9-2009

Accelerating Sequence Searching: Dimensionality Reduction Method

Guojie SONG
Peking University

Bin CUI
Peking University

Baihua ZHENG
Singapore Management University, bhzheng@smu.edu.sg

Kunqing XIE
Peking University

Dongqing YANG
Peking University

DOI: https://doi.org/10.1007/s10115-008-0180-0

Accelerating Sequence Searching: Dimensionality Reduction Method*

Guojie Song [1], Bin Cui[2], Baihua Zheng[3],Kunqing Xie[1] ,Dongqing Yang[2]

[1] Key Laboratory of Machine Perception(Peking University),Ministry of Education, China
{gjsong@pku.edu.cn,kunqing@cis.pku.edu.cn}

[2] Department of Computer Science, Peking University, Beijing, China
{bin.cui@pku.edu.cn,dqyang@pku.edu.cn}

[3] School of Information System, Singapore Management University, Singapore
{bhzheng@smu.edu.sg}

Abstract

Similarity search over long sequence dataset becomes increasingly popular in many emerging applications, such as text retrieval, genetic sequences exploring, etc. In this paper, a novel index structure, namely *Sequence Embedding Multiset tree*(*SEM-tree*), has been proposed to speed up the searching process over long sequences. The SEM-tree is a multi-level structure where each level represents the sequence data with different compression level of multiset, and the length of multiset increases towards the leaf level which contains original sequences. The multisets, obtained using sequence embedding algorithms, have the desirable property that they do not need to keep the character order in the sequence, i.e. shorter representation, but can reserve the majority of distance information of sequences. Each level of the tree serves to prune the search space more efficiently as the multisets utilize the predicability to finish the searching process beforehand and reduce the computational cost greatly. A set of comprehensive experiments are conducted to evaluate the performance of the SEM-tree, and the experimental results show that the proposed method is much more efficient than existing representative methods.

Keywords: Sequence similarity search, Sequence embedding, Index, Dimension reduction

1 Introduction

Indexing technologies, which were originally proposed to speed up data search performance, have been successfully applied in many areas of data processing. Some of the index structures have even been used in commercial database systems, e.g. B^+-tree [34]. However, none of the existing

*Supported by the National Natural Science Foundation of China under Grant No. 60703066, and the National High-Tech Research and Development Plan of China (863) under Grant No.2006AA12Z217.

indexes is universally efficient since the performance of an index structure is highly dependent on the underlying data.

Similarity search is an important common operation for many applications, such as text retrieval, handwriting recognition, and multimedia search. Recently, similarity search over long sequences attracts more attentions due to some new emerging applications. As illustrated in [9], in computational biology, searching for specific sequences over DNA and protein sequences appears as a fundamental operation for problems such as assembling the DNA chain from the pieces obtained by the experiments, looking for given features in DNA chains and determining how different two genetic sequences were. In such applications, the problem of sequence similarity search is typically based on block edit distance with move[7]. Block edit distance with move is the minimum cost transformation from one sequence to another through a series of edit operations (insert, delete, move) on block characters. It is another kind of distance metric except for character edit distance, and is widely used in computational biology and text processing environments. Due to the great length of sequences which can be up to thousands, similarity search is an expensive operation.

Many index structures have been proposed to handle similar sequence search problems, such as suffix trees and vector space indexing. Suffix tree based index structures [12] have been gaining favor as the methods for sequence search, but they consume too much memory space. Classical indexing techniques can be used for sequence search, such as R*-trees [11], X-trees [15] and SR-trees [25] etc. They typically perform well in low to medium dimensional spaces (up to $20 - 30$ dimensions), but their performance deteriorates drastically for long sequence match. Differently, embedding based index techniques, such as FastMap [3], have also been proposed to decrease computational cost. However, it has to scan the original sequence dataset in order to construct embedded space. Since the distance computation in the original string space is very expensive, the construction of the embedded space is impractical. Furthermore, the approximate factor keeps increasing as the data size expands, rendering this approach unworkable for large sequence databases.

In this paper, we proposed a novel index structure, namely *SEM-tree*, to facilitate efficient similarity sequence search. Our strategy is to replace the expensive full sequence similarity computation with the comparison over shorter multisets, which are embedded from sequences. Thereafter, as a compressed representation of sequence, multiset enables much faster distance computation. Moreover, the distance comparison is dependent on the number of distinct characters that form the sequence set, instead of the length of the sequence. As a sequence set usually has a finite set of distinct characters, it can be assumed that the computation is bounded by a constant. Another interesting characteristic is its predicability, which means that query processing in an interval node can predict whether the sub-tree is included in final results, so as to stop the recursion process beforehand. Lastly, although the distance over multisets is an approximation of the real distance between original sequences, we can exploit the boundary of the approximations [7] to avoid false negative. The main contribution of this work is four-fold.

- To reduce the computational cost for long sequences′ comparison, we adopted sequence embedding algorithm to convert the sequence to a multiset which has compressed representation. The distance computation among original sequences can be approached on multisets with high approximate factor but in linear time.
- Just as the well-known PCA[14] and SVD[8] methods, sequence oriented dimensionality reduction method with predicability, named **SDR**, has been developed based on sequence embedding. To the best of our knowledge, this is the first method using embedded sequence for sequence search.
- A novel index structure, SEM-tree, is proposed by using SDR techniques developed to efficiently shrink the search space and hence speed up the search process.
- A comprehensive simulation has been conducted to evaluate the performance of SEM-tree. As experimental results indicate, search over proposed index structure shows superior performance over other approaches in terms of efficiency.

A preliminary version of this paper appears in [10], where we presented the basic idea. In this paper, we make the following additional contributions. First, we provide more detailed description of the framework, including tree construction and query processing. Second, we present the details of proof and analysis for applying SDB techniques in our work. Furthermore, we run a more comprehensive set of experiments to demonstrate the effectiveness of the SEM-tree using different datasets.

The paper is organized as follows. In Section 2, we introduce related works. In section 3, we review the basic sequence embedding technique and design the sequence dimension reduction mechanism based on multiset, followed by the structure of SEM-tree and detailed algorithms. Experimental results are presented in Section 4. Finally, we conclude this paper in Section 5.

2 Related Work

In this section, we first briefly review related work, and then analyze the limitations of the existing data structures.

Many heuristic-based search methods have been developed to conduct sequence search. They fall into two categories: hash-table based methods and suffix-tree-based methods. Some of the important hash-table-based methods are FASTA[26] and BLAST[37]. These techniques are similar in spirit: they construct a hash table on one of the strings, and insert all substrings of a certain length l. The tools start by finding exactly matching substrings (known as seeds) of length l using this hash table. In the second phase, the seeds are extended in both directions, and combined, if possible, in order to find better alignments. Current hash-table-based search tools handle short queries well, but become very inefficient, in terms of both time and space, for long queries. Suffix trees were first proposed by Weiner[27] under the name position tree. Later, efficient suffix tree construction methods[6] and variations[21] were developed. However, there

are two significant problems with the suffix-tree approach: (1) suffix trees manage mismatches inefficiently, and (2) they are notorious for their excessive memory usage[11]. The size of the suffix tree varies from 10 to 37 bytes per letter[15]. Some indexing techniques and dimension reduction methods have also been proposed for dealing with time-series dataset[16, 17, 32, 19, 24], however they are not much suitable for sequence data comprised by character set.

Vector space based indexing is another kind of method. The VP-tree[25] partitions the data space into spherical cuts by selecting random reference points from the data. A second method, the MVP-Tree[20] (a variation of VP-Tree) uses more than one vantage point at each level. Reference[31]uses the VP-Tree to index measures that are almost metric. In the recently proposed reference-based sequence indexing methods [28, 32, 15], reference sequences are selected from the convex hull of the dataset, which is done by selecting sequences that are far away from each other. All these methods are based on the original sequence space and are still inefficient for high dimensional data.

Embedding based index techniques, such as FastMap [3] and MetricMap [35], have also been proposed to decrease the computational cost. However, both approaches ask for a distance-preserving mapping function. Finding a suitable mapping is a tough and time-consuming process, and no such function is available for block edit distance. Additionally, it has to scan the original sequence space in order to construct embedded space, making the construction of the embedded space impractical in our case.

Recently proposed indexing techniques, such as M-trees [23], Slim-tree[22] and DBM-tree[2], can be used to support sequence search. The M-tree is a height-balanced tree, where the data elements are stored in leaf nodes. The Slim-Tree is an evolution of the M-Tree, embodying the first published technique to reduce the amount of overlap between tree nodes, which leads to a smaller number of disk accesses to answer similarity queries. These two structures are height balanced and attempt to reduce the height of the tree at the expense of flexibility in reducing overlap between nodes. This constraint was released in the DBM-Tree by reducing the overlap between nodes in high density regions, resulting in an unbalanced tree.

3 The SEM-Tree

Handling similarity search over long sequence has always been a challenge to the database research community because of the heavy computational cost. In this section, A new **S**equence **D**imensionality **R**eduction method, named **SDR**, is proposed based on sequence embedding for the construction of SEM-tree. Thereafter, the new index structure of SEM-tree is presented to facilitate sequence similarity search on long sequence database.

3.1 SDR: Sequence Dimensionality Reduction

In this section, we first introduce an existing sequence embedding technique [7]. Thereafter, Sequence Dimensionality Reduction (SDR) mechanism is developed to reduce the computational

cost for sequence match, which is the basis of the proposed SEM-tree.

3.1.1 Review of Sequence Embedding

Embedding strategy helps to reduce the cost of expensive distance computation of long sequences by transforming longer sequences to shorter multisets. Here, we use an example to illustrate the embedding process proposed in [7], as shown in Figure 1. The original sequence s, also denoted as $ET_1(s)$ (the sequence before the first embedding iteration), contains *cabagehcadbba* and it is partitioned into six blocks after first iteration, with each block having 2 or 3 elements. Each block $ET_1(s)[c_s, c_e]$ (c_s and c_e are the start and end character in this block respectively)is thereafter represented by an element based on a hashing function $h(ET(s)_1[*, *])$ and all these new elements form $ET_2(s)$ (the sequence before the second iteration). For example, the first block of $ET_1(s)[ca]$ is hashed to an element k. h is a one-to-one Karp-Miller hash function[29] on sequences of length at most 3, therefore the same sequence must correspond to the same sub-tree in $ET(s)$. The partition and hashing continues until at one level the sequence only contains one element. The time complexity of the whole process is $O(|s| \log^* |s|)$[1].

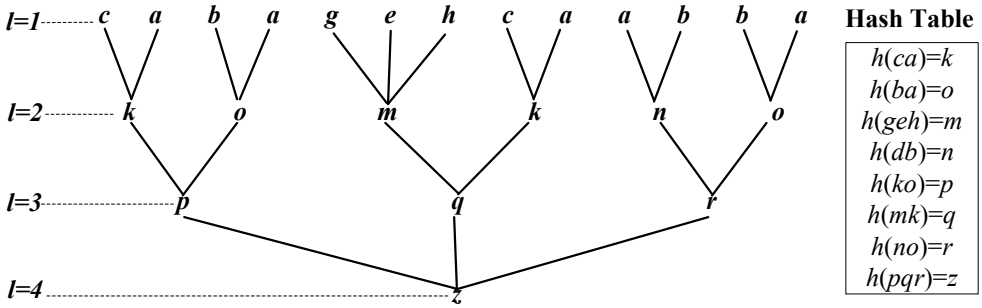


Figure 1: $ET(s)$ ($s = cabagehcadbba$)

Given an $ET(s)$, all the elements within the entire tree can be represented by a family of multisets, denoted as $T(s)$. $ET_i(s)$ is used to represent the sequence at the i-th level, and the set of the elements within $ET_i(s)$ is denoted by the sub-multiset $T_i(s)$. Via Example 1, it is easy to notice that $ET_i(s)$ and $T_i(s)$ contain the same set of elements, while the order is important in the former but irrelevant in the latter.

Based on above sequence embedding method, the multiset and sequence shown in Figure 1

[1]This formula for time complexity was first introduced in [33]. Given an integer k, $\log^* k$ denotes, $min\{i \geq 0: log^{(i)} k \leq 1\}$, where $log^{(i)} k = log(log^{(i-l)} k)$, and $log^{(0)} x = 0$. In practice, $log^* n$ is never more than 5.

can be divided as follows[2]:

$$\begin{aligned} T(s) &= \{T_1(s), T_2(s), T_3(s), T_4(s)\} \\ &= \{\{a_4, b_3, c_2, d_1, e_1, g_1, h_1\}, \{k_2, m_1, n_1, o_2\}, \\ &\quad \{p_1, q_1, r_1\}, \{z_1\}\} \\ ET(s) &= \{ET_1(s), ET_2(s), ET_3(s), ET_4(s)\} \\ &= \{\{cabagehcadbba\}, \{komkno\}, \{pqr\}, \{z\}\} \end{aligned}$$

3.1.2 The Analysis of Sequence Dimensionality Reduction

The work of [7] only focuses on how to embed sequences into multiset space using a sequence embedding technique, while it does not consider any indexing structure and mine characteristics of (sub-)multisets. The dimensionality reduction techniques, such as PCA, have been widely used to facilitate similarity search in high-dimensional space. However, such approaches can not be applied in sequence matching domain due to the different distance evaluation. The multisets generated by embedding technique motivate us to exploit the sub-multisets for sequence dimensionality reduction.

Suppose we have two sequences, s_1 and s_2, in the sequence dataset. Let $T(s_1) = \bigcup_{i=1}^{l_1} T_i(s_1)$ and $T(s_2) = \bigcup_{j=1}^{l_2} T_i(s_2)$ denote the transformed multisets, where l_1 and l_2 are the heights of trees $ET(s_1)$ and $ET(s_2)$ respectively. Its original distance, $\hat{d}(s_1, s_2)$, is a *edit distance with moves*[7], which can only be approximated by the sequence embedding method. We develop the novel SDR mechanism with the following nice properties:

1. Since the order of the elements does not affect the multiset, the distance between two multisets is only determined by the number of the different elements σ in the multisets, where they share a common character set $\Sigma = \{a_1, a_2, ..., a_\sigma\}$.

   The sub-multisets $T^i(s)$ can be defined as the union of all the sub-multisets in the first i levels of $T(s)$, denoted as $T^j(s) = \bigcup_{k=1}^{j} T_k(s)$, thus the formal definition of distance between two sub-multisets, $T^i(s_1)$ and $T^i(s_2)$, is provided by Equation 1.

   $$d_e(T^i(s_1), T^i(s_2)) = card(T^i(s_1) \ominus T^i(s_2)) + card(T^i(s_2) \ominus T^i(s_1)) \tag{1}$$

   To facilitate the distance comparison between two sequences with different length, normalized standard distance between two multisets is defined in Equation 2. The distance computation between two multisets can be finished within $O(\sigma)$.

   $$\hat{d}_e(T^i(s_1), T^i(s_2)) = \frac{d_e(T^i(s_1), T^i(s_2))}{card(T^i(s_1) \cup T^i(s_2))} \tag{2}$$

2. Contractness property for the construction of index is satisfied with $\hat{d}_e(T^i(s_1), T^i(s_2)) \leq \hat{d}_e(T^j(s_1), T^j(s_2))\ \forall i, j$ with $i \leq j \leq max(l_1, l_2)$, which will be further proved by lemma 1 and lemma 2.

[2]In this example, a_4 means that character 'a' occurrence '4' times.

3. Predictability is another important characteristic of sequence embedding, which can quicken the query processing with such ability: *known which sub-tree(s) of index to be included in the final result beforehand when querying the internal node*, which will be proved by lemma 3. The idea behind is that if we know the distance on a sub-multiset, e.g. $d_e(T^i(s_1), T^i(s_2))$, we can estimate upper bound of the real distance between s_1 and s_2. This promising property makes it different from other dimensionality reduction methods, such as PCA. These methods can reduce the complexity of data, but they cannot quantify the distance information loss.

Next, we will present the details of proof for embedded sequence properties. The basic operations on the multiset are introduced in the Appendix, such as $\ominus$, $card()$, $\cap$ and $\cup$ etc.

LEMMA 1 *Given two sequences s_1 and s_2, $\forall i, j$ with $i \leq j \leq max(l_1, l_2)$, we have:*

$$\hat{d}_e(T_i(s_1), T_i(s_2)) \leq \hat{d}_e(T_j(s_1), T_j(s_2))$$

PROOF Suppose $r_1 = T_i(s_1) \cap T_i(s_2)$, $r_2 = T_i(s_1) \ominus T_i(s_2)$, and $r_3 = T_i(s_2) \ominus T_i(s_1)$. According to Equation 2, $\hat{d}_e(T_i(s_1), T_i(s_2))$
$= \frac{|r_2|+|r_3|}{2|r_1|+|r_2|+|r_3|}$. When $ET_i(s_1)$ and $ET_i(s_2)$ are embedded into $ET_{i+1}(s_1)$ and $ET_{i+1}(s_2)$, let us suppose r' as the common subsequence of $ET_{i+1}(s_1)$ and $ET_{i+1}(s_2)$, i.e., $r' \sqsubseteq ET_{i+1}(s_1)$ and $r' \sqsubseteq ET_{i+1}(s_2)$. Since sequence embedding adopts the hash-function which is deterministic, the common subsequence r'(such as 'k') can only be produced by the common subset r_1(such as '$\{a, c\}$'). However, sequences that share the same set of elements r_1 may not be embedded into the same sequences(except for $h(ac) = k$, we can have $h(ca) = q$), i.e., $|r'| \leq \frac{|r_1|}{k}$, with $k = \frac{|ET_i(s_1)|}{|ET_{i+1}(s_2)|}$.

Therefore, $\hat{d}_e(T_{i+1}(s_1), T_{i+1}(s_2)) = \frac{(2|r_1|+r_2|+r_3|)/k-|r'|}{(2|r_1|+r_2|+r_3|)/k} \geq \hat{d}_e(T_i(s_1), T_i(s_2))$. The Lemma is proved.

□

Based on LEMMA 1, the contractiveness of the distance between sub-multisets is further identified by LEMMA 2.

LEMMA 2 ***Contractness:*** *Given any two sequences s_1 and s_2, for any $i \leq j$ we have:*

$$\hat{d}_e(T^i(s_1), T^i(s_2)) \leq \hat{d}_e(T^j(s_1), T^j(s_2)).$$

PROOF Assume $a_i = d_e(T_i(s_1), T_i(s_2))$ and $b_i = card(T_i(s_1) \cup T_i(s_2))$, according to LEMMA 1, we have:$\frac{a_1}{b_1} \leq \frac{a_2}{b_2} \leq ... \leq \frac{a_l}{b_l}$, with $\frac{a_i}{b_i} = \hat{d}_e(T_i(s_1), T_i(s_2))$. We have following induction process:

Basic step: Based on LEMMA 1, we have $\frac{a_1}{b_1} < \frac{a_2}{b_2}$. Therefore, $\frac{a_1}{b_1} \leq \frac{a_1+a_2}{b_1+b_2} \leq \frac{a_2}{b_2}$, i.e., $\hat{d}_e(T^1(s_1), T^1(s_2)) \leq \hat{d}_e(T^2(s_1), T^2(s_2))$.

Inductive step: Suppose the Lemma is satisfied when j equals $i + 1$ ($i \geq 1$), i.e., $\frac{A_i}{B_i} \leq \frac{A_i+a_{i+1}}{B_i+b_{i+1}}$ with $A_i = \sum_{k=1}^{i} a_k$ and $B_i = \sum_{k=1}^{i} b_k$. Therefore, we have $A_i * b_{i+1} \leq B_i * a_{i+1} \Rightarrow$

$\frac{A_i}{B_i} \leq \frac{A_i+a_{i+1}}{B_i+b_{i+1}} \leq \frac{a_{i+1}}{b_{i+1}}$. Based on LEMMA 1 we have $\frac{a_{i+1}}{b_{i+1}} \leq \frac{a_{i+2}}{b_{i+2}}$. In other words, $\frac{A_i}{B_i} \leq \frac{A_i+a_{i+1}}{B_i+b_{i+1}} \leq \frac{A_i+a_{i+1}+a_{i+2}}{B_i+b_{i+1}+b_{i+2}} \leq \frac{a_{i+2}}{b_{i+2}}$, i.e, $\frac{A_i}{B_i} \leq \frac{A_{i+1}}{B_{i+1}} \leq \frac{A_{i+2}}{B_{i+2}}$.

Therefore, the Lemma is also satisfied for j equals $i+2$, and the proof is completed.

□

LEMMA 3 ***Predictability****: For any two sequences s_1 and s_2, upper bound of distance $\hat{d}_e(T^i(s_1), T^i(s_2))$, denoted as $B_u(\hat{d}_e$ $(T^i(s_1), T^i(s_2)))(>\hat{d}_e(T(s_1), T(s_2)))$, can be estimated based on sub-multiset $T^i(s_1)$ and $T^i(s_2))$ as follows:*

$$B_u(\hat{d}_e(T^i(s_1), T^i(s_2))) = \frac{card(T^i(s_1) \ominus T^i(s_2)) + v}{card(T^i(s_1) \cup T^i(s_2)) + v}$$

where $v = card(T_i(s_1) \cup T_i(s_2))$

PROOF According to the definition of distance computation in Equation 2.

$$\hat{d}_e(T^i(s_1), T^i(s_2)) = \frac{card(T^i(s_1) \ominus T^i(s_2))}{card(T^i(s_1) \cup T^i(s_2))}$$

Obviously, this distance value is bounded by $\hat{d}_e(T(s_1), T(s_2))$,

$$\hat{d}_e(T^i(s_1), T^i(s_2)) < \hat{d}_e(T(s_1), T(s_2))$$

$$= \frac{a + card(\cup_{j=i+1}^{l_1} T_j(s_1) \ominus \cup_{j=i+1}^{l_2} T_j(s_2))}{b + card((\cup_{j=i+1}^{l_1} T_j(s_1)) \cup (\cup_{j=i+1}^{l_2} T_j(s_2)))}$$

$$< \frac{a + card(\cup_{j=i+1}^{l_1} T_j(s_1) \cup (\cup_{j=i+1}^{l_2} T_j(s_2)))}{b + card((\cup_{j=i+1}^{l_1} T_j(s_1)) \cup (\cup_{j=i+1}^{l_2} T_j(s_2)))}$$

where $a = card(T^i(s_1) \ominus T^i(s_2))$ and $b = card(T^i(s_1) \cup T^i(s_2))$.

Since the scale of the union $\cup_{j=i+1}^{l_1} T_j(s_1)$ and $\cup_{j=i+1}^{l_2} T_j(s_2)$ will be smaller than that of $T_i(s_1)$ and $T_i(s_2)$, $card((\cup_{j=i+1}^{l_1} T_j(s_1))$ $\cup (\cup_{j=i+1}^{l_2} T_j(s_2)))$ can be bounded by $card(T_i(s_1) \cup T_i(s_2))$.

In other words, $B_u(\hat{d}_e(T^i(s_1), T^i(s_2)))$ can be approximated by $\frac{card(T^i(s_1) \ominus T^i(s_2))+v}{card(T^i(s_1) \cup T^i(s_2))+v}$, where $v = card(T_i(s_1) \cup T_i(s_2))$. The proof is finished.

□

3.2 The Index Structure

It has been pointed out that the distance computational cost of multiset is much cheaper than that of the full sequences as we can reduce the length by transformation. Given the fact that the distance between two multisets, $T(s_1)$ and $T(s_2)$, approximately preserves the distance between original sequences s_1 and s_2, the similarity search among sequences can be replaced by the search

over corresponding (sub-)multisets. Furthermore, multisets have some very attractive features, like contractiveness, which can facilitate the search process.

The basic idea of SEM-tree is based on the characteristic of distance preservation and contractibility. Each node stands for one cluster represented by a sub-multiset which serves as the center and the radius, and the multisets whose sequences located in this cluster form the children nodes. The tree structure based on multisets not only keeps the original clustering characteristics of sequences, but also eases the construction of index and query processing.

3.2.1 The structure of SEM-tree

Given a sequence dataset *SD*, we apply sequence embedding process on *SD* to transform the sequences into a new space, named SEM-Space. Consider a sequence $s \in SD$, we define $\mathbf{SEM}(s, l)$ to be an operator that projects sequence s on its first l-level sub-mulitiset in $ET(s)$ $(1 \leq l \leq L)$, denoted as:

$$\mathbf{SEM}(s, l) = \bigcup_{i=1}^{l} T_i(s) \quad (i.e.\ T^l(s))$$

where l is called an embedding level and L is the maximum level.

The SEM-tree is a multi-tier tree and tries to partition the space into clusters and refine the clustering process as tree grows. Figure 2 shows an example. However, the indexing keys at each level of the tree are different—nodes closer to the root use the keys with lower embedding level, i.e. shorter sub-mulitiset, and the keys at the leaves are the original sequence data. At the root node of SEM-tree, only the sub-multisets from the first embedding level of the sequences contribute to the partition. Take the sequence *cabagehcadbba* in Section 3.1.1 as an example, we may use T_1 in level 1, $T_1 \cup T_2$ (i.e. T^2) in level 2, and full sequences in leaf level. Note that, we can select different levels of sub-multiset for each level during tree construction. However, the lengths of the sub-multisets are in non-descending order from root to leaf.

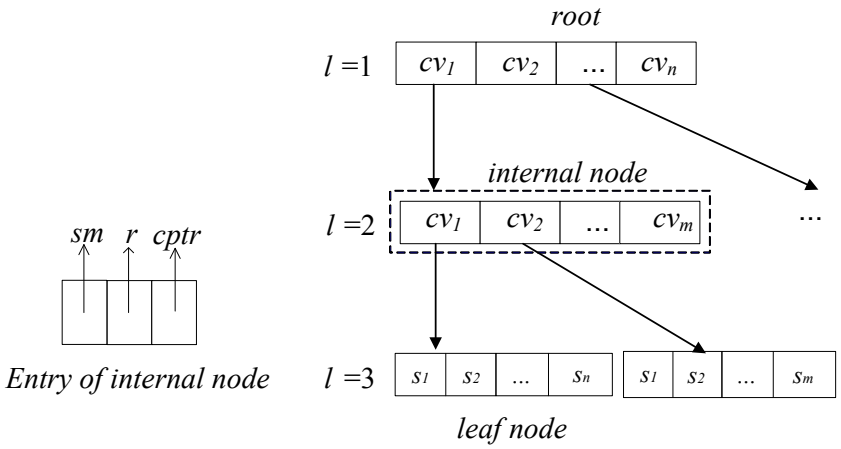


Figure 2: The Structure of SEM-tree

The entry in the internal node of SEM-tree is a vector, denoted by cluster vector *cv*, which

corresponds to a cluster at the SEM-Space. All the vectors within an internal node form a larger cluster, which refers to an entry in the parent level. We name the vectors within a certain node *vector set* (VS). Each cv is a 3-tuple vector $(sm, r, cptr)$, as shown in Figure 2. For a cv at i-th level, the sub-multiset sm represents the center of the cluster, and r is the radius of this cluster such that any sequence s in this cluster satisfies $\hat{d}_e(sm, T^m(s)) \leq r$, where m is the embedding level of multisets of the node. Pointer $cptr$ points to the child node formed by all the immediate child vectors of cv. Different from the internal node, the leaf node simply stores the original sequences within a certain cluster.

The SEM-tree can be used to prune the search space effectively. Recall (in lemma 3) that the distance between two sequences with a shorter embedding level in the embedding space is always smaller than the distance between two sequences in a longer embedding level. Thus, we can use the distance at a shorter embedding level to prune away sequences that are far away (i.e., if the distance between a database sequence and the query sequence at a shorter embedding level is already larger than the specified range query threshold, it can be pruned away). More importantly, the shorter embedding level at upper levels of the tree not only decreases the distance computational cost, but also saves storage space due to its condense characteristics.

For the SEM-tree to be effective, we need to determine the optimal number of fanout F (or the number of cv) in each internal node. The index key in the upper level of the SEM-tree utilizes a shorter sub-multiset, but we increase the embedding level of sub-multisets when traversing down the tree, till the original sequences are stored at the leaf node. In this manner, if each node corresponds to a page with a fixed page size $PageSize$ (for example $PageSize$=4KB), fan-out is large at the upper levels but small in lower level. Assume that the size of sub-multisets in the l-th level internal node is denoted by $avSize_l$, the fan-out of l-th level internal node is:

$$F_l = \lfloor \frac{PageSize}{avSize_l} \rfloor$$

3.2.2 The construction of SEM-tree

Algorithm 1 shows in detail the process of constructing a SEM-tree for a given dataset. We adopt a top down approach to get better clustering effect. At first, We use the sequence embedding routine SEM() to transform the original sequences into the SEM-Space (line 1). Then we initialize the root node (in line 2). We treat the whole sequence dataset as a cluster and refer to these new points as MS. After that, we call the recursive routine **Insert(Node, MS, l)** that essentially determines the content of the entries of the node at level l - one entry per sub-cluster. Note that we are dealing with sequences in the transformed space (i.e., MS), and that l determines the sub-multiset in MS that this node is handling.

At the beginning of **Insert(Node, MS, l)**, we check whether a leaf node can be generated. If the sequences represented by (sub-)multiset MS can fit in a disk page, then a leaf node is created (line 1-3), otherwise an internal node should be generated(in line 5). The number of clusters (or cv), F_l, is calculated using the sub-multiset length in level l in line(6-7), and then we

Algorithm 1 SEM-tree Construction

Input: SD: sequence datasets;

Output: *SEM-tree*;

Procedure:

1: MS=SEM(SD); /*Embedding SD into SEM-Space*/
2: Init($root$);
3: **Insert(**$root$**,** MS**, 1)**;

Proc Insert(Node N_d, Multiset ms, level l)

1: **if** *sizeof*(*sequences represented by ms*)$\leq$ *PageSize* **then**
2: $leaf$=New-leafNode(ms);
3: $N_d = leaf$;
4: **else**
5: N_d=New($internal\_node$);
6: $avSize_l$= *sizeof*(ms_l);/*ms_l *means the first* l *level multiset*/
7: F_l= $\lfloor \frac{PageSize}{avSize_l} \rfloor$;
8: $seqClusters$=k-means(ms, F_l);
9: **for** each $strClt_i \in seqClusters$ **do**
10: cv_i = new(cluster vector); $cv_{i-1}.cptr$=cv_i;
11: $cv_i.sm$=$strClt_i.center$; $cv_i.r$=$strClt_i.radius$;
12: **Insert(**$cv_i.cptr$**,** $strClt_i$**,** $l+1$**)**;
13: **end for**
14: **end if**

partition the data of the cluster ms into F_l sub-clusters by K-means (in line 8)[3]. However, this partitioning is performed only on the first l levels sub-multiset. For each sub-cluster, we fill the information on the center and radius into the corresponding entry in node, and then recursively invoke routine **Insert()** to build the next level of the tree(in line 9-12).

In order to illustrate the construction algorithm, a running example is provided in Figure 3. Suppose each of these sequences has already been partitioned into three sub-multisets, denoted by $T_1(s)$, $T_2(s)$, and $T_3(s)$ in table 1. In the first level, we got two clusters by using the distance computation based on the first level sub-multiset $T^1(.)$. One cluster take $T_1(s_2)$ as the center and 1/9 as the radius. The other take $T_1(s_5)$ as cluster centered with radius of 1/3. Similarly, at the second level, we follow the same principle for each sub-cluster based on the sub-multiset $T^2(.)$. In the leaf level, we index the full sequences if we assume each node can store two sequences.

We note that the SEM-tree can be made balanced by generating equal sized clusters by using the method in[4]. However, such a balanced tree has not achieved better performance, since it increases the radius of small clusters, which leads to more overlap and more page accesses. Thus, for efficiency reason, we do not require the SEM-tree to be height-balanced. Since the data may be skewed, it is possible that some clusters may be large, while others contain fewer points.

[3]Other related clustering algorithms can also be adopted here, but K-means is preferred because it can provide K clusters directly.

Table 1: Sequence Dataset

	Embedded Sub-multisets		
Sequences	$T_1(s)$	$T_2(s)$	$T_3(s)$
$s_1 = aaachakee$	$\{a_4, c_1, e_2, h_1, k_1\}$	$\{b_1, r_1, g_1\}$	$\{i_1\}$
$s_2 = aaachaeke$	$\{a_4, c_1, e_2, h_1, k_1\}$	$\{b_1, r_1, l_1\}$	$\{j_1\}$
$s_3 = aachaeke$	$\{a_3, c_1, e_2, h_1, k_1\}$	$\{x_1, r_1, l_1\}$	$\{m_1\}$
$s_4 = babanavae$	$\{a_4, b_2, e_1, n_1, v_1\}$	$\{s_1, w_1, z_1\}$	$\{o_1\}$
$s_5 = babanavea$	$\{a_4, b_2, e_1, n_1, v_1\}$	$\{s_1, w_1, y_1\}$	$\{q_1\}$
$s_6 = babana$	$\{a_3, b_2, n_1\}$	$\{s_1, w_1\}$	$\{u_1\}$

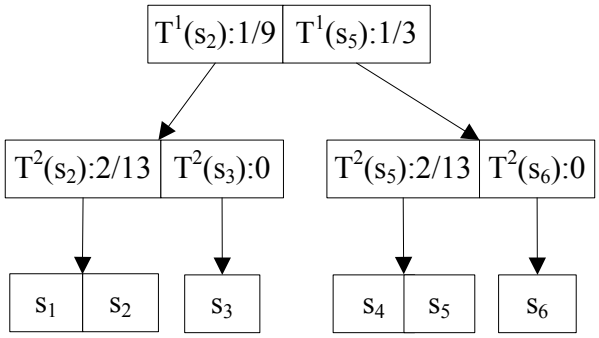


Figure 3: An Example of SEM-tree

Note that SEM-tree is a dynamic index structure, which can facilitate the data update based on the properties of multiset. For example, when a new sequence is inserted, we simply transform the original sequence to SEM-Space, and insert it into the appropriate sub-cluster from root. Due to the space constraint, we omit the details here.

3.3 Query Processing on SEM-tree

In this section, we investigate the distance property of multisets, and introduce a novel approach to accelerate the node filtering process. Finally, we present the whole algorithm of range query in the SEM-tree.

3.3.1 Double Bounds of Real Distance Between Sequences

While the proposed SEM-tree can conduct efficient similarity search over long sequences, it cannot guarantee absolute accuracy. This is because the search is conducted according to the distance between corresponding multisets rather than real sequences in the internal levels of the tree. As mentioned in Section 3.1.2, the distance between two multisets can approximate the real distance between two sequences with certain error, but such error has not been taken into consideration during tree construction. In this section, we develop a double-bound theory

to make sure that the searching process can cover all candidate sequences and complete query results can be returned.

According to [7], for sequences s_1 and s_2, the real distance can be bounded by the distance on embedding space as follows:

$$\alpha * \hat{d}_e(T(s_1), T(s_2)) \leq \hat{d}(s_1, s_2) \leq \beta * \hat{d}_e(T(s_1), T(s_2)) \tag{3}$$

where $\alpha = \frac{m}{8nlognlog^*n}$, $\beta = \frac{2m}{n}$,$n = max(|s_1|, |s_2|)$ and $m = max(card(T(s_1)), card(T(s_2)))$.

Based on equation 3, the real distance between any two sequences can be limited by double bounds $[\alpha * \hat{d}_e(T(s_1), T(s_2)), \beta * \hat{d}_e(T(s_1),$
$T(s_2))]$.

Based on the predictability of multiset in lemma 3, such double-boundary can be applied on the transformed sub-multiset for each internal node of SEM-tree, and we have the following revised boundary to support search process.

LEMMA 4 *For sequences s_1 and s_2, the real distance $\hat{d}(s_1, s_2)$ can be approximated by clustering vector at i-th level based on the double boundaries.*

$$\begin{aligned} \hat{d}(s_1, s_2) &\geq \alpha * B_l\Big(\hat{d}_e(T^i(s_1), T^i(s_2))\Big) \\ \hat{d}(s_1, s_2) &\leq \beta * B_u\Big(\hat{d}_e(T^i(s_1), T^i(s_2))\Big) \end{aligned}$$

Here, the lower boundary of $\hat{d}_e(T^i(s_1), T^i(s_2))$, i.e., $B_l\Big(\hat{d}_e(T^i(s_1),$
$T^i(s_2))$, is itself.

3.3.2 Node Filtering based on predictability

One promising feature of SEM-tree is that some of the final results can be determined beforehand when visiting the internal node. Although the query range is extended by using approximate distance in the internal nodes, we can quicken the query processing greatly based on the predicability property of sequence embedding. To the best of our knowledge, this mechanism has not been explored before.

LEMMA 5 *For a query sequence q, a clustering vector cv in an internal node at level i and query range γ, all sequences covered by cv are included in the results if the following inequation holds:*

$$\beta_d * B_u(\hat{d}_e(T^i(q), cv.sm)) + \beta_r * rad_u(cv_i) \leq \gamma$$

where β_d is the β value for $\hat{d}_e(T^i(q), cv.sm)$, and the same with the β_r for $rad_u(cv_i)$.

This lemma means that if the summation of upper bound distance $\beta_d * B_u(\hat{d}_e(T^i(q), cv.sm))$ and upper bound distance $\beta_r * rad_u(cv_i)$ is less than specified range γ, then all sequences covered by *cv* must be included in the final results. $B_u(\hat{d}_e(T^i(q), cv.sm))$ has been proved in lemma 4, and we have lemma 6 for $rad_u(cv_i)$.

LEMMA 6 *For a clustering vector cv at level i in SEM-tree, if $T^i(s)$ is the center of cv, and r is the radius, then the upper bound of radius cv in multiset can be estimated as:*

$$rad_u(cv_i) = \frac{|T^i(s)| * r + (2-r) * |T_i(s)|}{|T^i(s)| + (2-r) * |T_i(s)|}$$

PROOF Assume a sequence s' within cv has the distance r from s. According to the definition of distance computation in Equation 2, there are many possible choices for sequences within cv satisfying such constraints. To make sure that all sequences within cv are covered by the distance $\hat{d}_e(T(s), T(s'))$, we choose one representative sequence s' satisfying the constraints: $T^i(s') \supset T^i(s)$ with $\hat{d}_e(T^i(s), T^i(s')) = r$.

Obviously, the radius r is the maximal distance among the whole multiset between s and s', which is bounded by

$$\hat{d}_e(T(s), T(s')) = \frac{\frac{|T^i(s)|*r}{(1-r)} + card(\cup_{j=i+1}^{l_1} T_j(s) \ominus \cup_{j=i+1}^{l_2} T_j(s'))}{\frac{|T^i(s)|}{(1-r)} + card(\cup_{j=i+1}^{l_1} T_j(s) \cup (\cup_{j=i+1}^{l_2} T_j(s')))}$$

$$< \frac{\frac{|T^i(s)|*r}{(1-r)} + card((\cup_{j=i+1}^{l_1} T_j(s)) \cup (\cup_{j=i+1}^{l_2} T_j(s')))}{\frac{|T^i(s)|}{(1-r)} + card((\cup_{j=i+1}^{l_1} T_j(s)) \cup (\cup_{j=i+1}^{l_2} T_j(s')))}$$

where l_1 and l_2 are the height of $ET(s)$ and $ET(s')$ respectively.

Since the scale of the union $\cup_{j=i+1}^{l_1} T_j(s)$ and $\cup_{j=i+1}^{l_2} T_j(s')$ is smaller than that of $T_i(s)$ and $T_i(s')$, $card((\cup_{j=i+1}^{l_1} T_j(s)) \cup (\cup_{j=i+1}^{l_2} T_j(s')))$ can be bounded by $card(T_i(s) \cup T_i(s'))=\frac{(2-r)*T_i(s)}{(1-r)}$. In other words, $rad_u(cv_i)$ can be approximated by $\frac{|T^i(s)|*r+(2-r)*|T_i(s)|}{|T^i(s)|+(2-r)*|T_i(s)|}$. The proof is finished.

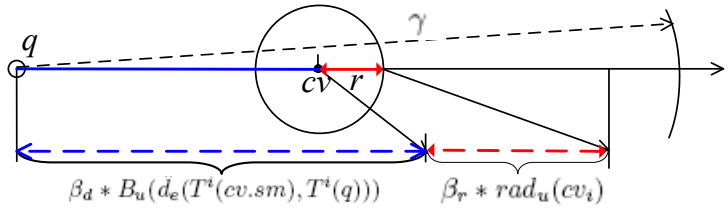
□

Thus, query processing can be accelerated with the following filtering strategy: At a particular internal node at level i, all irrelevant clustering vectors should be filtered out. First, for each cv, the double double of the real distance between $cv.sm$ and query sequence q can be obtained directly based on Lemma 4. Then, a filtering step will prune unqualified cv according to the following criteria.

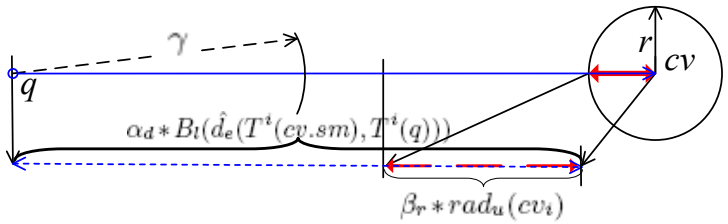
- As depicted in figure 4(a), if $\beta_d * B_u(\hat{d}_e(T^i(cv.sm), T^i(q))) + \beta_r * rad_u(cv_i) \leq \gamma$, cv does not need further checking. In such situation, no sequences contained by cluster cv need to be further checked, because they belong to the final results with the guarantee of upper-bound, and only need to be refined from the leaf nodes.

- As depicted in figure 4(b), if $\alpha_d * B_l(\hat{d}_e(T^i(cv.sm), T^i(q))) - \beta_r * rad_u(cv_i) > \gamma$, cv will not be further checked. The sub-tree rooted in cv will be filtered out, because it has no chance of being included in the result set with the guarantee of lower-bound.

- After filtering out above cvs, remaining cv set will be the results for further checking at the next level of the tree.



(a) prediction strategy



(b) filtering strategy

Figure 4: The filtering strategy of query processing

3.3.3 Query Processing over SEM-tree

With the above preparations, we now discuss how to process similarity search efficiently over SEM-tree. We only present the algorithm for range query, but the algorithm can support k-NN query with minor change.

A range query $Q_{Range} =< q, \gamma >$ retrieves all sequences s in the sequence datasets that satisfy the range condition $\hat{d}(q, s) \leq \gamma$, where γ is the query range specified by the user and $\hat{d}(*)$ is the distance measurement in original sequence space. To improve the efficiency of search, (sub-)multisets have been adopted as index keys in internal nodes of the SEM-tree. The detailed algorithm is provided in Algorithm 2.

Firstly, the query sequence q will be embedded into a multiset $T(q)$ and a search starting from the root of SEM-tree based on $T(q)$ is thereafter conducted. For each clustering vector in an internal node, those irrelevant clustering vectors will be first filtered out based on proposed filtering principe (in line 3). Otherwise, for each qualified clustering vector cv, it will be determined based on the proposed multiset predictability whether all sequences covered by the sub-tree rooted by cv belong to the final results(in line 4). If so, all these sequences will be appended to the final result RS and this subtree will not be further checked. Otherwise, the left clustering vectors need to be further checked by calling search process **RangeSearchOnTree**(*) recursively(in line 8). At the leaf node, each candidate sequence needs to be examined based on real distance function in original sequence space to determine whether they belong to final result set RS.

To conclude, the proposed sequence searching strategy can be summarized as following: *space-efficient multiset and related time-efficient distance function have been used to filter out those irrelevant sequence in internal nodes. Classical edit distance will be used in leaf nodes to refine the final results accurately.*

Algorithm 2 RangeQuery(q,γ)

Input: q: query sequence, γ: searching radius specified by user;

Output: RS: all the sequences that have a distance to q shorter than γ;

Procedure:

1: $RS = \emptyset$;
2: **RangeSearchOnTree(**$root, T(q), RS, 0$**)**;

Proc RangeSearchOnTree(Node N_d, multiset $T(q)$, SET RS, level l)

1: **if** T is not a leaf node **then**
2: **for** each $cv \in N_d$ **do**
3: **if** $\alpha_d * \hat{d}_e(T^l(cv.sm), T^l(q)) - \beta_r * rad_u(cv_i) \leq \gamma$ **then**
4: **if** $\beta_d * B_u(\hat{d}_e(T^l(q), cv.sm)) + \beta_r * rad_u(cv_i) \leq \gamma$ **then**
5: /**Node filtering based on predictability proposed*;*/
6: $RS = RS \cup$ {*all sequences in sub-tree rooted by* cv};
7: **else**
8: ***RangeSearchOnTree***$(cv.cptr, T(q), RS, l+1)$;
9: **end if**
10: **end if**
11: **end for**
12: **else**
13: **for** each $tp \in N_d$ **do**
14: **if** $\hat{d}(q, tp.seq) \leq \gamma$ **then**
15: $RS = RS \cup \{tp.seq\}$;/**filtering based on real distance**/
16: **end if**
17: **end for**
18: **end if**

4 Experiments

In order to evaluate the performance of proposed SEM-tree, we conducted extensive empirical study. We compared our method with several other existing global sequence indexing methods: 1) the M-Tree[23], 2) the DBM-Tree[22] and 3) the Slim-Tree[2]. The C++ implementations of the M-Tree, the DBM-Tree and the Slim-Tree, were obtained from Arboretum MAM library. All the implementations are running on a PC with Intel P4 CPU 1.5 GHz and 1G MB memory. The performance metric *query cost*, i.e. I/O and time cost, is taken into consideration. All the indexes are stored on disk, and the disk page size is set to 4KB. Therefore, I/O cost stands for the number of page readings for each query. On the other hand, time cost is the overall response time required to navigate the index and return the answers. The results shown in this section are the average performance of 200 range queries. The query sequences are randomly selected from the dataset.

Both synthetic and real datasets are employed in the experiments. Specifically, two datasets have been used in our experiments:

Synthetic Dataset: Dataset simulator SUMATRA [36] is employed to generate the syn-

thetic sequence dataset. It is a popular data simulator in mobile environments, which produces the moving trajectory sequences with various length and scale.

GENE Dataset: We also employ a real human GENE dataset comprising of four characters A, C, G and T, from GenBank[5].

The detailed parameter setting of the performance study is shown in Table 2 with default setting denoted by bold texts, where the character set size represents the number of distinct characters in the database. In each experiment, we only change the value of one parameter with all the rest adopting the default values.

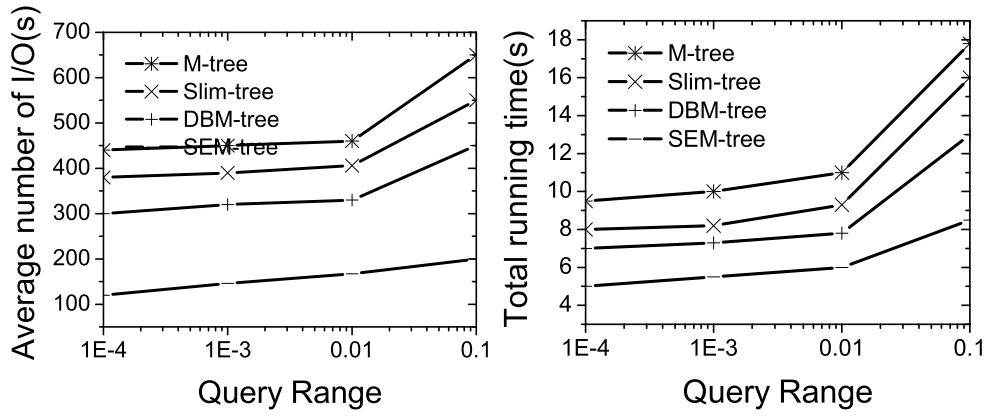
Table 2: Query parameters

Param	Description	Default Value
sl	Average length of the sequence dataset	50,**100**,150,200
s	The number of sequences	5k,**10k**,15k,20k
γ	Query range	0.0001,0.001,**0.01**,0.1
τ	The character set size (for SUMMTRA)	5,10,**15**,20,25

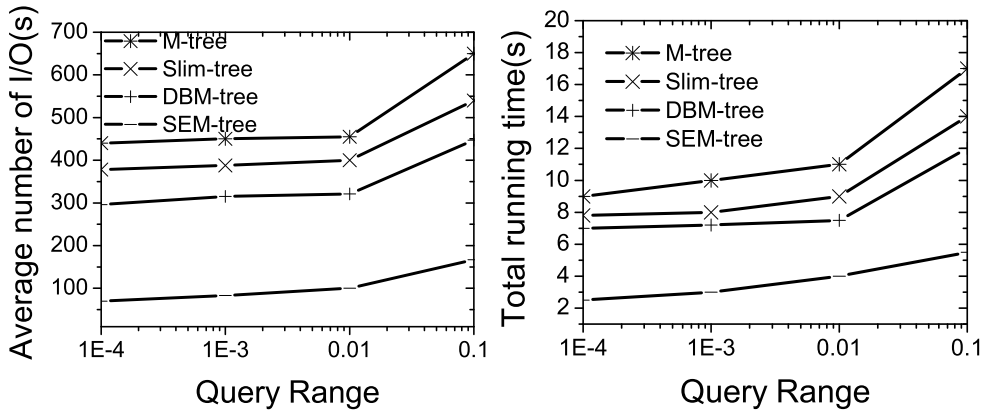
4.1 Impact of Query Range(γ)

The goal of our first experiment is to compare the performance of our method with existing methods for different query ranges on SUMATRA and GENE datasets with the range varying from $\gamma = 0.0001$ to 0.1. The results are shown in Figure 5, where the plots on x-axis are given in log-scale.

With the increase of query range, query cost (including both I/O and running time) increases for all the methods, but SEM-tree always performs better than three existing methods, i.e. DBM-tree, Slim-tree and M-tree. All these three indexes use cluster information over the full sequence length in the internal nodes, therefore, the computational cost of distance is high. The small tree fan-out also incurs heavy overlaps between nodes. Among three existing methods, DBM-tree always performs better than others(M-tree and Slim-tree). The DBM-Tree is unbalanced, which makes it reduce the overlap between nodes in high density regions. Compared with the DBM-tree, average running times of the SEM-tree are 33% and 47% for SUMATRA and GENE datasets. Regarding the I/O cost, SEM-tree saves around 50% and 69% of the cost of DBM-tree. The SEM-tree enables the distance computation between (sub-)multisets (*not the original sequences*) finished within linear time in the internal levels of the tree. The distance on the short (sub-)multisets can effectively filter most irrelevant sequences in the database. On the other hand, we can reduce the I/O cost greatly by storing more entries in each internal node, as each entry is represented by (sub-)multisets with small length. Additionally, the multiset representation can increase the fanout of the SEM-tree, thus reducing the overlap between nodes. Furthermore, according to the predictability of SEM-tree, some query processes in a certain query can be finished beforehand in internal node levels without accessing the leaf nodes, which can further reduce query cost.



(a) Synthetic Data: SUMATRA



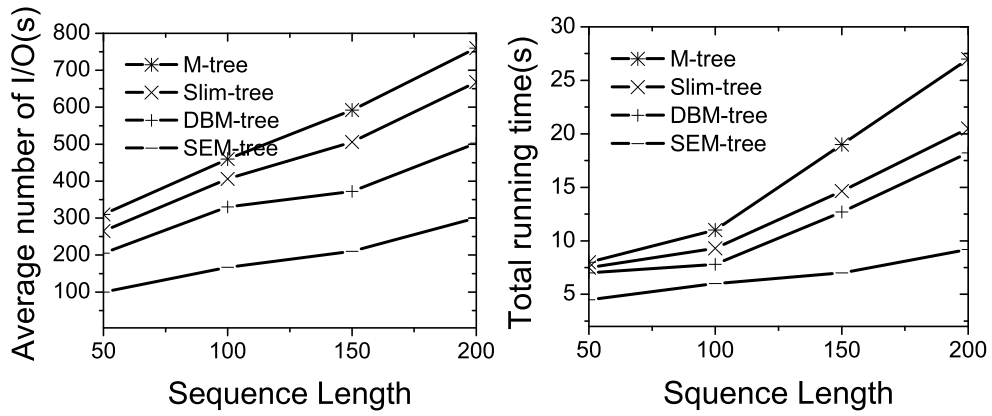
(b) Real Data: GENE

Figure 5: Average number of I/O(s) and total running time(s) for SEM-tree, M-tree, Slim-tree and DBM-tree on SUMATRA and GENE for queries with varying ranges from 0.0001 to 0.1. The default values are (sl=100, s=10000 and $\tau = 15$ for SUMATRA).

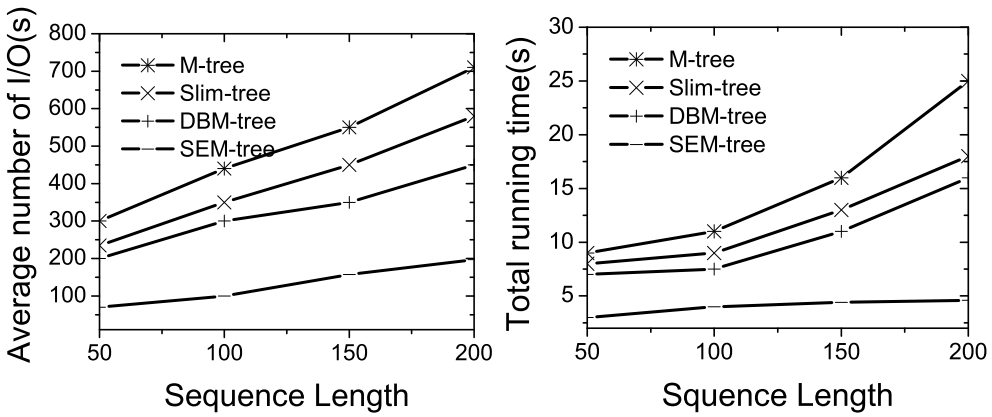
We also consider the Sequential Scan approach as the base-line solution which does not employ any index, however the Sequential Scan method yields much worse performance compared with all the index based approaches since it has to access all the sequence data for any query, and hence we omit the results in the figures. For example, the I/O cost for Sequential Scan on SUMATRA dataset is 1000 given the default parameters, which is 2-6 times higher than index based methods.

4.2 Scalability in Sequence Length (sl)

The resilience to the increase of sequence length is one of the most critical features for an index structure targeting long sequence dataset. The goal of this experiment is to compare the performance of our method with existing methods for increasing lengths of sequences. Figure 6 presents the results with sl(*sequence length*) varying from 50 to 200 under SUMATRA dataset and GENE dataset. Query range γ is 0.1.



(a) Synthetic Data: SUMATRA

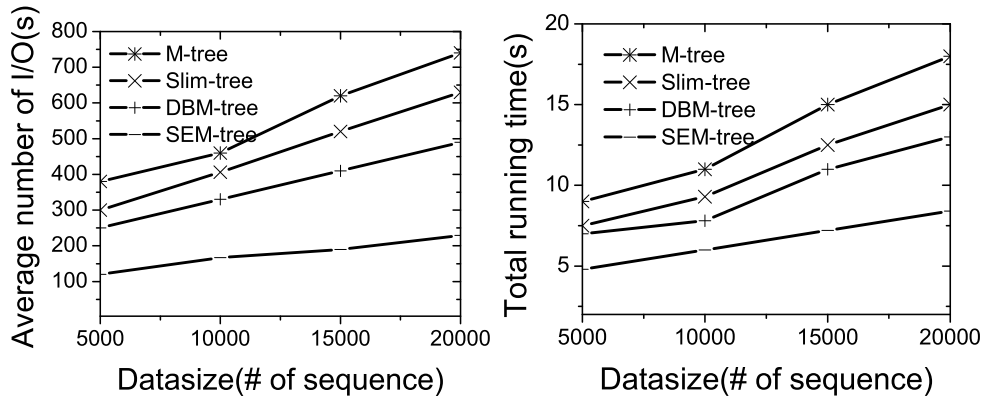


(b) Real Data: GENE

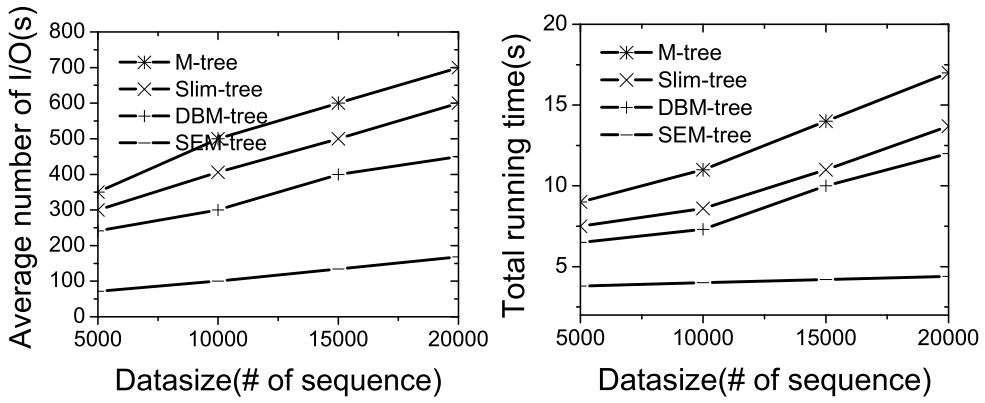
Figure 6: Average number of I/O(s) and total running time(s) for SEM-tree, M-tree, Slim-tree and DBM-tree on SUMATRA and GENE for queries with varying sequence length from 50 to 200. The default values are (γ=0.01, s=10000 and τ = 15 for SUMATRA).

As sl increases, each sequence has more characters, which results in higher tree and more expensive distance computation. However, the SEM-tree is based on the multiset, which is a compact representation of the real sequence. A multiset only records the distinct characters of sequence and corresponding occurrence times, not the orders. Therefore, an increased sl does not deteriorate the performance of the SEM-tree obviously as depicted in figure 6.

Our SEM-tree can be about 23% (46%) faster than the other three methods in SUMATRA dataset (GENE dataset), especially when the sequence is long. The SEM-tree has two advantages over other three indexes. Firstly, the SEM-tree reduces the number of I/O(s) compared to other three indexes (represented by DBM-tree). Because we index different levels of multisets of the dataset, the length of sub-multisets in the upper levels is much smaller than that of the original sequence. Besides, the fanout of SEM-tree in upper levels is much larger than other methods, making the index size of the SEM-tree smaller. With the default parameters of the datasets, the height of SEM-tree is only 3 for GENE data and 4 for SUMATRA data, while the other three



(a) Synthetic Data: SUMATRA



(b) Real Data: GENE

Figure 7: Average number of I/O(s) and total running time(s) for SEM-tree, M-tree, Slim-tree and DBM-tree on SUMATRA and GENE for queries with varying datasize from 5000 to 20000. The default values are (γ=0.01, sl=100 and τ = 15 for SUMATRA).

trees have 6 levels. Secondly, the computational cost of SEM-tree is smaller than DBM-tree. The reason is that in the internal levels, the distance computation is much faster because of reduced length of multisets.

4.3 Scalability in Database Size (s)

In this experiment, we test the scalability of performance with respect to the data size. We fix the length of sequences at 100 and query range γ at 0.01, and vary the dataset size from 5,000 to 20,000 sequences. The performance of range queries on SUMATRA and GENE datasets for SEM-tree, M-tree, DBM-tree and Slim-tree, including an average number of I/O(s) and total running time, has been shown in Figure 7.

Generally, with the increase of data size, more sequences will be covered for the fixed query range γ, which not only incurs the increment of distance computation but also but also leads to more I/O operations involved. Therefore, the performances of all indexes decrease as depicted in Figure 7. However, the SEM-tree is still about 40-50% better than other methods for range

query on SUMATRA and GENE dataset. These results clearly show the efficiency of the SEM-tree over the three compared indexes. This is because SEM-tree uses (sub-)multisets rather than the full length in the internal nodes, and the latter can lead to more I/O operations and distance computational cost.

4.4 Impact of character set size (τ)

In the last experiment, we evaluate the performance of our method using different character set sizes. Here, we compare the performance of SEM-tree with other three indexes by fixing the query range (γ=0.01) for a varying character set size τ = 5, 10, 15, 20 and 25 under SUMATRA data. Based on the sequence embedding technique, we know that the length of multiset (or sub-multiset) highly depends on the character set size τ. Therefore, the performance of SEM-tree may be influenced by the changing of τ, which has been proved in these experiments.

As shown in Figure 8, as the number of character set size increases, the performance of SEM-tree degrades gradually. The reason is that the effectiveness of sequence embedding is deteriorated, as we need to use longer multiset to represent the original sequence. However, the SEM-tree still performs better than other schemes for a relatively large τ, e.g. 20. Note that, the performances for DBM-tree, Slim-tree and M-tree are kept unchanged because the query processing is unrelated with τ but only influenced by the length of sequence *sl*.

The result of this experiment also justifies the superiority of the proposed SEM-tree for the sequence datasets with a small number of distinct characters, such as DNA sequences.

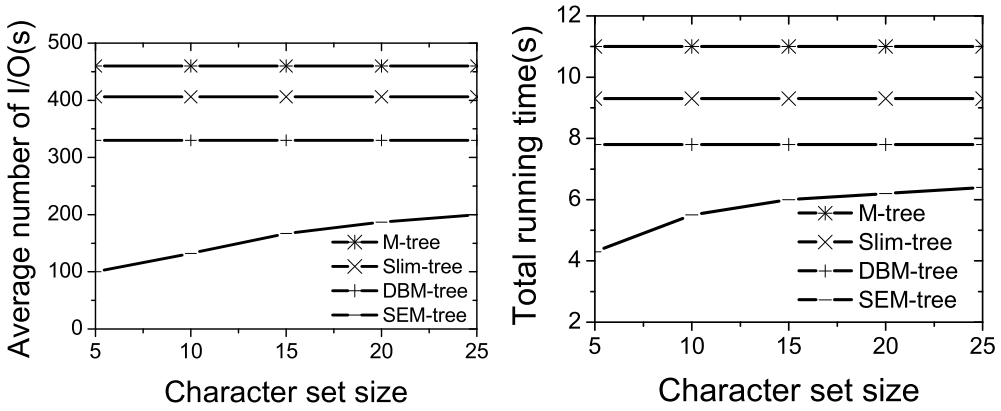


Figure 8: Average number of I/O(s) and total running time(s) for SEM-tree, M-tree, Slim-tree and DBM-tree on SUMMTRA for queries with varying character set size from 5 to 25. The default values are (γ=0.01, s=10000 and sl = 100).

5 Conclusion

With numerous emerging applications requiring efficient similarity search over sequence datasets, there is a high demand for a scalable index structure that can serve long sequence datasets.

In this paper, we proposed a "dimensionality reduction" like mechanism based on sequence embedding technique to minimize the expensive distance computational cost. As an application of the derived properties, a novel index structure, SEM-tree was presented to index sequences. In a SEM-tree, each level represents the sequence data with different compression level of multisets, whose sizes increase from root to leaf level. As demonstrated by the comprehensive simulations, SEM-tree shows a much better performance than existing schemes, in terms of CPU cost and I/O cost.

References

[1] B. Cui, B. C. Ooi, J. Su, K.L. Tan. Contorting High Dimensional Data for Efficient Main Memory KNN Processing. In Proc. of the International Conference on Management of Data (SIGMOD'03), San Diego, 2003, pages 479-490.

[2] Caetano Traina, Agma J. M. Traina, Bernhard Seeger, and Christos Faloutsos. Slim-Trees: High Performance Metric Trees Minimizing Overlap Between Nodes. In proc. of the 7th International Conference on Extended Database Technology(EDBT'00), pages 51-65, 2000.

[3] C. Faloutsos and K. I. Lin. Fast Map: A fast algorithm for indexing, data mining and visualization of traditional and multimedia datasets. In Proc. of the International Conference on Management of Data (SIGMOD'95), pages 163-174, 1995.

[4] DanTong Yu,Aidong Zhang. ClusterTree:Integration of Cluster Representation and Nearest-Neighbor Search for Large Data Sets with High Dimensions[J].IEEE Transactions on Knowledge and Data Engineering, 2003; 15 (5): 1316-1337

[5] D.A. Benson, I. Karsch-Mizrachi, D.J. Lipman, J. Ostell, B.A. Rapp, and D.L. Wheeler. GenBank. Nucleic Acids Research, 28(1):15-18, 2000.

[6] E.M. McCreight. A Space-Economical Suffix Tree Construction Algorithm. JACM, 23(2):262-272, 1976.

[7] G. Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. In Proc. of the 13th annual ACM-SIAM symposium on Discrete Algorithms, pages 667-676, 2002.

[8] G.H. Golub and C.F. Van Loan. Matrix Computations. The Johns Hopkins University Press, 1989.

[9] G.Navarro. A guided tour to approximate sequence matching. ACM Comput Survey 33:1:31-88, 2000.

[10] Guojie SONG, Bin Cui, Baihua ZHENG, Kunqing Xie, Dongqing Yang. Squeezing Long Sequence Data for Efficient Similarity Search. in: Proc. Apweb, 2008, pp. 438-449.

[11] G. Navarro and R. Baeza-Yates. A Hybrid Indexing Method for Approximate String Matching. J. Discret. Algorithms, 1(1):205-239, 2000.

[12] Gusfield, D. 1997. Algorithms on sequences, trees and sequences: Computer science and computational biology. Cambridge University Press, UK.

[13] H. V. Jagadish, Beng Chin Ooi, Kian-Lee Tan, Cui Yu, and Rui Zhang. iDistance: An adaptive B^+-tree based indexing method for nearest neighbor search. ACM Trans. on Data Base Systems, 2005,30(2):364-397.

[14] I. T. Jolliffe. Principal Component Analysis. Springer-Verlag, 1986.

[15] Jayendra Venkateswaran, Deepak Lachwani, Tamer Kahveci, Christopher M. Jermaine: Reference-based Indexing of Sequence Databases. In Proc. 24th VLDB Conference(VLDB'06), pages 906-917.

[16] Keogh E and Ann RC (2005). Exact indexing of dynamic time warping. Knowl Info Syst 7(3): 358-386

[17] Keogh E, Chakrabarti K, Mehrotra S and Pazzani M (2001). Dimensionality reduction for fast similarity search in large databases. Knowl Info Syst 3(3): 263-286.

[18] K. Chakrabarti and S. Mehrotra. Local dimensionality reduction: A new approach to indexing high dimensional spaces. In Proc. 26th VLDB Conference, pages 89-100, 2000.

[19] Lee S, Chun S, Kim D. Similarity search for multidimensional data sequences. In Proc. of the 16th Int'l Conf. on Data Engineering. Washington: IEEE Computer Society, 2000. 599-608.

[20] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Tolga Bozkaya and Meral Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. In Proc. of the International Conference on Management of Data (SIGMOD'97), pages 357-368, 1997.

[21] M. Li, J. H. Badger, C. Xin, S. Kwong, P. Kearney, and H. P. Ferragina and R. Grossi. The String B-tree: A New Data Structure for String Search in External Memory and Its Applications. JACM, 46(2):236-280, 1999.

[22] Marcos R. Vieira, Caetano Traina, Fabio Jun Takada Chino, and Agma J. M. Traina. DBM-Tree: A Dynamic Metric Access Method Sensitive to Local Density Data. Simposio Brasileiro de Bancos de Dados(SBBD'04), pages 163-177, 2004.

[23] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In Proc. 24th VLDB Conference(VLDB'97), pages 194-205, 1997.

[24] Popivanov I, Miller RJ. Similarity search over time series data using wavelets. In Proc. of the 18th Int'l Conf. on Data Engineering, ICDE 2002. San Jose: IEEE Computer Society, 2002. 212-221.

[25] P.N. Yianilos. Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces. In SODA, pages 311-321, 1993.

[26] Pearson,W. and Lipman,D. Improved tools for biological sequence comparison. In Proc. Natl Acad. Sci., USA, 85, 2444-2488, 1988.

[27] P. Weiner. Linear Pattern Matching Algorithms. In IEEE Symposium on Switching and Automata Theory, pages 1-11, 1973.

[28] Roberto F. Santos Filho, Agma J. M. Traina, Caetano Traina, and Christos Faloutsos. Similarity Search without Tears: The OMNI Family of All-purpose Access Methods. In Proc, of the 19th International Conference on Data Engineering (ICDE'01), pages 623-630.

[29] R. M. Karp, R. E. Miller, and A. L. Rosenberg. Rapid identification of repeated patterns in strings, trees and arrays. In Proc. of the 4th Symposium on Theory of Computing, pages 125-136, 1972.

[30] R. Weber, H. J. Schek, and S. Blott. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In Proc. International Conference on Very Large Databases (VLDB'98), pages 194-205, 1998.

[31] S. C. Sahinalp, M. Tasan, J. Macker, and Z. M. Ozsoyoglu. Distance-Based Indexing for String Proximity Search. In Proc. of the 19th International Conference on Data Engineering (ICDE'03). pages 125-136.

[32] Srividya Kadiyala and Nematollaah Shiri (2007). A compact multi-resolution index for variable length queries in time series databases. Knowl Info Syst 15:131-147.

[33] S. Muthukrishnan and S. C.S. ahinalp. Approximate nearest neighbors and sequence comparison with block operations. In Proceedings of the 32nd Symposium on Theory of Computing, pages 416-424, 2000.

[34] T.H. Cormen, C.E. Leiserson, R.L. Rivest. Introduction to algorithms. MIT, 1990.

[35] T. L. Wang, X. Wang, K. I. Lin, D. Shasha, B. Shapiro, and K. Zhang. Evaluating a class of distance-mapping algorithms for data mining and clustering. In Proc. of the 5th ACM International Conference of Knowledge Discovery and Data Mining (SIGKDD'99), pages 307-311, 1999.

[36] http://www-db.stanford.edu/pleiades/SUMATRA.html.

[37] Zhang,Z., Schwartz,S., Wagner,L. and Miller,W. (2000) A greedy algorithm for aligning DNA sequences. J. Comput. Biol., 7, 203-214.

6 Appendix

Multiset and Related Operations

Definiton 1 *(**Multiset**:) A multiset is a pair (X,f), where X is a set, and f is a function mapping X to the cardinal numbers greater than zero. For any $x \in X$, $f(x)$ is called the multiplicity of x.*

Definiton 2 *(**card()**) Suppose that $\mathcal{A} = < A, f >$ is a multiset, then its cardinality, denoted card($\mathcal{A}$), is defined as: card($\mathcal{A}$)=$\sum_{x \in A}(f(a))$.*

Definiton 3 *(**$A \ominus B$**) Suppose that $\mathcal{A} = < A, f >$ and $\mathcal{B} =< A, g >$ are two multisets, then the removal of multiset B from A, denoted $A \ominus B$, is the multiset $\mathcal{C} =< A, h >$, where for all $a \in A$: $h(a) = max(f(a) - g(a), 0)$.*

Definiton 4 *(**$A \cup B$**) Suppose that $\mathcal{A} = < A, f >$ and $\mathcal{B} =< A, g >$ are two multisets, then their union, denoted $A \cup B$, is the multiset $\mathcal{C} =< A, h >$, where for all $a \in A$: $h(a) = max(f(a), g(a))$.*

Definiton 5 *(**$A \cap B$**) Suppose that $\mathcal{A} = < A, f >$ and $\mathcal{B} =< A, g >$ are two multisets, then their intersection, denoted $A \cap B$, is the multiset $\mathcal{C} =< A, h >$, where for all $a \in A$: $h(a) = min(f(a), g(a))$.*