_j, q) < τ then
10:             Update and maintain the max-heap H;
11:          Mark s_j as a processed sequence;
12: Output the k sequences in H;
```

Inverted indexes on the n-grams of the sequences are commonly used, such that references to original locations of the same n-gram are kept in a list structure. Algorithm 1 shows a typical threshold-based algorithm using the inverted index on the n-grams as well as an auxiliary heap structure. This algorithm dynamically updates the frequency threshold using the maximum edit distance maintained in a max-heap H (lines 6 - 7). The query performance depends on the efficiencies of two operations, the inverted list scan and the edit

distance computation for the candidate verification (lines 3 - 11).

Algorithm 1 could be improved by using optimization strategies, such as *length filtering* [28] and *MergeSkip* [21]. The intuition behind *length filtering* is as follow: if two sequences are within an edit distance of τ, their length difference is no larger than τ. Therefore, the inverted list scan is restricted to the sequences within the length constraint. Inverted lists are thus sorted in ascending order of the sequence length. On the other hand, the *MergeSkip* strategy preprocesses inverted lists such that the references are sorted in ascending order of the sequence identification number. When the maximum entry in the max-heap H is updated, it is used to compute a new frequency threshold ϕ, and those unprocessed sequences with frequencies less than ϕ are skipped. As an example, in Figure 2, sequence no. 10 is first visited and pushed to the top-1 heap. The temporal frequency threshold is computed as $\phi = 3$, and the candidate for next visit is sequence no. 35. In this way, sequences 20 and 30 are skipped as their frequencies are less than 3.

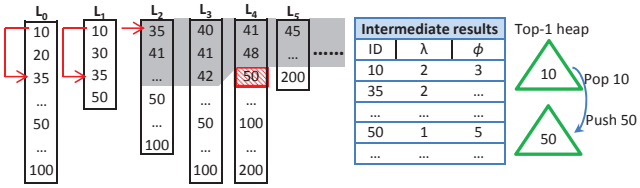


Figure 2: Illustration of the MergeSkip strategy

Although such approaches may somehow improve the efficiency of list processing, they may have limited performance since they are strictly relying on the efficient processing of inverted lists. For example, the length filtering can be useless in a database where most sequences are around the same length. In Figure 2, the top-1 heap is updated when sequence no. 50 is visited, the new frequency threshold is $\phi = 5$, and the next visiting candidate is sequence no. 45. In this case, no sequence may be skipped. The reason is that sequences from 35 to 45 are located in the grey area may have been processed as the frequencies of their matched n-grams are larger than 3. As the frequency threshold is a loose bound that can generate too many false positives, the candidate verification becomes the most time consuming step.

4. NEW FILTERING THEORY

Due to the limited pruning effectiveness of exact n-gram matching, we aim to develop new theories for sequence search filtering by using approximate matching between n-grams of the two sequences. This is motivated by the observations that using exact n-gram matching will typically require n to be small (so that the probability of having exact matching will not be too low) which will in turn lower the selectivity of the n-grams. By allowing approximate matching for these n-grams, we can increase the size of n without compromising the chance of a matching taking place, thereby increasing selectivity of the n-grams and reducing the length of the inverted list to be scanned. As shown in Definition 2, an n-gram is a subsequence of the original sequence. Consequently, gram edit distance is computed as the sequence edit distance between two n-grams.

Count filtering is the first pruning strategy we design based on gram edit distance, It is an extension of the existing count filtering on exact n-gram matchings. Basically, we want to estimate the maximal number of n-grams modified by τ edit operations such that the gram edit distance between the affected n-gram and the queried n-gram is larger than a certain value of t ($t \geq 0$). This leads to the new count filtering using approximate n-grams, as is shown in the following proposition.

PROPOSITION 1. *Consider two sequences s_1 and s_2. If s_1 and s_2 are within an edit distance of τ, then s_1 and s_2 must have at most $\eta(\tau, t, n) = \max\{1, n - 2 \times t\} + (n - t)(\tau - 1)$ n-grams with gram edit distance $> t$, where $t < n$.*

PROOF. Let $t = 0$. Then $\eta(\tau, 0, n) = \max\{1, n - 2 \times 0\} + (n - 0) \times (\tau - 1) = n \times \tau$. Intuitively, this holds because one edit operation can modify at most n n-grams. Consequently, τ edit operations can modify at most $n \times \tau$ n-grams (i.e., there are at most $n \times \tau$ n-grams between s_1 and s_2 with gram edit distance > 0).

Let $t \geq 1$. We first analyze the effect of edit operations on the n-grams with certain gram edit distance (GED). We show the first edit operation in two cases: it is applied on the first or last n-1 n-grams, and it is applied into other positions not within the first or last n-1 n-grams. As shown in Figure 3, in Case 1, one edit operation is applied in the position in the pink box. Two types of edit operations will affect n-grams to have different distance distributions. Obviously, one substitution will cause n n-grams to have GED $= 1$; while one insertion or deletion will cause one new n-gram and n-1 n-grams of various GEDs. Consequently, the upper bound value of $\eta(\tau, t, n)$ will cause at least 1 n-gram with GED $= n$. We now show the distribution of the GEDs. As shown in the figure, two 5-grams g_1 and g_5 have GED $= 1$ in Figure 3(a). However, two 5-grams g_2 and g_4 can have GED ≤ 2. Generally, one such operation can cause at most $n - 2 \times t$ n-grams to have GED $> t$. Remember that there are at least one new derived n-gram of GED $= n$. Therefore, an upper bound on the number of affected n-grams with GED $> t$ should be $\max\{1, n - 2 \times t\}$. In case 2, one edit operation is applied to the first or last n-1 n-grams. The total number of affected n-grams, denoted by n', is less than n, and the number of affected n-grams have GED $> t$ should be less than that of Case 1. It is obvious that the number of affected n-grams in Case 2 is less than that in Case 1. It is indeed true that Case 1 can infer an upper bound value on the affected n-gram number when the insertion or deletion operation is applied.

We now show how the distribution of edit operations will affect the maximum number of n-grams with GED $> t$. Suppose $E = \{e_1, e_2, \ldots, e_\tau\}$ is a series of edit operations that is needed to transform one sequence into another sequence. Suppose the τ edit operations are evenly distributed in a sequence. That means no n-gram is simultaneously affected by multiple edit operations. In this case, the number of affected n-grams can be maximized . As analyzed above, one edit operation will affect at most $n - 2 \times t$ n-grams to have GED $> t$. This is the boundary case where the edit operation is the first or the last operation. It is clear that the number of affected n-grams with GED $> t$, on the left of the first edit operation and on the right of the last edit operation, is at most $\max\{1, n - 2 \times t\}$. For the remaining $\tau - 1$ edit operations, one new operation will cause $n - t$ newly affected

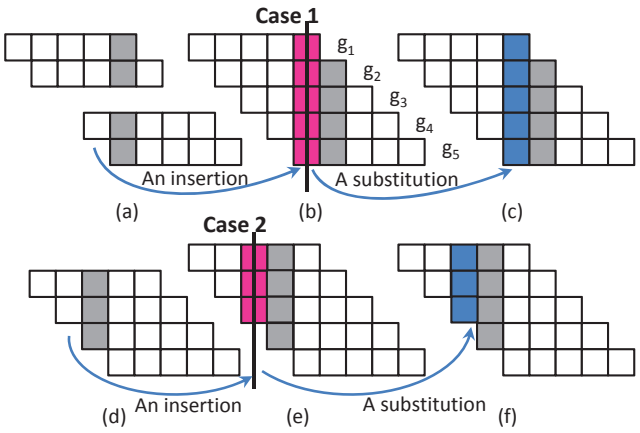


Figure 3: Effect of edit operations on n-grams

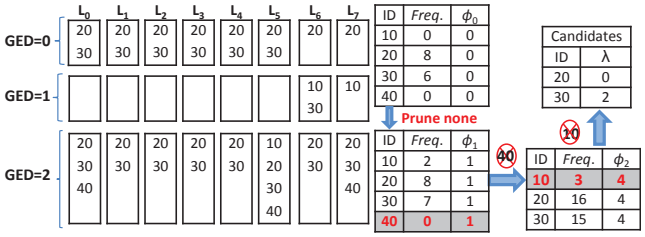


Figure 4: An example of the count filtering

n-grams ahead its previous edit position, as the boundary position will be affected only in this case. Consequently, the maximum number of affected n-grams with GED $> t$ would be $\eta(\tau, t, n) = \max\{1, n - 2 \times t\} + (n - t)(\tau - 1)$. □

LEMMA 1. *Consider two sequences s_1 and s_2. If s_1 and s_2 are within an edit distance of τ, then s_1 and s_2 must share at least $\phi_t(s_1, s_2) = |s| - n + 1 - \eta(\tau, t, n)$ n-grams with gram edit distance $\leq t$. Here, $|s|$ is equal to $\max\{|s_1|, |s_2|\}$.*

The proposed count filtering offers new opportunities to improve the search performance as it has a stronger filtering ability. As is shown in Figure 4, no sequence is pruned using the count filtering with common n-grams of $\phi_0 = 0$. By using the count filtering with n-grams of $GED = 1$, sequence no. 40 can be pruned by ϕ_1 as its frequency (i.e., $Freq.$) of n-grams with $GED \leq 1$ is less than ϕ_1. Similarly, the sequence 10 is pruned by using the count filtering of ϕ_2.

Mapping filtering is a more complicated pruning strategy, but provides more effective pruning based on the gram edit distance. To begin with, we first define the distance between two multi-sets of n-grams.

DEFINITION 3. *(Gram Mapping Distance)(GMD) Given two gram multi-sets G_{s_1} and G_{s_2} of s_1 and s_2, respectively with the same cardinality. The mapping distance between s_1 and s_2 is defined as the sum of distances of the optimal mapping between their gram multi-sets, and is computed as*

$$\mu(s_1, s_2) = \min_{P} \sum_{g_i \in G_{s_1}} \lambda(g_i, P(g_i)), \ P : G_{s_1} \rightarrow G_{s_2}$$

The computation of gram mapping distance is accomplished by finding an optimal mapping between two grams multi-sets. Similar to the work in [25], we can construct a weighted matrix for each pair of grams from two sequences, and apply the Hungarian algorithm [8, 19]. Based on gram mapping distance, we show how a tighter lower bound on the edit distance between two sequences could be achieved.

LEMMA 2. *Given two sequences s_1 and s_2. The gram mapping distance $\mu(s_1, s_2)$ between s_1 and s_2 satisfies*

$$\mu(s_1, s_2) \leq (3n - 2) \times \lambda(s_1, s_2)$$

PROOF. Let $E = \{e_1, e_2, \ldots, e_K\}$ be a series of edit operations that is needed to transform s_1 into s_2. Accordingly, there is a set of sequences $s_1 = M_0 \rightarrow M_1 \rightarrow \ldots \rightarrow M_\tau = s_2$, where $M_{i-1} \rightarrow M_i$ indicates that M_i is the derived sequence from M_{i-1} by performing e_i for $1 \leq i \leq K$. Assume there are K_1 insertion operations, K_2 deletion operations and K_3 substitution operations, then we have $K = K_1 + K_2 + K_3$. We analyze the detailed influence of each type of edit operation as follows.

Insertion operation: When a character is inserted into the sequence M_{i-1}, at most n n-grams are affected. The edit distance is less than 2 for $(n - 1)$ n-grams , and n for one newly inserted n-gram. Thus, we conclude that $\mu(M_{i-1}, M_i) \leq [2(n - 1) + n] = 3n - 2$.

Deletion operation: When one character is deleted from the sequence M_{i-1}, thus a total number of n n-grams may be affected. The edit distance is less than 2 for $(n-1)$ n-grams, and n for one newly deleted n-gram. Thus, in the case of deleting one character, $\mu(M_{i-1}, M_i) \leq [2(n-1)+n] = 3n-2$.

Substitution operation: When a character in sequence M_{i-1} is substituted by another character, a total number of n n-grams are affected. Then, the edit distance for each affected n-gram is equal to 1, and thus we have $\mu(M_{i-1}, M_i) \leq n$.

By analyzing the effect of the above three operations, we conclude that GMD and SED have the following relationship.

$$\begin{aligned} \mu(s_1, s_2) &\leq (3n - 2) \times K_1 + (3n - 2) \times K_2 + n \times K_3 \\ &\leq (3n - 2) \times (K_1 + K_2 + K_3) \\ &\leq (3n - 2) \times \lambda(s_1, s_2) \end{aligned}$$

□

Lemma 2 naturally brings us a new lower bound estimation method on the sequence edit distance. Given two sequences s_1 and s_2, and an edit distance threshold τ, if $\frac{\mu(s_1, s_2)}{3n-2} > \tau$, then $\lambda(s_1, s_2) > \tau$. While the bound is effective, it remains computational expensive if we directly apply this bound for pre-pruning. In this work, we employ this bound function to compute the aggregation value in the CA filtering algorithms. That is, we use the summation of gram edit distances as the aggregation function, instead of directly computing the mapping distance. We will introduce new implementing filtering strategies and algorithmic frameworks to make these theories practical.

5. FILTERING ALGORITHMS

Based on the filtering theories derived in the previous section, we introduce new algorithms to support efficient filtering. Given a query sequence q, we assume that there are existing inverted lists that support efficient search on the n-grams under specified edit distance constraint, as shown in Figure 5 and 6 with $L_G = \{L_0, L_1, ..., L_{|q|-n}\}$.

Algorithm 2 KNN Search Algorithm Using the F-queue

Require: N-gram lists L_G with $GED = t$ for q, and k'
Initialize the f-queue as $\emptyset$;
Initialize a max-heap H using first visited k sequences;
for $L_i \in L_G$ **do**
  for all unprocessed $s_j \in L_i$ **do**
    $frequency[s_j] + +$;
    Update the top-k' f-queue;
    if k' items in f-queue **then**
      for all $s_c \in top - k'$ **do**
        $\tau = max\{\lambda_s | s \in H\}$;
        $\eta(\tau, t, n) = \max\{1, n - 2 \times t\} + (n - t)(\tau - 1)$;
        $\phi_t = max\{|s_c|, |q|\} - n + 1 - \eta(\tau, t, n)$;
        if $frequency[s_c] \geq \phi_t$ **then**
          Compute the edit distance $\lambda(s_c, q)$;
          Mark s_c as a processed sequence;
          if $\lambda(s_c, q) < \tau$ **then**
            Update and maintain the max-heap H;
Output the k sequences in H;

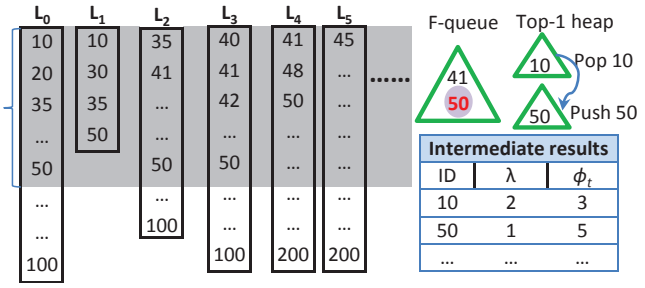


Figure 5: Illustration of the frequency queue

Algorithm 3 KNN Search Algorithm Using the CA Method

Require: N-gram lists L_G with $GED = t$ for q, and h
Initialize the f-queue as $\emptyset$;
Initialize a max-heap H using first visited k sequences;
Initialize $\underline{t_i} = t$ with $i = 0 \ldots |q| - n$;
for $L_i \in L_G$ **do**
  for all unprocessed $s_j \in L_i$ **do**
    $frequency[s_j] + +$;
    Update the top-k' f-queue;
    if The end of L_i is visited **then**
      $\underline{t_i} = t + 1$;
      if $t > 0$ **then**
        $\tau = max\{\lambda_s | s \in H\}$;
        $\tau(t) = \sum_{l=0}^{l=|q|-n} \underline{t_l}$;
        if $\tau(t) \geq \tau \times (3n - 2)$ **then**
          Terminate the list processing;
    if k' items in f-queue **then**
      Apply the filtering strategy with the f-heap;
Output the k sequences in H;

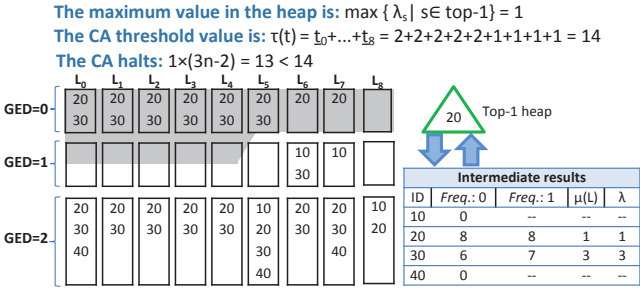


Figure 6: An example of CA based filtering

We use a frequency queue (f-queue) to speed up our inverted list processing. The basic intuition is that sequences that share higher number of matched approximate n-grams with the query sequence will be given preference for processing. Here, the f-queue is an unordered queue, which maintains the top-k' unique visited sequences with frequency of approximately matched n-grams larger than a temporary frequency threshold.

The f-queue is first initialized to be an empty set, and we perform access to the inverted lists to count the frequency of approximately matched n-grams. As shown in Algorithm 2, if the queue contains k' unprocessed sequences, our algorithm first sorts the visited sequences in ascending order based on the frequencies. Subsequently, we verify the sequences in the f-queue using the temporary frequency threshold (highest frequency first). Those sequences passing the count filtering are verified with the exact edit distance computation, and used to update the top-k heap. Note that the temporary frequency threshold is immediately updated when a new value is inserted into the top-k heap. Generally, the f-queue technique avoids frequent verifications on the visited sequences. It offers new opportunities to employ count filtering with approximate n-grams, as only the top-k' sequences are processed for every batch of h elements that are retrieved from the inverted list ($h \geq k'$). The f-queue can be used to improve the performance of existing algorithms based on the length filtering or the MergeSkip strategy.

Figure 5 illustrates the idea of the f-queue and explains why it improves the *MergeSkip* strategy. The top-1 heap is initialized using the first sequence which is visited i.e. sequence no. 10 and the frequency threshold is set as 3. Here, we set the value of k' as 2. After scanning the lists in the gray box, two unprocessed sequences no. 41 and 50 are pushed into the f-queue since the frequency of their approximately matched n-grams are higher than the temporary frequency threshold. The f-queue is then traversed for candidate verification, and sequence no. 50 is verified first since it possesses a highest number of matching n-grams. As its edit distance is computed as 1, a new threshold is computed for the top-1 matching and the top-1 heap is updated accordingly, by discarding the sequence no. 10 and pushing sequence no. 50. Finally, the new frequency threshold is 5, based on our update rule. Compared against the standard method in Figure 2, our approach can successfully skip the sequences from 35 to 45.

As the novel count filtering can be applied without any constraint on the gram edit distance, the list processing with the f-queue can continue until all the sequences are processed. However, applying more count filters will mean that more list processing time is required. To avoid having large overhead on list processing, we use the CA based strategy [6] and use the summation of gram edit distances as the aggregation function. As inverted lists of certain GED are fetched out separately, they are naturally sorted by the gram edit distance (lowest distance first). Algorithm 3 shows the details of the CA based filtering algorithm. We use Example 1 to illustrate the effectiveness of the CA-based filtering framework for supporting the KNN search.

EXAMPLE 1. *In Figure 6, we consider the nine sorted lists for a query sequence q. The length of the n-gram is set to be $n = 5$. Each list has three groups of n-grams with various GEDs of 0, 1, and 2. Each entry in the lists stores the sequence id number. Suppose we perform sorted access to each sorted list L_i. For each list L_i, let $\underline{t_i}$ be the GED score of the last sequence visited under sorted access. The CA threshold value is computed as $\tau(t) = \sum_{i=0}^{i=8} \underline{t_i}$. As soon as $\tau(t) \geq max\{\lambda_s | s \in top-k\} \times (3n-2)$, CA halts and we stop scanning the inverted lists. In the figure, CA halts at the positions at the bottom of the grey area. In this case, each $\underline{t_i}$ in the group of GED=1 is initialized to be 1. When we do sorted access to the list L_4, each value of $\underline{t_i}$ with $i = 0, 1, .., 4$ is set to be equal to 2 as no entry has distance of 1, and 2 is the smallest score that can be obtained for unseen elements. Therefore, $\tau(t)$ is computed as $\sum_{i=0}^{i=8} \underline{t_i} = 2+2+2+2+2+1+1+1+1 = 14$. Consequently, the CA halts as $\tau(t) \geq max\{\lambda_s | s \in top-1\} \times (3n-2) = 1 \times 13$. In this case, the unseen sequences 10 and 40 are safely pruned.*

The correctness of CA based strategy is easily to be shown according to Lemma 2. When CA halts, any unseen sequence has a mapping distance $\mu(s_{unseen}, q)$ that is not less than $\tau(t)$. Then, we have $(3n-2) \times \lambda(s_{unseen}, q) \geq \mu(s_{unseen}, q) \geq \tau(t) \geq max\{\lambda_s | s \in top-k\} \times (3n-2)$. Consequently, any unseen sequence has an edit distance that is not less than the maximum edit distance in the top-k heap, i.e., $\lambda(s_{unseen}, q) \geq max\{\lambda_s | s \in top-k\}$. The sequences in the top-k heap are the required KNN results.

6. INDEXING AND QUERY PROCESSING

In the previous section, we developed our algorithms based on the assumption that we can find the approximately matched n-grams efficiently and is thus able to access the corresponding inverted list of these approximately matched n-grams. In this section, we will explain how this can be done on top of a two-levels inverted index. Based on this two-levels index, we will develop a pipeline framework to support efficient KNN sequence search.

We build a two-levels inverted index based on n-grams with different granularity of n_1 and n_2 respectively. As shown in Figure 7, the index consists of the upper-level index and the lower-level index. In particular, given a sequence database D, the upper-level is used to index the n_1-grams that are obtained from the original sequences in D, and the lower-level is used to index the n_2-grams that are obtained from n_1-grams in the upper-level ($n_1 > n_2$).

There are two steps to build the index: 1) we extract n_1-grams from sequences in D, and build the upper-level inverted index. The index is made up of two main components: an index for all distinct n_1-grams and an inverted list below each n_1-gram. In general, each entry in the inverted lists contains the sequence identifier. 2) we further extract n_2-grams from all distinct n_1-grams, and build the lower-level inverted index. Similarly, the index consists of two main parts, which stores the n_2-grams with the reference to the corresponding n_1-gram in the inverted lists. Generally, the inverted list entries in the upper-level index are usually sorted into various orders when using different filtering techniques. For example, they are sorted into order of increasing sequence identifier for the MergeSkip strategy and increasing sequence length for the length filtering.

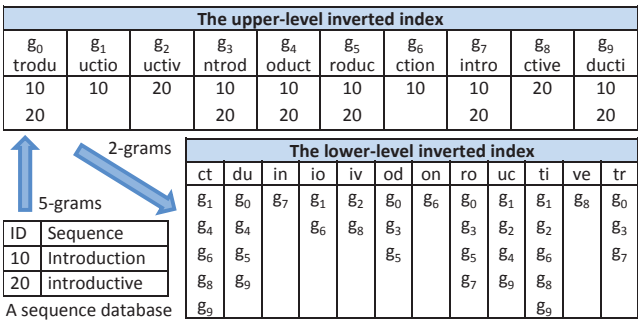


Figure 7: An example of the two-level index

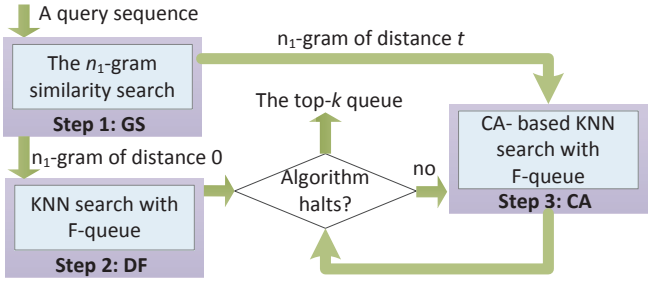


Figure 8: The simple serial query processing flow

This paper will investigate the effect of the list order on the proposed techniques in the experimental study.

6.1 A Simple Serial Solution

We first introduce a simple serial solution using the proposed two-level inverted index. Our approach follows a filter-and-refine framework. Given a query sequence q, it is first decomposed into a set of n_1-grams G_q. As shown in Figure 8, the serial algorithm works as follows.

1. **GS**: For each $g_i \in G_q$, the lower-level index is used to support the sequence similarity search to return n_1-grams with GED $\leq t$ to g_i. The returned list pf n_1-grams are naturally grouped based on the GED distance of $0, 1, ..., t$.

2. **DF**: Given the output from the GS step, we fetch out the inverted lists from the upper-level index for those matching n_1-grams of distance 0. The list merging algorithm with the proposed f-queue technique is employed to support fast frequency aggregation and maintain the top-k queue for processed sequences (See Algorithm 2).

3. **CA**: If the DF step does not halt the algorithm, we further fetch the inverted lists from the upper-level index for those similar n_1-grams of distance t ($t \geq 1$). Given the f-queue and the top-k queue from the output of the DF step, the list merging algorithm will continue to accumulate frequencies of similar n_1-grams, and use the proposed count filtering bound for further pruning. Noted that, the CA filtering bound will be employed if a new gram edit distance appears (See Algorithm 3).

In the GS step, we employ the fastest approximate string matching algorithm to support efficient n_1-gram similarity

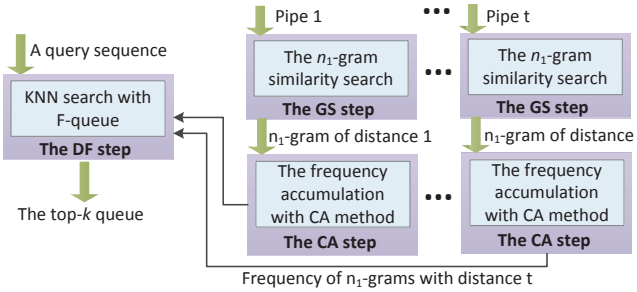


Figure 9: The pipelined query processing flow

search. In our implementation, we use the traditional count filtering with exact n-gram matching to do pre-pruning. The quality of the approximate string matching is sufficiently high to return similar n_1-grams with negligible query time.

We employ the proposed f-queue technique in the DF step and adopt a CA based algorithm to do further pruning in the CA step if the previous processing cannot terminate our search. The algorithm halts under the following conditions: 1) All sequences in the database have gone through the candidate verification; 2) The maximum value in the root of the top-k queue is within an acceptable small distance. Generally, if the temporary frequency threshold is equal to 0, those unseen sequences cannot be safely pruned and need post processing. The algorithm will request for further list processing by relaxing the distance threshold for n_1-gram if these conditions are not meet.

In general, the filter-and-refine framework can be evaluated using the cost model with $T_q = T_f + |C_q| \times T_r$, where T_f is the filtering time and T_r is the verification time for each candidate. Our solution attempts to reduce the number of false positives, as it is costly to verify the candidates for long sequences. Compared with existing n-gram based methods, our approach offers new opportunities to speed up the query processing in the CA step by using our novel count filtering.

6.2 A Novel Pipeline Framework

Next, we propose a dynamic method that is easy to be pipelined to enhance query processing. The main idea is that similar n_1-grams are dynamically returned from the lower-level index for pruning in the upper-level index.

As shown in Figure 9, we adapt the CA step to perform the frequency accumulation without processing sequences using the f-queue. We develop a pipelined algorithm to execute the GS step and the CA step. In this way, the DF step can process visited sequences in the f-queue by using the temporary frequency thresholds that is computed from the approximate n_1-grams, instead of only using the frequency bound of common n_1-grams. The pipelined execution offers new opportunities for reducing the overhead costs of employing multiple filtering techniques.

6.3 The Pipelined KNN Search

To support efficient KNN search, the three stages are implemented differently in the pipeline framework as described below.

The GS stage are shown in Algorithm 4. The algorithm takes the n_1-gram set G_q obtained from a query q as the input. Given a constraint on the maximum value of gram

Algorithm 4 The GS Stage

Require: The n_1-gram set G_q, a constraint of t_{max}
1: **loop**
2: Update the GED threshold value of t;
3: **if** The halting signal is detected **then**
4: Terminate this stage;
5: **else**
6: **for** $L_i \in G_q$ **do**
7: Apply $gramSimilaritySearch(g_i, t)$;
8: Pipe similar n_1-grams with GED=t to the CA stage;

Algorithm 5 The CA Stage

Require: The global top-k heap H
1: **loop**
2: Update the GED threshold value of t;
3: **if** The halting signal is detected **then**
4: Terminate this stage;
5: Obtain inverted lists L_G from the upper-level index for all n_1-grams;
6: **for** $L_i \in L_G$ **do**
7: **for all** unprocessed $s_j \in L_i$ **do**
8: $frequency[s_j] + +$;
9: **if** The end of L_i is visited **then**
10: $\underline{t_i} = t + 1$;
11: **if** $t > 0$ **then**
12: $\tau = max\{\lambda_s | s \in H\}$;
13: $\tau(t) = \sum_{l=0}^{l=|q|-n} \underline{t_l}$;
14: **if** $\tau(t) \geq \tau \times (3n - 2)$ **then**
15: All unseen strings are safely pruned;
16: Send a global halting signal;
17: Terminate this stage;

edit distance (GED) value, this stage performs the similarity search to return n_1-grams with GED=t to each $g_i \in G_q$. The output of the query results will be fed into the CA stage. In the CA stage, given the n_1-gram set with GED=t, the inverted lists are fetched from the upper-level index, and scanned to accumulate the frequencies for each visited sequence. The CA strategy is used to terminate the whole process if the CA threshold value of the gram edit distance summation is larger than the temporary threshold computed from the top-k heap. It is convenient to compute the new CA threshold value $\tau(t)$ with a summation of the gram edit distance returned from the GS stage. For example, if the GED for all returned grams is equal to t, then we have $\tau(t) = t \times |G_q|$. However, this value is updated when a new distance value appears. As shown in Line 9 and Line 12 in Algorithm 5, CA can enhance the total query processing by avoiding access to those very dissimilar strings. If the halting condition with the CA aggregation value has been found, this stage immediately stops and sends a global signal to invoke the termination of the whole search. The details of the CA stage are shown in Algorithm 5.

In the DF stage, we maintain a global max-heap H for storing the current $top - k$ similar sequences and use the maximum edit distance score in the root of the heap to update the temporary frequency value. As shown in Algorithm 6 (lines 13 - 15), the distance value of the top element in the $top - k$ heap is selected as a new range bound for the CA stage and the DF stage.

Algorithm 6 The DF stage

Require: A query sequence q
Require: The global top-k heap H
Initialize the f-queue as $\emptyset$;
Initialize a max-heap H using first visited k sequences;
Obtain the n_1-gram set G_q from q
Obtain N-gram lists L_G for all $g_i \in G_q$
for $L_i \in L_G$ **do**
  for all unprocessed $s_j \in L_i$ **do**
    if The halting signal is detected **then**
      Terminate this stage;
    $frequency[0][s_j] + +$;
    Update the top-k' f-queue;
    if k' items in f-queue **then**
      for all $s_c \in top - k'$ **do**
        $\tau = max\{\lambda_s | s \in H\}$;
        $\eta(\tau, t, n) = \max\{1, n - 2 \times t\} + (n - t)(\tau - 1)$;
        $\phi_t = max\{|s_c|, |q|\} - n + 1 - \eta(\tau, t, n)$;
        for all $l = 0 \dots t$ **do**
          $frequency(s_c)_l = \sum_{m=0}^{m=l} frequency[m][s_c]$;
        if all $frequency(s_c)_l \geq \phi_l$ with $l = 0 \dots t$ **then**
          Compute the edit distance $\lambda(s_c, q)$;
          Mark s_c as a processed sequence;
          if $\lambda(s_c, q) < \tau$ **then**
            Update and maintain the max-heap H;
Output the k sequences in H;

7. EXPERIMENTAL STUDY

We compare the performance of our proposed approach **AppGram** against several state-of-the-art methods over a wide spectrum of real datasets.

7.1 Setup

The algorithms used in the following experiments are presented as below.

- **B^{ed}-tree** [29] is proposed to support the string similarity queries using a B^+-tree based index structure. We use the implementation from the authors.
- **Flamingo** [9] is an open-source data cleaning system which supports approximate string search. We use the latest release 4.1. For existing work [21] and [28], we use the implementation from the Flamingo, as it has integrated the previous filtering techniques.
- **TopkSearch** [5] is the most recent method that is proposed to support top-k sequences similarity search with edit-distance constraints. We obtain the executable binary file from the authors.

We use two real datasets that are available publicly. They cover different domains and are widely used in previous studies. We use another paragraph dataset obtained from an ebook reading system. The details of the datasets are shown as follows, and the statistics are shown in Table 2.

- **IMDB** consists of movie titles which are taken from a public database of IMDB[2], and we use the dataset provided in paper [29].

[2]http://www.imdb.com

Table 2: Datasets

Dataset	Size	Avg. Len	Max. Len
IMDB	1,553,914	19	240
DBLP	1,385,668	105	1626
ANNOTEXT	1,572,561	75	1250

Table 3: Parameter Settings

Parameter	Description	Value
k	k value for the search	$1, 2, \mathbf{4}, 6, 8, 10$
$\|q\|$	average query size	$10, 20, 30, 40, 50$
τ	distance threshold	$1, 2, 4, 8, 10, 16$

- **DBLP** consists author names and titles of publications which are extracted from the DBLP Bibliography[3], and we use the dataset provided in paper [23].
- **ANNOTEXT** is a dataset containing annotated paragraphs. We extract the paragraphs from a paper abstract collection that are taken from a public citation database[4]. The annotated paragraphs are generated with a random sampling method. We maintain the length distribution of this dataset to be the same as the DBLP dataset.

The query files of the first two public datasets are also available in their original work. Each query file includes 100 sequences, and we obtain them from the authors together with the datasets. For the ANNOTEXT dataset, we select 100 queries by random sampling. Table 3 presents major parameters used in our experiments, including their descriptions and values (with default values in bold). Hereafter, the default values will be used in all the experiments unless otherwise stated.

AppGram was implemented in C++, and the pipeline algorithm is implemented using pthread. In all the experiments, we only implement two threads to support two pipelines. We compiled all the algorithms with gcc 4.4.6 in Red hat Linux Operating System, and all experiments were done on a server with Quad-Core AMD Opteron(tm) Processor 8356, 128GB memory, running RHEL 4.7AS.

7.2 Construction Time and Index Size

Table 4 and 5 show the construction time and the index size on the three datasets. The n-gram length is set at n=5 for all datasets. As the selection of n value has been comprehensively investigated in existing n-gram based methods, we skip the results here. In AppGram, the gram length for the lower level is set at 3. As shown in the figure, the B^{ed}-tree takes less time and smaller space than AppGram and Flamingo. Since the AppGram decomposes the sequences into n-grams without any prefix and suffix, it takes slightly smaller space than the Flamingo. The AppGram takes slightly more construction time than the Flamingo, as it needs to build the lower-level index for the n-grams in the upper level. As the binary file for the TopkSearch algorithm does not provide the preprocessing time and space costs, we exclude the results on this method.

To evaluate the space and time overhead on the extra lower-level index, we present the percentage of total index

[3]http://www.informatik.uni-trier.de/∼ley/db
[4]http://arnetminer.org/citation

Table 4: Construction time (sec)

	AppGram	B^{ed}-tree	Flamingo
IMDB	13	57	25.59
DBLP	154.3	35	116.22
ANNOTEXT	89.3	45	64.12

Table 5: Index size (MB)

	AppGram	B^{ed}-tree	Flamingo
IMDB	108	63.2	159
DBLP	563	222.9	608
ANNOTEXT	444	183.7	492

cost on three datasets in Figure 10. Obviously, the lower-level index has a very small ratio compared with the upper-level index. As shown in the figure, it represents 11.1% of the index size over the IMDB dataset, and no more than 5% of the index size over the DBLP and ANNOTEXT datasets. This small cost indicates that the overhead for the lower-level processing with queries may be negligible compared to the total time cost. We will present more results in the following subsections.

7.3 Quality of Count Filtering

We evaluate the quality of the proposed count filtering technique. The AppGram-0 is the proposed algorithm which employs the common count bound with n-grams of distance 0; while the AppGram-1 is one which uses not only the common count bound but also the count filter with approximate n-grams of distance 1. A query file of 10000 sequences are randomly sampled from each dataset by reserving the original distribution. We vary the edit distance threshold τ as 1, 2, 4, 8, 10, and 16. We accumulate number of sequences which are pruned. Figure 11 shows the average number of sequences that are filtered with respect to the edit distance threshold on the three datasets. As shown in the figure, the AppGram-1 can filter out more sequences than the AppGram-0, which means that the proposed count filtering based on approximate n-grams have better filtering power than the existing common count bound.

Generally, the filtering power of the AppGram-1 prunes about 10% more sequences compared with the AppGram-0, and the improvement becomes more significant when the edit distance threshold becomes larger. As shown in Figure 11(c), when the edit distance threshold becomes 8 and 16, the AppGram-1 can prune $20K$ sequences compared to $2K$ sequences that are pruned by the AppGram-0. The difference even becomes larger if we further increase the edit distance threshold value.

7.4 Effect of Various Filters

We evaluated various ways to integrate our proposed filters with existing techniques on the three datasets. We use the default query file containing 100 sequences. For each query, we execute the KNN search with various k values from 1 to 10, and count the number of accessed elements in the list processing and the generated candidate size.

Figure 12 shows the average number of sequences on the query inverted lists for various filter combinations. As shown in the figure, using only the MergeSkip technique to support dynamic count filtering will access too many sequence ids for

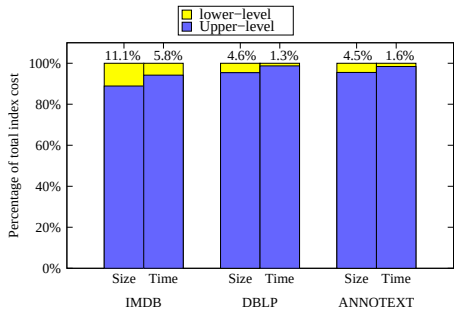


Figure 10: Percentage of the index cost

processing the inverted lists. The length filtering technique is useful in reducing the number of entries that are accessed for list processing. We also design an algorithm to combine our proposed technique with the MergeSkip technique and the length filter. It is obvious that, the combined technique can significantly reduce the number of accessed entries in the processed lists.

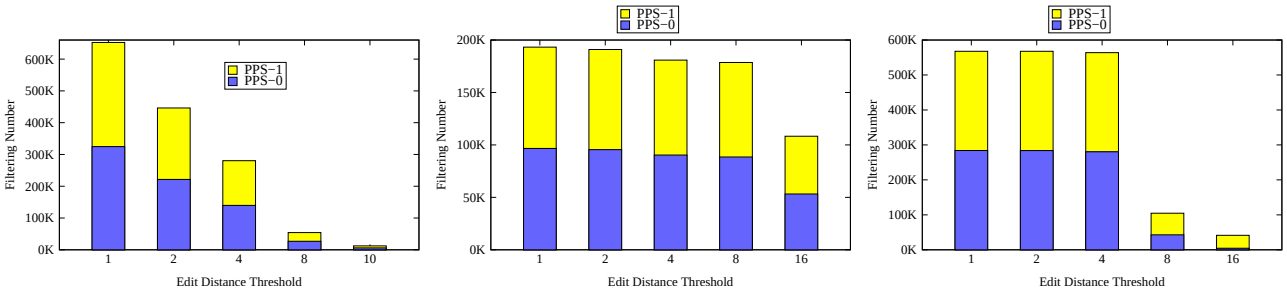
Figure 13 shows the candidate size for each filter combination. The results indicate that combining the MergeSkip with the length filter can help to reduce the candidate size and improve the query performance. Our proposed filter can further reduce the candidate size. The proposed filtering technique enhances the query performance of the MergeSkip technique and the length filter, as this filter combination needs access to the smallest number of entries in the invert lists and generates the smallest number of candidates.

7.5 Query Evaluation

We evaluate the query performance of the proposed approach compared to exiting methods of B^{ed}-tree, Flamingo, and TopkSearch. The KNN search are conducted as follows. We vary k over 1, 2, 4, 8, and 16. For each k, we execute the 100 queries, and compute the average of the query results.

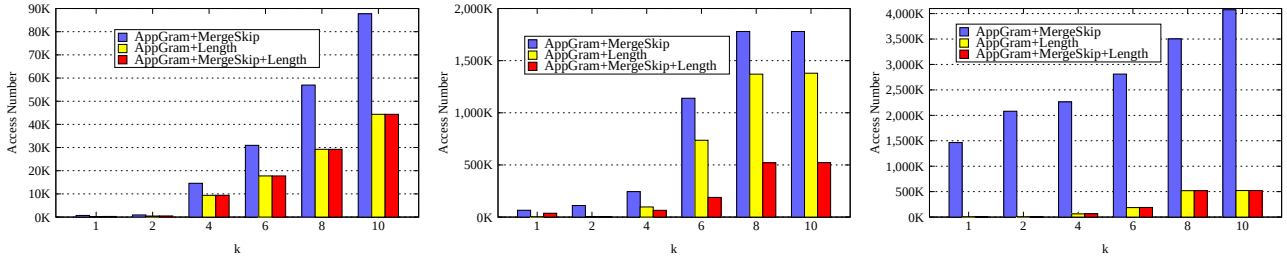
Figure 14 shows the average query time with respect to k on three datasets. It can be seen that the AppGram method outperforms all the competitive techniques with $k \geq 2$. The proposed f-queue and CA based filtering strategies can reduce the cost of the candidate verification. When the k value is as small as 1, Flamingo can run efficiently as it only needs to execute a range query once to obtain the top-1 result. In this case, a small edit distance threshold of 1 may be enough to report the top-1 sequence. Similarly, the TopkSearch is also more efficient for k=1 as its range-based algorithm only needs to verify a small number of entries in the dynamic matrix. However, when the k value increases, our AppGram method indeed outperforms other algorithms, and has a stable average query time. Note that TopkSearch takes too long to run on the long sequences of DBLP and the ANNOTEXT datasets and its results are excluded.

Figure 15 compares the overhead of two query phases: the filtering time and the candidate verification time. We randomly select five groups of queries with various average lengths of 10, 20, 30, 40, and 50, and each group has 100 query sequences. For each group, we execute the KNN search, and compute the average filtering time and verification time. It is clear that candidate verification is the most consuming step in the KNN search. That means that a tighter bound is required for speeding up KNN search and it is reasonable to sacrifice slightly higher filtering overhead to reduce the candidate sequences as much as possible.



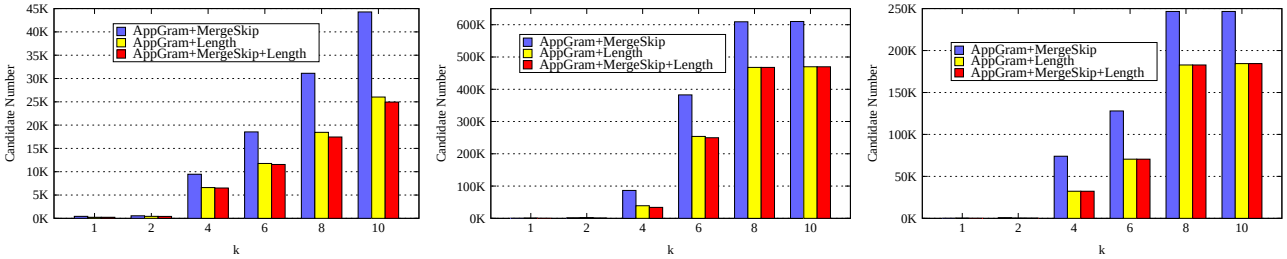
(a) IMDB (b) DBLP (c) ANNOTEXT

Figure 11: Average filtering number vs. τ



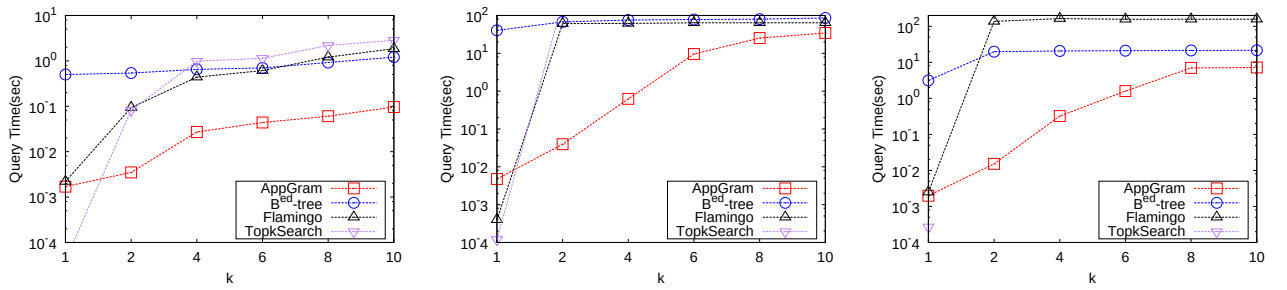
(a) IMDB (b) DBLP (c) ANNOTEXT

Figure 12: Average accessed number of sequences on lists vs. k



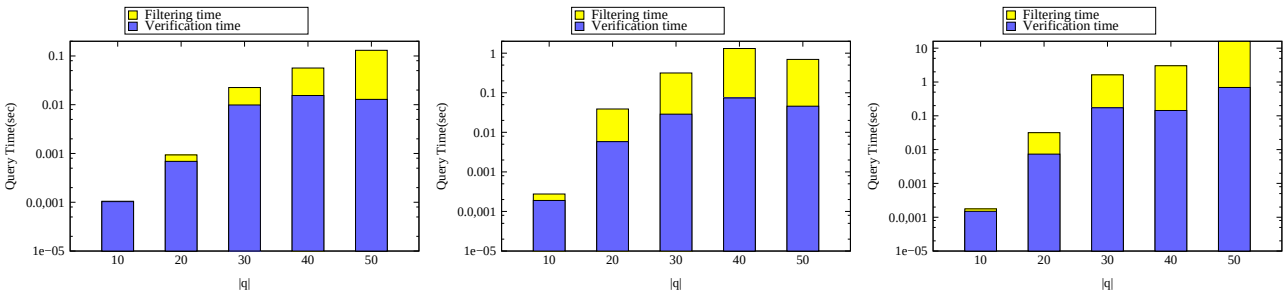
(a) IMDB (b) DBLP (c) ANNOTEXT

Figure 13: Average candidate size vs. k



(a) IMDB (b) DBLP (c) ANNOTEXT

Figure 14: Average query time vs. k



(a) IMDB (b) DBLP (c) ANNOTEXT

Figure 15: Detailed analysis on the query cost vs. $|q|$

8. CONCLUSION

In this paper, we study the problem of k-nearest neighbor sequence search based on edit distance. While existing approaches often suffer from poor filtering power and low query performance when sequences in the database are long, we tackle the problem by designing a novel file-and-refine pipeline approach utilizing approximate n-gram matchings. In the filtering phase, we develop a novel filtering technique based on counting the number of approximate n-grams. Experimental results show that this technique provides strong pruning on the unqualified candidates. We also propose an efficient searching algorithm with the frequency queue and the CA strategy. The frequency queue supports our proposed filtering techniques by reducing the number of candidate verification, while CA supports early termination to stop unnecessary computation of the whole pipeline framework. As a conclusion, our proposed filtering strategies show excellent performance on the KNN search, and the pipeline framework is easy to extend to parallel computation.

Acknowledgment

Wang was supported by the Singapore NRF under its IRC@SG Funding Initiative and administered by the IDMPO, at the SeSaMe Centre. Tung was supported under the Campus for Research Excellence And Technological Enterprise (CREATE) programme. Ding was supported by NSFC China grant 61100060. Zhang was partly supported by Human Sixth Sense Project by A*STAR at Advanced Digital Sciences Center.

9. REFERENCES

[1] S. Alsubaiee, A. Behm, and C. Li. Supporting location-based approximate-keyword queries. In *ACM GIS*, pages 61–70, 2010.

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

[3] X. Cao, S. C. Li, and A. K. H. Tung. Indexing dna sequences using q-grams. In *DASFAA*, pages 4–16, 2005.

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

[5] D. Dong, L. Guoliang, F. Jianhua, and L. Wen-Syan. Top-k string similarity search with edit-distance constraints. In *ICDE*, 2013.

[6] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, pages 102–113, 2001.

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

[8] H. W. Kugn. The hungarian method for the assignment problem. *Naval Research Logistics*, 2:83–97, 1955.

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

[10] C. Li, B. Wang, and X. Yang. Vgram: improving performance of approximate queries on string collections using variable-length grams. In *VLDB*, pages 303–314, 2007.

[11] G. Li, D. Deng, J. Wang, and J. Feng. Pass-join: a partition-based method for similarity joins. *PVLDB*, 5(3):253–264, 2011.

[12] W. J. Masek and M. Paterson. A faster algorithm computing string edit distance. *JCSS*, 20(1):18–31, 1980.

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

[14] G. Navarro, R. A. Baeza-Yates, E. Sutinen, and J. Tarhio. Indexing methods for approximate string matching. *IEEE Data Eng. Bull.*, 24(4):19–27, 2001.

[15] R. Prasad and S. Agarwal. Study of bit-parallel approximate parameterized string matching algorithms. In *CCIS*, pages 26–36, 2009.

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

[17] Z. Su, B.-R. Ahn, K.-Y. Eom, M.-K. Kang, J.-P. Kim, and M.-K. Kim. Plagiarism detection using the levenshtein distance and smith-waterman algorithm. In *ICICIC*, 2008.

[18] E. Sutinen and J. Tarhio. Filtration with q-samples in approximate string matching. In *CPM*, pages 50–63, 1996.

[19] I. H. Toroslu and G. íçoluk. Incremental assignment problem. *Inf. Sci.*, 177(6):1523–1529, 2007.

[20] Ukkonen and Esko. Approximate string-matching with q-grams and maximal matches. *Theor. Comput. Sci.*, 92:191–211, 1992.

[21] R. Vernica and C. Li. Efficient top-k algorithms for fuzzy search in string collections. In *KEYS*, pages 9–14, 2009.

[22] J. Wang, J. Feng, and G. Li. Trie-join: efficient trie-based string similarity joins with edit-distance constraints. *PVLDB*, 3(1-2):1219–1230, 2010.

[23] J. Wang, G. Li, and J. Feng. Can we beat the prefix filtering? an adaptive framework for similarity join and search. In *SIGMOD*, 2012.

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

[25] X. Wang, X. Ding, A. K. H. Tung, S. Ying, and H. Jin. An efficient graph indexing method. In *ICDE*, 2012.

[26] S. Wu, U. Manber, and E. W. Myers. A subquadratic algorithm for approximate limited expression matching. *Algorithmica*, 15(1):50–67, 1996.

[27] X. Yang, B. Wang, and C. Li. Cost-based variable-length-gram selection for string collections to support approximate queries efficiently. In *SIGMOD*, pages 353–364, 2008.

[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*, pages 915–926, 2010.

[30] M. Zini, M. Fabbri, M. Moneglia, and A. Panunzi. Plagiarism detection through multilevel text comparison. In *AXMEDIS*, pages 181–185, 2006.