Indexing DNA Sequences Using q-grams

Xia Cao, Shuai Cheng Li, and Anthony K. H. Tung
{caoxia,lisc,atung}@comp.nus.edu.sg

Department of Computer Science, National University of Singapore

Abstract. We have observed in recent years a growing interest in similarity search on large collections of biological sequences. Contributing to the interest, this paper presents a method for indexing the DNA sequences efficiently based on q-grams to facilitate similarity search in a DNA database and sidestep the need for linear scan of the entire database. Two level index – hash table and c-trees – are proposed based on the q-grams of DNA sequences. The proposed data structures allow the quick detection of sequences within a certain distance to the query sequence. Experimental results show that our method is efficient in detecting similarity regions in a DNA sequence database with high sensitivity.

1 Introduction

Similarity search on DNA database is an important function in genomic research. It is useful for making new discoveries about a DNA sequence, including the location of functional sites and novel repeatitive structures. It is also useful for the comparative analysis of different DNA sequences. Approximate sequence matching is preferred to exact matching in genomic databases due to evolutionary mutations in the genomic sequences and the presence of noise data in a real sequence database. Many approaches have been developed for approximate sequence matching. The most fundamental one is the Smith-Waterman alignment algorithm [14] which is a dynamic programming approach that seeks the optimal alignment between a query and the target sequence in $O(mn)$ time, m and n being the length of the two sequences.

However, these methods are not practical for long sequences in the megabases range. Effort to improve the efficiency falls into the common idea of filtering by discarding the regions with low sequence similarity. A well known approach is to scan the biological sequences and find short "seed" matches which are subsequently extended into longer alignments. This method is used in program like FASTA [13] and BLAST [1] which are the most popular tools used by biologists. An alternative approach is to build index on the data sequences and conduct the search on the index. Various index structure models [2, 4, 7, 17] have been proposed for this purpose.

Our method is based on the observation that two sequences share a certain number of q-grams if the edit distance between them is within a certain threshold. Moreover, since there are only four letters in the DNA alphabet, we know that the number of all combinations of q-grams in a DNA sequence is 4^q.

In this paper, we propose two level index to prune data sequences that are far away from the query sequence. The disjoint segments with the length ω are generated from the sequence. In the first level, the clusters (called qClusters) of similar q-grams in

DNA sequence are generated; then a typical hash table is built in the segments with respect to the qClusters. In the second level index, the segments are transformed into the c-signatures based on their q-grams; then a new index called the *c-signature trees* (c-trees) is proposed to organize the c-signatures of all segments of a DNA sequence for search efficiency.

In the first level of search, the sliding segment of query sequence is generated and encoded into the key in terms of the coding function, and then the neighbors of this key will be enumerated. Thus a set of candidate segments will be extracted from the buckets pointed by the key and its neighbors, and be put into the second index structure c-trees for future filtering. In the second level of search, we only access the tree paths in c-trees that include possible similar data sequences in their leaf nodes. We also propose a similarity search algorithm based on the c-trees for query segments.

The rest of paper is organized as follows. In Section 2, we define the problem of similarity search in DNA sequence databases and briefly review related work. In Section 3, the concept of qClusters and c-signature is presented. The filter principle based on q-grams is also described. In Section 4, we propose two-level index scheme constructed on the q-grams for DNA sequences. In Section 5, an efficient similarity search algorithm is presented based on the proposed index structure. The test data and experimental results are presented in Section 6. Section 7 summarizes the contribution of this paper.

2 Problem Definition and Related Work

In this section, we formalize the similarity search problem in a DNA sequence database and describe the related existing work.

2.1 Problem Definition

The problems of approximate matching and alignment are the core issues in sequence similarity search. To process approximate matching, one common and simple approximation metric is called $edit\ distance$.

Definition 1. Edit Distance
The edit distance between two sequences is defined as the minimum number of edit operations (i.e., insertions, deletions and substitutions) of single characters needed to transform the first string into the second. $\mathrm{ed}(S, P)$ *is used to denote the edit distance between sequence* S *and* P*.*

In general, this problem of sequence search can be described formally as follow:

Problem 1. Given the length l and edit distance ϑ, find all subsequences S in $\mathcal{D}$ which have length $|S| \geq l$ and $ed(S, Q') \leq \vartheta$ for subsequence Q' in query sequence Q.

Since with high possibility there exists a similar segment pair $(s, q), s \in S, q \in Q'$ if S is similar to Q', we instead solve the following problem.

Problem 2. Given the length ω and edit distance ε, find all the segments s_i with length ω in $\mathcal{D}$ which meet $ed(s_i, q_j) \leq \varepsilon$ for the query segments q_j with length ω in Q.

2.2 Related Work

A great amount of work has been done to improve search efficiency and effectiveness in DNA sequence databases. BLAST[1] is a heuristic method for finding similar regions between two genomic sequences. It regards the exact match of W contiguous bases as candidates which are then extended along the left side and the right side to obtain the final alignments. Unfortunately, BLAST faces the dilemma of DNA homology search: increasing the seed size W decreases sensitivity while decreasing the seed size results in too many random results. PatternHunter [8] is an improvement on BLAST both in speed and sensitivity through the use of non-consecutive k letters as model. In essence, PatternHunter's basic principles are similar to those of BLAST.

Researchers have also proposed indices for sequence databases. The suffix tree family is a well-studied category of indices to resolve string-related problems [16, 9, 2, 11, 10]. QUASAR [2] applies a modification of q-gram filtering on top of a suffix array. However, its performance deteriorates dramatically if the compared sequences are weakly similar. Also, the resulting index structure based on the suffix array and suffix tree is large compared to the size of the sequence database. Even if the suffix tree is used without links as proposed in [5], the suffix tree structure index is still nearly 10 times the size of the sequence database. Oasis [10], a novel fast search algorithm is driven by a suffix tree and it also suffers the large size of index structure.

In [15], the ed-tree is proposed to support probe-based homology search in DNA sequence databases efficiently. But the size of the tree-structure index is larger than the sequence database and also it is very time-consuming to build the ed-tree for sequences. Recently, some attempts [7, 12] have been made to transform DNA sequences into numerical vector spaces to allow the use of multi-dimensional indexing approaches for sequence similarity search. Though these methods avoid false dismissals and offer very fast filtering, their drawback is that the approximation of edit distance is not sufficiently tight, which increases the cost of refining results for final output.

SST [4] has been shown to be much faster than BLAST when searching for highly similar sequences. Unfortunately, since the distance between sequences in vector space does not correspond well with the actual edit distance, a larger number of false dismissals may occur if the similarity between the query sequence and the target sequence is not sufficiently high. Williams et al. [17] proposed a search algorithm in a research prototype system, CAFE, which uses an inverted index to select a subset of sequences that display broad similarity to the query sequence. The experiments show that CAFE is faster but also less sensitive than BLAST when searching for very similar sequences.

3 Notations

Although the edit distance is a simple but fairly accurate measure of the evolutionary proximity of two DNA sequences, the computation complexity is $O(mn)$, m and n being the length of the two sequences. To speed up approximate sequence matching, filtering is an efficient way to quickly discard parts of a sequence database, leaving the remaining part to be checked by the edit distance. Our proposed approach to sequence similarity search is based on q-grams, where the q-gram similarity is used as a filter.

3.1 Preliminaries

Before we define $qClusters$ and $c\text{-}signature$, we shall briefly review q-grams and q-gram based filter. The intuition behind the use of q-grams as a filter for approximate sequence search is that two sequences would have a large number of q-grams in common when the edit distance between them is within a certain number.

Definition 2. q-gram of Sequence
Given a sequence S, its q-grams are obtained by sliding a window of length q over the characters of S. For a sequence S, there are $|S| - q + 1$ q-grams.

Lemma 1. Filter based on q-grams (Jokinen and Ukkonen [6])
Let an occurrence of $Q[1 : w]$ with at most ε edit or hamming distance end at position j in sequence database S. Then at least $w + 1 - (\varepsilon + 1)q$ of the q-grams in $Q[1 : w]$ occur in the substring $S[j - w + 1 : j]$. In another word, there are at most εq q-grams in $Q[1 : w]$ which do not occur in $S[j - w + 1 : j]$, and vice versa. So obviously, the number of different q-grams between $Q[1 : w]$ and $S[j - w + 1 : j]$ is at most $2\varepsilon q$.

3.2 The qClusters and c-Signature

The alphabet of the DNA sequence comprises four letters: $\Sigma = \{A, C, G, T\}$. It means there are in all $|\Sigma|^q = 4^q$ kinds of q-grams, and we may arrange them according to the lexicographic order, and use r_i to denote the ith q-gram in this order. All the possible q-grams are denoted as: $\Re = \{r_0, r_1, \ldots, r_{4^q-1}\}$. The q-gram clusters(qClusters) can be defined below:

Definition 3. q-gram Clusters (qClusters)
All the possible q-grams, $\Re = \{r_0, r_1, \ldots, r_{4^q-1}\}$ are divided into λ clusters (denoted as $qClusters$) $\{qCluster_1, \ldots, qCluster_\lambda\}$ by a certain principle. In this paper, we simply cluster the m continuous q-grams $\{r_{(i-1)m}, \ldots, r_{im-1}\}$ together into $qCluster_i$, $1 \le i \le \lambda = \lceil \frac{4^q}{m} \rceil$.

The q-gram signature and c-signature of the DNA sequence are defined as follows:

Definition 4. q-gram Signature
The q-gram signature is a bitmap with 4^q bits where ith bit corresponds to the presence or absence of r_i. For a given sequence S, the ith bit is set as '1' if $r_i \in \Re$ occurs at least once in sequence S, else it is set as '0'.

Definition 5. c-signature
Let $sig^1(S)$=$(a_0, \ldots, a_{n-1})$ be a q-gram signature of the DNA segment S with n=4^q, then its c-signature is defined as: $sig^c(S) = (u_0, \ldots, u_{k-1})$ where k=$\lceil n/c \rceil$, and $u_i = \sum_{j=ic}^{(i+1)c-1} a_j$. Set $a_j = 0$ when $n \le j < ck$. For sequence S and P, we define the distance between $sig^c(S) = (u_0, \ldots, u_{k-1})$ and $sig^c(P) = (v_0, \ldots, v_{k-1})$ as $SDist(sig^c(S), sig^c(P))$=$\sum_{i=0}^{k-1} |u_i - v_i|$.

For better understanding of the definition of q-gram signature and c-signature, we consider the below example:

Example 1. For sequence P="ACGGTACT", its q-gram signature is ($\underline{01}$ $\underline{00}$ $\underline{00}$ $\underline{11}$ $\underline{00}$ $\underline{11}$ $\underline{10}$ $\underline{00}$) with 16(=4^2) dimensions when $q = 2$. In P, the q-gram 'AC' occurs twice in position 0 and 5, so we set the corresponding bit in position 1 in q-gram signature as '1'. As there is no occurrence of 'AA' in sequence P, the corresponding bit in position 0 in q-gram signature is set as '0'. For c=2, the c-signature of P is (10020210) with respect to the definition of the c-signature.

With the property $|a| + |b| \geq |a + b|$, it is not difficult to obtain the following lemma for filtering in terms of c-signature:

Lemma 2. Filter Based on c-signatures
Given a sequence S, there is at most ε edit or hamming distance from another sequence P with $|S| = |P|$. Let $sig^1(S) = (a_0, a_1, \ldots, a_{n-1})$ and $sig^1(P) = (b_0, b_1, \ldots, b_{n-1})$ be the q-gram signatures generated for sequence S and P respectively. Denote the c-signatures of S and P as $sig^c(S) = (u_0, u_1, \ldots, u_{k-1})$ and $sig^c(P) = (v_0, v_1, \ldots, v_{k-1})$, $c > 1$, respectively. Then $\sum_{i=0}^{k-1} |u_i - v_i| \leq \sum_{j=0}^{n-1} |a_j - b_j| \leq 2\varepsilon q$.

Proof: *In term of Lemma 1 and the definition of q-gram signature, $\sum_{i=0}^{n-1} |a_i - b_i| \leq 2\varepsilon q$ holds. According to the definition of c-signature, $u_i = \sum_{j=ic}^{(i+1)c-1} a_j$ and $v_i = \sum_{j=ic}^{(i+1)c-1} b_j$. The following formula holds: $\sum_{i=0}^{k-1} |u_i - v_i| = \sum_{i=0}^{k-1} |\sum_{j=ic}^{(i+1)c-1} a_j - \sum_{j=ic}^{(i+1)c-1} b_j| \leq \sum_{i=0}^{k-1} \sum_{j=ic}^{(i+1)c-1} |a_j - b_j| = \sum_{j=0}^{kc-1} |a_j - b_j| = \sum_{j=0}^{n-1} |a_j - b_j| \leq 2\varepsilon q$.*

4 An Indexing Scheme for DNA Sequences

A two-level indexing scheme is proposed to organize the segments in DNA sequence database and support the similarity search.

4.1 The Hash Table

In order to hash the DNA segments to a hash table with size 2^λ, it is necessary to encode the segment into a λ-bit integer. Given a segment s, we encode it into a λ bitmap $e = (e_1, e_2, \ldots, e_\lambda)$ with respect to qClusters=$\{qCluster_1, qCluster_2, \ldots, qCluster_\lambda\}$. If there exists a q-gram $gram$ in s which meets $gram \in qCluster_i$, we set $e_i = 1$, else $e_i = 0$, where $1 \leq i \leq \lambda$. Following the encoding principle, any DNA segment s can be encoded into a λ-bit integer $(e_1, \ldots, e_\lambda)$ by the coding function:

$$coding(s) = \sum_{i=1}^{\lambda} 2^{i-1} e_i$$

The hash table has totally 2^λ buckets for the qClusters $\{qCluster_1, \ldots, qCluster_\lambda\}$, and each segment s_i can be inserted into the corresponding bucket in the hash table with the use of the hash function $coding(s_i)$. Note that λ is set as 15 and 22 for q=3 and 4 respectively in the experimental studies for better performance, and we will not declare it again.

4.2 The c-trees

The c-trees are a group of rooted dynamic trees built for indexing c-signatures. The height of the trees, ℓ is set by the users. Given the c-signature of the segment s, $sig^c(s)$= $(v_0, v_1, \ldots, v_{k-1})$, there are $\delta = \lceil \frac{k}{\ell} \rceil$ trees in total. We denote these trees as T_0,...,$T_{\delta-1}$. Each path from the root to a leaf in T_i corresponds to the c-signature string $sig_i^c(s) = (v_{i\ell}, v_{i\ell+1}, \ldots, v_{(i+1)\ell-1})$. For ease of discussion, we shall assume without loss of generality that k is divisible by ℓ and thus $T_{\delta-1}$ also has a height of ℓ. For each internal node of the tree, there are at most $c+1$ children. Each edge in a tree of c-trees is labeled with the respective value from 0 to c.

The DNA segments are transformed into the c-signatures in order to build the c-trees on them. Note that it is not necessary to store the c-signatures themselves after the trees are constructed. To further consolidate the definition of c-trees, we shall present a straightforward algorithm to build c-trees for a group of c-signatures.

In Algorithm 1, $label[\langle N_x, N_y\rangle]$ denotes the label of edge $\langle N_x, N_y\rangle$ in the c-trees. For notation convenience, define $S - S^{'}$ as a suffix of S, where $S^{'}$ is a prefix of S, and the concatenation of $S^{'}$ and $S - S^{'}$ is S. ϵ is used to refer an empty string. Also $lNode$ denotes the leaf node in the c-trees, and $E_0[lNode]$ is a group of segments in $lNode$ of the first tree T_0. Note that $E_0[*]$ will be constructed only for the tree T_0. For the other trees, the link from the c-signature to the leaf node will be constructed instead.

Algorithm 1 Tree Construction
Input: c-signatures $sig^c(s_0), \ldots, sig^c(s_{|\mathcal{D}|-\omega})$ **Output:** c-trees $(T_0, T_1, \ldots, T_{\delta-1})$

1: $T_i \leftarrow NULL, 0 \leq i < \delta$
2: **for** each c-signature $sig^c(s_j)$ **do**
3: **for** $i \leftarrow 0 \ldots \delta - 1$ **do**
4: TreeInsert(T_i, $sig_i^c(s_j)$, s_j)
5: **end for**
6: **end for**
7:
8: Function TreeInsert(N_x, sig, s)
9: **if** $sig = \epsilon$ **then**
10: insert(N_x, s, i) /*N_x is the leaf node*/
11: **else if** there exists an edge $\langle N_x, N_y\rangle$ where $label[\langle N_x, N_y\rangle]$ is a prefix of sig **then**
12: TreeInsert(N_y, $sig - label[\langle N_x, N_y\rangle]$, s)
13: **else if** there exists an edge $\langle N_x, N_y\rangle$ where $label[\langle N_x, N_y\rangle]$ shares a longest prefix pf with sig, $pf \neq \epsilon$ **then**
14: split $\langle N_x, N_y\rangle$ into two parts with a new node N_z, such that $pf = label[\langle N_x, N_z\rangle]$
15: create a new leaf $lNode$ with edge label sig-$label[\langle N_x, N_z\rangle]$ under N_z
16: insert($lNode$, s, i)
17: **else**
18: create a new leaf node $lNode$ under N_x with edge label $label[\langle N_x, lNode\rangle]$=$sig$
19: insert($lNode$, s, i)
20: **end if**
21:
22: Function insert($lNode$, s, i)
23: **if** i=0 **then**
24: $E_0[lNode] \leftarrow E_0[lNode] \cup \{s\}$
25: **else**
26: build the link from c-signature of s to $lNode$ in T_i
27: **end if**

The c-signature strings $sig_i^c(s)$ are inserted into the growing trees T_i $0 \leq i < \delta$ one by one by executing the function TreeInsert($T_i, sig_i^c(s), s$) recursively. We now demonstrate the c-trees construction for DNA segments with the following example.

Example 2. Consider the five DNA segments s_0="ACGGT", s_1="CTTAG", s_2="ACGTT", s_3="TAAGC" and s_4="GACGT". When we set q=2 and c=2, the c-signatures are: $sig^2(s_0)$=($\underline{1001}$ $\underline{0200}$), $sig^2(s_1)$=($\underline{0101}$ $\underline{0011}$), $sig^2(s_2)$=($\underline{1001}$ $\underline{0101}$), $sig^2(s_3)$=($\underline{1100}$ $\underline{1010}$), $sig^2(s_4)$=($\underline{1001}$ $\underline{1100}$). If $\ell = 4$, we get $\frac{4^q}{c\ell}$=2 trees. The first tree T_0 is constructed from the c-signature strings $sig_0^2(s_i)$,

$0 \leq i \leq 4$, and the tree T_1 is constructed from $sig_1^2(s_i)$, $0 \leq i \leq 4$. The c-trees (T_0, T_1) for the five DNA segments are shown in Fig. 1.

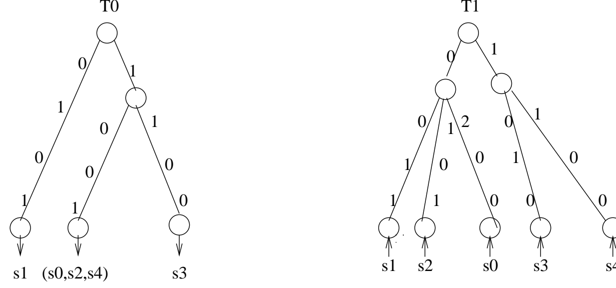


Fig. 1. The c-trees for the DNA segments

5 Query Processing

In this section, we present how to use the two-level index to get the candidates by pruning data segments that are far away from the query sequence. Then the dynamic programming is conducted to obtain the final alignments with high alignment score between the candidates and query sequence. This phase is a standard procedure, so we just skip the details about it in this paper. Before sequence similarity search begins, a hash table HT and the c-trees are built on the DNA segments. The query sequence Q is also partitioned into $|Q| - \omega + 1$ sliding query patterns $q_1, \ldots, q_{|Q|-\omega+1}$.

5.1 The First Level Filter: Hash Table Based Similarity Search

The query pattern q_i is first encoded to a hash key h_i, which is a λ bit integer. Then all the encoded neighbors $ngbr$ of the hash key h_i are enumerated, and the neighbors are those λ bit integers encoded from the segments which are within a small edit distance from q_i. In [3], an approach has been proposed to enumerate a segment's neighbors. The main idea is also applicable for our current case, but the difference is that we need to consider the impact on the q-grams to get the encoded neighbors when some edit operations are conducted on the segment. d edit operations on segment s will result in at most dq q-grams which are different from those in s, and the new neighboring key will be computed in terms of the new group of q-grams by using the coding function. In our case, d is set as 3.

Once an encoded neighbor e_{ngbr} of q_i is enumerated, the segments in the bucket $HT[e_{ngbr}]$ of the hash structure HT will be retrieved as candidates and stored into the candidate set C_{ht}.

5.2 The Second Level Filter: The c-trees Based Similarity Search

The candidate segments C_{ht} generated from the first level filter will be further verified by the c-trees. According to the c-trees structure, the c-signature $sig^c(q)$ of query q is

divided into δ c-signature strings which are $sig_i^c(q)$, $0 \leq i < \delta$. Algorithm 2 shows how to retrieve the segment s which satisfies the range constraint $ed(q, s) \leq \varepsilon$ for a query segment q. For clarity, threshold γ in Algorithm 2 is set as $2q\varepsilon$, where q is the q-gram length and ε is the edit distance allowed between the DNA data segment and query segment.

In Algorithm 2, $w_i[lNode]$ is used to denote the distance between $sig_i^c(q)$ and the path label $pl = label[\langle root_i, lNode\rangle]$ from the respective root $root_i$ to $lNode$ in T_i, namely $w_i[lNode] = SDist(sig_i^c(q), pl)$. We use $score[s]$ to denote the partial distance for segment s during similarity search. Also for notation simplicity, we use $sig_i^c(s)$ as its corresponding path label for a leaf node in $T_i, 0 < i < \delta$, since each $sig_i^c(s)$ can only be mapped to one path or one leaf node in T_i.

Algorithm 2 Similarity Search Algorithm
Input: The c-trees $(T_0, T_1, ..., T_{\delta-1})$ on $\mathcal{D}$, query c-signature $(sig_0^c(q), ..., sig_{\delta-1}^c(q))$, Candidate segments C_{ht}, distance γ. **Output:** Candidate set C

1: $C \leftarrow \emptyset$
2: **for** $lNode \in T_0$ **do**
3:   **if** $w_0[lNode] < \gamma$ **then**
4:     $E_0^{'}[lNode] = E_0[lNode] \cap C_{ht}$; $\quad C \leftarrow C \cup E_0^{'}[lNode]$
5:     **for** each $s \in E_0^{'}[lNode]$ **do**
6:       $score[s] \leftarrow w_0[lNode]$
7:     **end for**
8:   **end if**
9:   **return** Search($\{T_1, \ldots, T_{\delta-1}\}$, C)
10: **end for**
11:
12: Function: $Search(TSet, C)$
13: **if** $TSet = \emptyset$ **then**
14:   **return** C
15: **else**
16:   $T_i \leftarrow$ first entry in $TSet$
17:   **for** each $s \in C$ **do**
18:     **if** $w_i[sig_i^c(s)] + score[s] \leq \gamma$ **then**
19:       $score[s] \leftarrow w_i[sig_i^c(s)] + score[s]$
20:     **else**
21:       $C \leftarrow C - \{s\}$
22:     **end if**
23:   **end for**
24:   **return** Search($TSet$-$\{T_i\}$, C)
25: **end if**

During query processing, for each leaf node $lNode$ in the tree T_0, the distance $w_0[lNode]$) between the path label of $lNode$ and $sig_0^c(q)$ are computed. And the initial candidate set C includes those segments in $E_0[lNode] \cap C_{ht}$ where $w_0[lNode] \leq \gamma$. For the trees $\{T_1, \ldots, T_{\delta-1}\}$, candidates will be pruned based on the partial distance gradually. For each candidate s in C, we can find its corresponding leaf node $lNode$ with label $sig_i^c(s)$ in $T_i (i \neq 0)$ in time O(1) with links constructed during tree construction, and the partial distance $score[s]$ can be computed as well.

5.3 The Space and Time Complexity Analysis

In this section, the space and time complexity are analyzed for the two-level index structure. For the space complexity of the hash table, we need $O(2^\lambda)$ for the table head. For the bucket of the table, segments will contribute space $\Theta(|\mathcal{D}|/\omega)$. Thus, the total space complexity for the hash structure will be $O(2^\lambda + |\mathcal{D}|/\omega)$. Each neighboring of the segment can be generated with time amortized complexity $O(1)$. Thus, the time complexity for the query is $O(|q|)$.

Essentially, the space complexity for the c-trees can be divided into two portions: the c-trees themselves, and space occupied by the $E_0[*]$ and links. According to the algorithm, $E_0[*]$ must be stored for the first tree; thus they require $O(|\mathcal{D}|/\omega)$ space. The height of each tree is bounded by $O(4^q/(\delta c))$, thus for each tree, the storage required for the edge labels is bounded by $O((c+1)^{4^q/(\delta c)} \log(c+1))$ for each tree. Besides, we also need to maintain the links for the other trees. The space required by links highly depends on the data distribution. Note there are lots of zeros in c-signatures, thus a lot of links will point to a dummy leaf (by dummy we mean that the path label is 0). So we may just compress those links. The time complexity depends on the pruning rate for each iteration. Suppose the filtering rate for each iteration is β, then the total time required to obtain the final candidate set is $O(\delta(c+1)^{4^q/(\delta c)} + (|\mathcal{D}|/\omega)\frac{(1-\beta)(1-(1-\beta)^\delta)}{\beta})$ in the worst case. Note in practice, the algorithm is much more efficient since we do not need to traverse the whole structure most of the time.

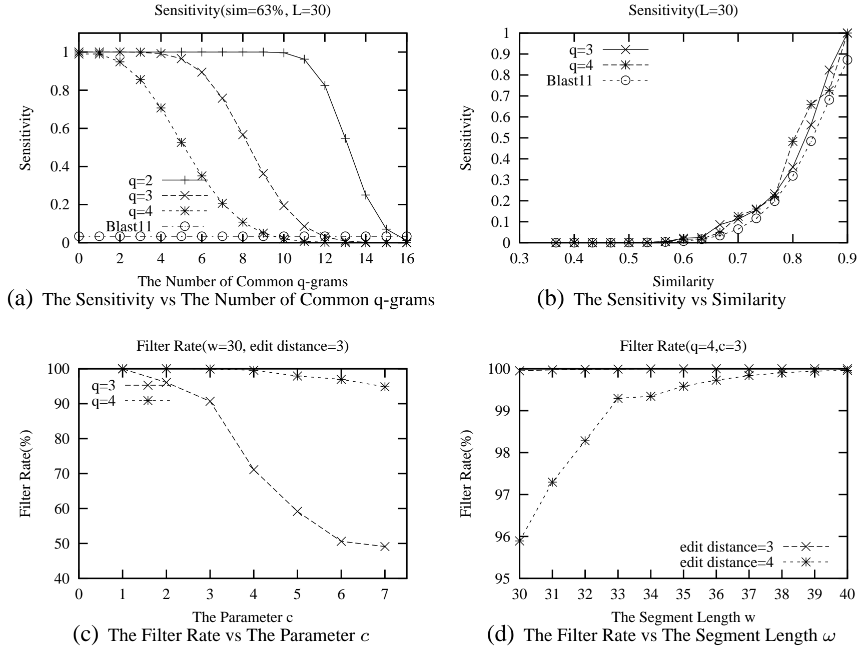
6 Experimental Studies

We evaluate the sensitivity, effectiveness and efficiency of our search method and compare it to the latest version of BLAST(NCBI BLAST2). For BLAST, we set the length of the seed as 11 in the experiment.

6.1 The Sensitivity Analysis

The key issue for the entire search approach is to find a trade-off between sensitivity and effectiveness while maintaining search efficiency. Sensitivity can be measured by the probability that a high score alignment is found by the algorithm. We define the sensitivity analysis problem as: Given a pair of genomic sequences with length L and the similarity ratio sim, compute the sensitivity or probability that they can be detected by the search model.

In the experiment, the sensitivity of the filter model is the probability that the two sequences S and P with length L and similarity sim can be regarded as similar sequences when the number of the common q-grams in S and P is at least ρ. Fig. 2(a) depicts how the number of common q-grams ρ affects the sensitivity of the filter model for L=30, sim=63% and $q = 2, 3, 4$, and compares the sensitivity to BLAST. Given sim=63%, for q=4, it shows that our filter method can achieve higher sensitivity than BLAST as long as the number of common q-grams is no more than 9. Fig. 2(b) shows how the sensitivity of our filter model varies with the similarity sim for L=30. In comparison with BLAST, our filter method for q=3,4 achieves higher sensitivity.



(a) The Sensitivity vs The Number of Common q-grams (b) The Sensitivity vs Similarity

(c) The Filter Rate vs The Parameter c (d) The Filter Rate vs The Segment Length ω

Fig. 2. Experimental Results for Sensitivity and Effectiveness

6.2 The Effectiveness Analysis

Two groups of experiments were conducted to measure the effectiveness of the proposed two-level index structure by using the dataset $ecoli.nt$. The filter rate used in the experiment description is defined as the ratio of the total number of hits found to the total number of segments in data sequence.

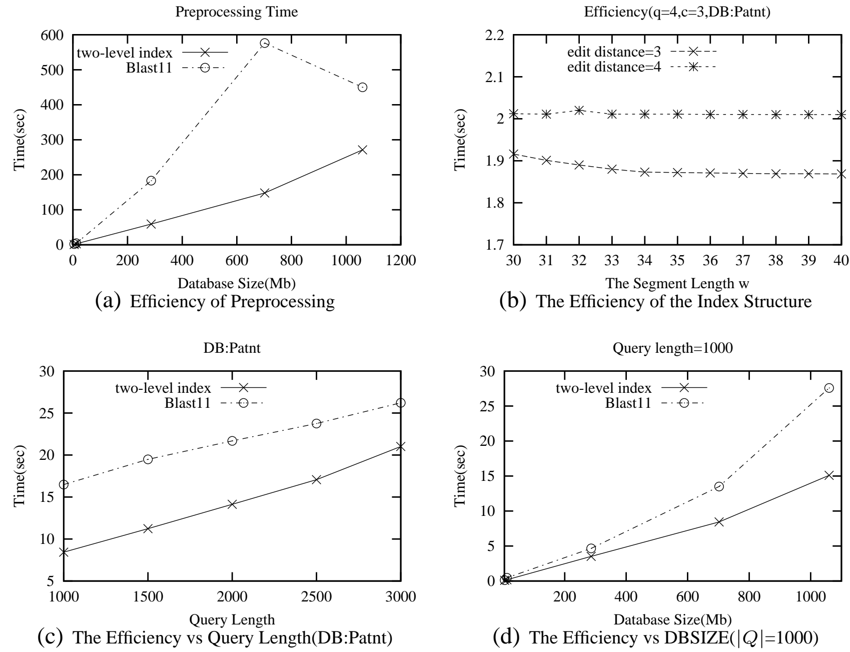
The first group of experiments measure how the parameters c and q affect the effectiveness of filtering when we fix ω=30 and ε=3, in the filter processing. The result in Fig. 2(c) shows that when c increasing, the filter rate drops down as the c-signature representing the segment becomes inaccurate. On the other hand, larger q results in better filter rate since the segment property can be captured more accurately by the c-signatures. The filter rate is 99.9495% for q=4 and c=3. We will use q=4 and c=3 for the efficiency analysis in the following experiment.

The second group of experiments evaluate the effectiveness of the index while varying the segment length ω as well as edit distance ε. The filter rate for different ω and ε is shown in Fig. 2(d). For ω=30 and ε=4, the filter rate can still be as high as 95.895%. All the results show that the proposed index structure is effective for the similarity search.

6.3 The Efficiency Analysis

The five DNA datasets used in the experiments are: other_genomic(1.06GB), Patnt(702.1MB), month.gss(286.2MB), yeast.nt(12.3MB) and ecoli.nt(4.68MB). All the datasets are downloaded from NCBI website.

We first evaluate the efficiency in data sequence preprocessing before performing similarity search in terms of the proposed index structure. Fig. 3(a) shows that pre-



(a) Efficiency of Preprocessing
(b) The Efficiency of the Index Structure
(c) The Efficiency vs Query Length(DB:Patnt)
(d) The Efficiency vs DBSIZE($|Q|$=1000)

Fig. 3. Experimental Results for Efficiency

processing with our method is much faster than that with BLAST due to the efficient algorithm for the hash table and c-trees construction.

The efficiency for searching a single segment is analyzed. The results of the efficiency of segment searching are presented in Fig. 3(b) for q=4 and c=3. We conduct the experiment by varying the segment length ω from 30 to 40, and the edit distance ε from 3 to 4 for the dataset $patnt$. The results show that better performance can be achieved for $\varepsilon = 3$ since it causes better filter rate than $\varepsilon = 4$.

An experiment is also carried out to investigate how the length of the query sequence affects the performance of our method in comparison with BLAST. To do this, we perform similarity search for query lengths of 1000, 1500, 2000, 2500 and 3000 on $patnt$. Fig.3(c) shows our search speed is about twice faster than BLAST when the query length is varied from 1000 to 3000. We performed a comparison of our search method with BLAST on the five datasets as well. We set q=4, c=3, ω=30 and ε=3. The results of the comparison are shown in Fig. 3(d) when the query length is fixed as 1000. The speed-up of the c-trees over BLAST ranges from 2 to 3 for the different size of sequence datasets.

7 Conclusion

We have devised a novel two-level index structure based on q-grams of the DNA sequences which can support efficient similarity search in DNA sequence database. The filter principle with respect to the index structure is presented and it can guarantee that we can achieve efficient sequence searching while keeping the higher sensitivity. We

also carry out the experiments to evaluate the performance of our method in the sensitivity, effectiveness and efficiency, and the results show that our method can efficiently detect the regions in DNA sequence database which are similar to the query sequence with high sensitivity.

References

1. S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman. A basic local alignment search tool. In *Journal of Molecular Biology*, 1990.
2. S. Burkhardt, A. Crauser, P. Ferragina, H. P. Lenhof, and M. Vingron. q-gram based database searching using a suffix array (quasar). In *Int. Conf. RECOMB*, Lyon, April 1999.
3. X. Cao, S.C. Li, B.C. Ooi, and A. Tung. Piers: An efficient model for similarity search in dna sequence databases. *ACM Sigmod Record*, 33, 2004.
4. E. Giladi, M. Walker, J. Wang, and W. Volkmuth. Sst: An algorithm for searching sequence databases in time proportional to the logarithm of the database size. In *Int. Conf. RECOMB*, Japan, 2000.
5. E. Hunt, M. P. Atkinson, and R. W. Irving. A database index to large biological sequences. In *International Journal on VLDB*, pages 139–148, Roma, Italy, September 2001.
6. P. Jokinen and E. Ukkonen. Two algorithm for approximate string matching in static texts. In *Proc. of the 16th Symposium on Mathematical Foundataions of Computer Science*, pages 240–248, 1991.
7. T. Kahveci and A. Singh. An efficient index structure for string databases. In *Proc. 2001 Int. Conf. Very Large Data Bases (VLDB'01)*, Roma, Italy, 2001.
8. B. Ma, J. Tromp, and M. Li. Patternhunter: faster and more sensitive homology search. *Bioinformatics*, 18:440–445, 2002.
9. U. Manber and G. Myers. Suffix arrays: a new method for on-line string search. *SIAM Journal on Computing*, 22:935–948, 1993.
10. C. Meek, J.M. Patel, and S. Kasetty. Oasis: An online and accurate technique for local-alignment searches on biological sequences. In *Proc. 2003 Int. Conf. Very Large Data Bases (VLDB'03)*, pages 910–921, Berlin, Germany, Sept. 2003.
11. S. Muthukrishnan and S.C. Sahinalp. Approximate nearest neighbors and sequence comparison with block operation. In *STOC,Portland, Or*, 2000.
12. O. Ozturk and H. Ferhatosmanoglu. Effective indexing and filtering for similarity search in large biosequence datsbases. In *Third IEEE Symposium on BioInformatics and BioEngineering (BIBE'03)*, Bethesda, Maryland, 2003.
13. W.R. Pearson and D.J. Lipman. Improved tools for biological sequence comparison. *Proceedings Natl. Acad. Sci. USA*, 85:2444–2448, 1988.
14. T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Molecular Biology*, 147:195–197, 1981.
15. Z. Tan, X. Cao, B.C. Ooi, and A. Tung. The ed-tree: an index for large dna sequence databases. In *Proc. 15th Int. Conf. on Scientific and Statistical Database Management*, pages 151–160, 2003.
16. P. Weiner. Linear pattern matching algorithms. In *Proc. 14th IEEE Symp. On Switching and Automata Theory*, pages 1–11, 1973.
17. H.E. Williams and J.Zobel. Indexing and retrieval for genomic databases. *IEEE Transactions on Knowledge and Data Engineering*, 14:63–78, 2002.