An Interactive Search Technique for String Databases

Tamer Kahveci Ambuj K. Singh
Department of Computer Science
University of California, Santa Barbara, CA 9310 6
{tamer,ambuj}@cs.ucsb.edu

Abstract

The explosive growth of string databases makes similarity search a challenging problem. Current search tools are non-interactive in the sense that the user has to wait a long time until the entire database is inspected. We consider the problem of interactive searching, and propose a set of innovative k-NN (k-Nearest Neighbor) search algorithms. We propose a new model for the distance distribution of a query to a set of strings. Using this distribution, our first technique orders the MBRs (Minimum Bounding Rectangles) of an index structure based on their order statistics. We also propose an early pruning strategy to reduce the total search time for this technique. Our second technique exploits existing statistical models to define an order on the index structure MBRs. We also propose a method to compute the confidence levels for the partial results. Our experiments show that our technique can achieve 75% accuracy within the first 2.5-35% of the iterations and 90% accuracy within the first 12-45% of the iterations. Furthermore, the reported confidence levels reflect the quality of the partial results accurately.

1 Introduction

With the growth of the Internet, the success of data-centric applications depends on how fast they allow access to relevant results, rather than on how fast the entire result set is computed. This is especially true when the user is interested in browsing through different data collections to decide on a subset of collections for targeting more demanding and exhaustive queries.

The complexity and the volume of scientific databases have been growing rapidly. For example, the size of the genome database of National Center for Biotechnology Information has doubled every 15 months [10], and it is growing even faster in recent years. Hundreds of Megabytes of video data are recorded everyday by companies like ABC, CNN, and CNET. AT&T generates upwards of 60MB of network flow information every day. There are similar complex and large datasets generated on account of the Sloan Sky Survey [1], protein structure databases [2], pathway databases [3], and physics experiment datasets [4]. It is obvious that supporting interactive searches for such complex and voluminous datasets, although urgently needed, is a difficult task. One needs to develop appropriate models for these novel datasets, so that statistics can be defined and used during interactive querying.

In this paper, we focus on supporting interactive access to one such complex scientific dataset, that of genome sequences. Genome databases are being used by researchers for drug design, medical care, phylogenetic analysis, evolutionary analysis, personalized medicine, and many other applications. Searching genome databases is a difficult problem for a number of reasons: i) There is an explosive growth in the size of genome databases. ii) These databases are also distributed, requiring that meaningful data collections be identified before exhaustive queries are posed [16]. iii) The similarity measures are complex. Therefore, CPU and memory demands of these databases slow them down considerably for long queries. For example, comparing a mouse DNA sequence against

the entire human genome database can take days with currently available search tools. There is a need for new index structures and search algorithms which can estimate the useful regions of a database fast and accurately, and define good query execution plans when the data is distributed.

Our contribution is the development of two interactive substring search techniques for k-NN (k-Nearest Neighbor) queries. The first technique approaches this problem by transforming the information from string space into vector space through the use of *frequency vectors* (count of each letter in a string) [25]. The frequency vectors are then clustered into MBRs (Minimum Bounding Rectangles). We identify a representative frequency vector for each MBR. Then, we estimate the distance distribution of the substrings contained in each MBR to the query substrings using the representative frequency vector. Based on these distributions, we calculate the *order statistics* [14]. The computation of this distribution is complicated by the fact that strings can use non-traditional distance measures such as weighted edit distances.

Our k-NN algorithm hierarchically finds the next MBR of the index structure in ascending order of the mean of their k^{th} order statistics. Later, these MBRs are inspected iteratively using a highly optimized search tool, such as BLAST [7], and the intermediate results are reported to the user. We reduce the total search time of this technique by pruning the MBRs on the fly that do not contain results better than the ones found so far.

Our second technique uses available statistical theory for string databases [27] to estimate the score of the maximal alignment. BLAST uses this theory to estimate the quality of a result once an alignment is determined. We use the same model to predict which database blocks contain good results prior to the actual alignment.

We also develop a novel technique to estimate the quality of the partial results. At any intermediate step, this technique computes the probability that an uninspected MBR does not contain any results better than the partial results reported so far. Later, these probabilities are multiplied for all the uninspected MBRs. This value is then reported as a *confidence estimate*.

The methods developed in this paper can be generalized to other novel scientific databases as long as we can model the distance distribution (with respect to a given query) in each subspace (MBR) of a given dataset. Order statistics can then be computed and interactive searches along with confidence estimates can be provided. This general strategy and its specific deployment for genome databases (development of models and statistics for DNA and protein sequences) are the two main contributions of this paper.

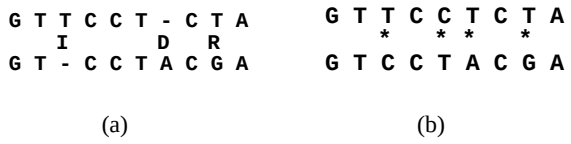
Experimental results on DNA and protein sequences show that our method reports results with high accuracy within a reasonable time: Our techniques achieve 75% accuracy within the first 2.5-35% of the iterations and 90% accuracy within the first 12-45% of the iterations. Our pruning strategy can prune up to 38% of the database.

The rest of the paper is as follows. Section 2 presents the background information on string searching and existing index structures. Section 3 discusses the frequency vectors and distance distributions based on these frequency vectors. Section 4 presents our first interactive substring search technique. Section 5 discusses our second interactive substring search tool based on BLAST's statistical model. Section 6 presents experimental evaluation. We conclude with a brief discussion in Section 7.

2 Background

2.1 Background on string searching

A string s_1 can be transformed into another string s_2 by using three *edit operations*, namely *insert*, *delete*, and *replace*, on individual characters of the string s_1. The *edit distance* between two strings is generally defined as the minimum number of edit operations to transform one string into the other. Figure 1(a) shows the edit distance between DNA strings GTTCCTCTA and GTCCTACGA. Three edit operations are sufficient to transform one of these strings to the other: insert T at the third position, delete A at sixth position, and replace G with T at eighth position. A special case of edit distance, *Gapless Edit Distance (GED)* or *Hamming Distance*, can be defined as follows.



(a) (b)

Figure 1. Alignment of DNA strings GTTCCTCTA and GTCCTACGA under (a) Edit distance (b) Gapless edit distance. The symbols I, D, and R represent insert, delete, and replace operations respectively. The star symbol (*) represent the mismatching characters in gapless alignment. The edit distance is three, while the gapless edit distance is four for this string pair.

Definition 1 *Let q and s be two strings of the same length. Let $q[i]$ be the i^{th} character of q. We define the* Gapless Edit Distance, GED(q, s)*, as the number of character pairs $(q[i], s[i])$ for which $q[i] \neq s[i]$.*

Figure 1(b) depicts the Gapless Edit Distance between DNA strings GTTCCTCGA and GTCCTACGA. These strings contain four mismatching characters: at the third, fifth, sixth, and eighth positions.

GED is an upper bound to the edit distance, and is widely used for statistical analysis. For example, BLAST, a widely used genome search tool, uses *GED* to determine the probability of having a match of a certain score for two random strings of prespecified lengths. This value is called the *E-value* of a match. Many computational experiments [8] show that results developed based on *GED* also extend to edit distances. This is also confirmed by analytical results [9]. We use *GED* in our analysis in Section 3.

Similarity between strings can also be computed based on scores. An *alignment* of strings s_1 and s_2 is obtained by matching each character of s_1 to a character in s_2 in increasing order. All the unmatched characters in both strings are matched with space. Each character pair is assigned a score based on their similarity, and these values are stored in a *score matrix*. Each gap incurs a (potentially large) *gap open penalty* for the first space and a (smaller) *gap extend penalty* for all the spaces. The value of an alignment is defined as the sum of the scores of all of their character pairs.

Example 1 *The default scoring scheme for nucleotides in BLAST assigns +1 for a match, -3 for a mismatch, -5 for gap open, and -2 for gap extend. Consider an alignment that has 7 matches, 1 mismatch, and a gap of length 2. Under this scoring scheme, the score of this alignment is*
$7 \cdot (+1) + 1 \cdot (-3) + ((-5) + 2 \cdot (-2))$ = *-5.* □

Global alignment (or *whole matching*) of s_1 and s_2 is defined as the the maximum valued alignment of s_1 and s_2. *Whole-substring matching* of s_1 and s_2 is defined as the highest valued alignment of s_1 and all the substrings of s_2. *Local alignment* [41] (or *substring-substring matching*) of s_1 and s_2 is defined as the highest valued alignment of all the substrings of s_1 and s_2. Both global and local alignments can be determined in $O(|s_1| \cdot |s_2|)$ time and space using dynamic programming [36, 41]. This complexity can be decreased to $O(r \cdot |s_1|)$, where r is the edit distance between these strings [34].

2.2 String alignment tools

FASTA [38] is a heuristic based program that finds similar substrings. It starts by finding exactly matching k-tuples between two strings (k is typically 6 for DNA strings and 2 for protein strings). In the second step, the score of the 10 best exact matches are determined by performing a gapless extension. Next, overlapping regions are combined, and the substrings whose score is less than a given threshold are discarded. Finally, each of the selected substrings are aligned using banded Smith-Waterman algorithm [41].

Unlike FASTA, BLAST [7] extends all the matches obtained in the first phase in both directions until the similarity between the two substrings falls below some threshold. BLS2SEQ [43] utilizes the BLAST algorithm

for pairwise DNA-DNA or protein-protein sequence comparison. Mega-Blast [46] uses a greedy algorithm for aligning DNA sequences instead of traditional dynamic programming techniques. SENSEI [42] extends the initial seeds by 8 base pairs at a time and it can work in both 2 bits per base and 1 bit per base modes.

MUMmer [17] and REPuter [31] use suffix trees to find the maximal unique matches. QUASAR [13] counts the number of common q-grams of subqueries and each of the database blocks using the suffix array [32] (A q-gram is a string of length q.). Muthukrishnan and Sahinalp [33] proposed an index structure for approximate nearest neighbor search based on suffix arrays and a partitioning of the pattern. However, both suffix trees and suffix arrays [32] have two major disadvantages. First, they use extensive amount of memory [23]. Second, they are good for exact matches, but slow for approximate matches.

Jagadish, Koudas, and Srivastava consider the problem of exact matching with wild-card for multidimensional strings in [24]. The authors map all the strings to real space based on their lexical order. Later, these multidimensional points are indexed using R-trees [19]. The strings within a data page are stored using *elided tries* for each dimension. Other approaches to substring searching have also been proposed in [18, 22, 26, 35, 39, 37, 45]

2.3 Frequency vectors & MRS index structure

Let s be a string from an alphabet Σ, where $|\Sigma| = \sigma$. The *frequency vector*, $f(s)$, of s is defined as the σ-dimensional vector whose entries are the number of occurrences of the letters in Σ. The frequency vector of a DNA string has 4 dimensions since the alphabet contains letters A, C, G, and T. The entries of frequency vectors are sorted in an alphabetical order. For example, let q = GTTCCTCTA be a DNA sequence, then $f(q) = [1, 3, 1, 4]$. Similarly, the frequency vector for a protein string has 20 dimensions, and that for a text in English (ignoring punctuation and spaces) has 26 dimensions.

The MRS index structure [25] maintains summary information for database strings at resolutions of powers of two. At a given resolution, w, the summary is constructed by sliding a window of length w on the database string. Each positioning of the window defines a frequency vector and a set of consecutive vectors defines an MBR. The size of this set determines the *capacity* of an MBR. We will utilize the MRS index structure for our search techniques because of its compactness and efficiency. However, our techniques will extend to other index structures that cluster subsequences of database strings.

3 Frequency statistics

In this section, we develop a statistical model for comparison of strings using frequency vectors. In Section 3.1, we consider the simple case where the frequency vectors of two strings are provided. We show how to approximate the *GED* distribution for two frequency vectors using a normal distribution efficiently. Section 3.2 extends this development to the *GED* distribution of a single frequency vector and an MBR corresponding to a collection of frequency vectors. We represent each MBR using a single representative frequency vector and employ order statistics to approximate the *GED* distribution for an MBR.

3.1 GED distribution for two vectors

At the time of a database search, the query string is known but only the frequency vectors for database strings are available in the index structure. Therefore, in order to plan the interactive query, we need to determine the distribution of *GED* between two frequency vectors.

Each frequency vector, v defines an equivalence class, $\mathcal{S}_v$, of the set of strings whose frequency vectors are equal to v. For example, if $v = [3, 0, 0, 1]$, then S_v = {AAAT, AATA, ATAA, TAAA}. Let $v = [v_1, v_2, ..., v_\sigma]$ be a frequency vector, then the size of the equivalence class defined by v is $\frac{(\sum v_i)!}{\prod (v_i!)}$.

It is easily shown that
$min_s\{GED(q,s)|s \in \mathcal{S}_v\} \geq 0$, and
$max_s\{GED(q,s)|s \in \mathcal{S}_v\} \leq max\{|q|, \sum v_i\}$.
The above inequalities bound the extreme values of $\{GED(q,s)|s \in \mathcal{S}_v\}$.

In order to plan queries and compute statistics, we need to know the distribution of $\{GED(q,s)|s \in \mathcal{S}_v\}$. Using the Central Limit Theorem [14], one can prove thatthe *GED* has normal distribution as follows: Let the random variable X_i represent the *GED* achieved by aligning the i^{th} letters of both strings for $1 \leq i \leq |s|$. The *GED* of an alignment can be calculated as the sum of the values of all X_is for that alignment. Since the sampling rate of all the letters is fixed and determined by the frequency vector of that string, the X_is are iid (independent and identical) random variables. Therefore, from Central Limit Theorem, we conclude that the distribution of the *GED* between a query string and the strings in the equivalence class of a frequency vector can be approximated using normal distribution for large $|s|$ (i.e. $|s| > 10$). Hence, one can compute this approximation if the mean and variance of the distribution are known.

Theorem 1 shows that the mean of this distribution can be computed efficiently in $O(\sigma)$ time, where σ is the alphabet size.

Theorem 1 *(*Mean*): Let q be a string from alphabet $\Sigma = \{\alpha_1, \alpha_2, ..., \alpha_\sigma\}$. Let $x = [x_1, x_2, ..., x_\sigma]$ be the frequency vector of q, and $y = [y_1, y_2, ..., y_\sigma]$ be a frequency vector, where $\sum_{i=1}^{\sigma} y_i = |q|$. Let $\mu_{q,y}$ be the mean of the* GED *distribution between q and the strings in $\mathcal{S}_y$, then*

$$\mu_{q,y} = |q| - \frac{\sum_{i=1}^{\sigma} x_i \cdot y_i}{|q|}$$

Proof:
Let X_i be a random variable whose distribution is defined by the number of matching characters of letter α_i in all alignments of q with $s \in \mathcal{S}_y$. Let $P(X_i = k)$ be the probability that the random variable X_i takes the value k, then

$$P(X_i = k) = \begin{cases} \frac{C_k^{x_i} \cdot C_{y_i-k}^{|q|-x_i}}{C_{y_i}^{|q|}} & 0 \leq k \leq min\{x_i, y_i\}, \\ 0 & otherwise. \end{cases}$$

where C_m^n is n choose m. Let μ_{X_i} be mean of X_i, then

$$\begin{aligned} \mu_{X_i} &= \sum_{k=1}^{min\{x_i,y_i\}} k \cdot P(X_i = k) \\ &= \sum k \cdot \frac{C_k^{x_i} \cdot C_{y_i-k}^{|q|-x_i}}{C_{y_i}^{|q|}} \\ &= \sum x_i \cdot \frac{C_{k-1}^{x_i-1} \cdot C_{y_i-k}^{|q|-x_i}}{C_{y_i}^{|q|}} \\ &= \frac{x_i \cdot C_{y_i-1}^{|q|-1}}{C_{y_i}^{|q|}} \\ &= \frac{x_i \cdot y_i}{|q|}. \end{aligned}$$

The mean of the number of matching characters in all alignments of q with $s \in \mathcal{S}_y$ is the mean of $X_1 + X_2 + ... + X_\sigma$. As a result of this,

$$\begin{aligned} \mu_{q,y} &= |q| - \sum_{k=1}^{\sigma} \mu_{X_i} \\ &= |q| - \frac{\sum_{i=1}^{\sigma} x_i \cdot y_i}{|q|} \end{aligned}$$

□

Unlike the mean, the computation of the variance of the sum of n random variables involves calculation of covariance. Theorem 2 develops a lower bound to the variance of the *GED* distribution between a string q and the strings in $\mathcal{S}_v$ for a given frequency vector v.

Theorem 2 *(*Variance*): Let q be a string from alphabet $\Sigma = \{\alpha_1, \alpha_2, ..., \alpha_\sigma\}$. Let $x = [x_1, x_2, ..., x_\sigma]$ be the frequency vector of q, and $y = [y_1, y_2, ..., y_\sigma]$ be a frequency vector, where $\sum_{i=1}^{\sigma} y_i = |q|$. Let $\sigma^2_{q,y}$ be the variance of the* GED *distribution between q and the strings in $\mathcal{S}_y$, then*

$$\sigma^2_{q,y} \geq \sum_{i=1}^{\sigma}(\frac{x_i \cdot y_i}{|q|} \cdot (1 + \frac{(x_i-1)(y_i-1)}{|q|-1} - \frac{x_i \cdot y_i}{|q|})).$$

Proof:
Let X_i, $P(X_i = k)$ and μ_{X_i} be defined as in proof of Theorem 1. Let $\sigma^2_{X_i}$ be variance of X_i. Then

$$\begin{aligned}
\sigma^2_{X_i} &= \sum_{k=1}^{min\{x_i,y_i\}} k^2 \cdot P(X_i = k) - \mu^2_{X_i} \\
&= \sum k^2 \cdot \frac{C_k^{x_i} \cdot C_{y_i-k}^{|q|-x_i}}{C_{y_i}^{|q|}} - \mu^2_{X_i} \\
&= \sum k \cdot x_i \cdot \frac{C_{k-1}^{x_i-1} \cdot C_{y_i-k}^{|q|-x_i}}{C_{y_i}^{|q|}} - \mu^2_{X_i} \\
&= \sum((k-1)+1) \cdot x_i \cdot \frac{C_{k-1}^{x_i-1} \cdot C_{y_i-k}^{|q|-x_i}}{C_{y_i}^{|q|}} - \mu^2_{X_i} \\
&= \sum(k-1) \cdot x_i \cdot \frac{C_{k-1}^{x_i-1} \cdot C_{y_i-k}^{|q|-x_i}}{C_{y_i}^{|q|}} + \mu_{X_i} - \mu^2_{X_i} \\
&= \sum x_i \cdot (x_i-1) \cdot \frac{C_{k-2}^{x_i-2} \cdot C_{y_i-k}^{|q|-x_i}}{C_{y_i}^{|q|}} + \mu_{X_i} - \mu^2_{X_i} \\
&= \frac{x_i \cdot (x_i-1) \cdot C_{y_i-2}^{|q|-2}}{C_{y_i}^{|q|}} + \mu_{X_i} - \mu^2_{X_i} \\
&= \frac{x_i \cdot (x_i-1) \cdot y_i \cdot (y_i-1)}{|q| \cdot (|q|-1)} + \mu_{X_i} - \mu^2_{X_i} \\
&= \mu_{X_i} \cdot (1 + \frac{(x_i-1) \cdot (y_i-1)}{|q|-1} - \mu_{X_i}).
\end{aligned}$$

Let $X = X_1 + X_2 + ... + X_\sigma$, then $\sigma^2_X = \sum_i \sigma^2_{X_i} + 2 \cdot \sum_j \sum_{i<j} Cov(X_i, Y_j)$. Hence, $\sigma^2_X \geq \sum_i \sigma^2_{X_i}$.

If c is a constant number, then the variance of the random variable $Y_i = c + X_i$ is equal to the variance of X_i. Therefore, we conclude that the variance of the number of mismatching characters is equal to that of matching characters since q is fixed. Hence,

$$\begin{aligned}
\sigma^2_{q,y} &\geq \sum_i \sigma^2_{X_i} \\
&= \sum_i (\mu_{X_i} \cdot (1 + \frac{(x_i-1) \cdot (y_i-1)}{|q|-1} - \mu_{X_i})) \\
&= \sum_{i=1}^{\sigma}(\frac{x_i \cdot y_i}{|q|} \cdot (1 + \frac{(x_i-1)(y_i-1)}{|q|-1} - \frac{x_i \cdot y_i}{|q|})).
\end{aligned}$$

□

The lower bound of Theorem 2 can be calculated in $O(\sigma)$ time. Unlike the formula for the mean in Theorem 1, this is a lower bound since it ignores the covariance between random variables X_i. Covariance corresponds to the dependency between random variables. Our experimental results show that the covariance between the random variables defined in the proof of Theorem 2 is much smaller than the mean, making the approximation very close to its actual value.

The following corollary summarizes the development so far.

Corollary 1 *Let q be a query string. Let v be a frequency with equivalence class $\mathcal{S}_v$. Let s be a string randomly drawn from $\mathcal{S}_v$, then*

$$P(GED(q,s) = d) \approx \frac{1}{\sqrt{2 \cdot \pi \cdot \sigma_{q,v}}} \cdot e^{-\frac{(d-\mu_{q,v})^2}{\sigma^2_{q,v}}},$$

where $\mu_{q,v}$ is the mean and $\sigma^2_{q,v}$ is the variance of the GED *distribution.* □

Example 2 *Assume that a query string of length 30 has frequency vector $q = [12, 10, 3, 5]$. Let two strings have frequency vectors $s_1 = [11, 11, 4, 4]$ and $s_2 = [6, 5, 9, 10]$. We would like to predict which of these strings is more*

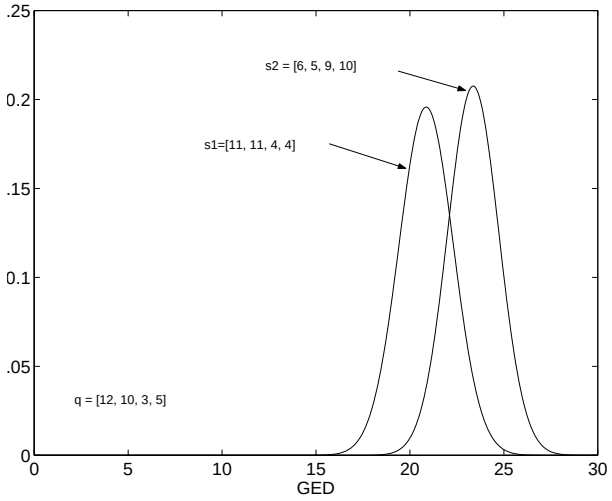


Figure 2. The *GED* distribution for two strings with frequency vectors $s_1 = [11, 11, 4, 4]$ and $s_2 = [6, 5, 9, 10]$ with respect to a query string whose frequency vector is $q = [12, 10, 3, 5]$.

similar to the query string without knowing the actual strings. Intuitively, the obvious answer is that s_1 is more similar, because its counts for each character is similar to that of query. Figure 2 displays the GED *distribution for each of these strings computed as in Corollary 1. Since the mean of the distribution of s_1 is smaller than that of s_2, this figure confirms that s_1 is probably closer to q.*

The distance distribution in Corollary 1 is formulated for string databases. For other types of scientific databases, one needs to develop a formula for the distance distribution between two objects by using their signatures. This formula depends on the type of the database and the signature used to represent the data objects.

3.2 GED distribution of a query to an MBR

So far, we considered the distribution of the distance between a query string and a database frequency vector. Now, we consider the distribution of the distance between a query and a set of database frequency vectors, specifically those contained in an MBR of an index structure. This approximation poses two problems. First, it requires knowing the *GED* distributions of all frequency vectors contained in the MBR. Second, an MBR does not store the individual frequency vectors contained in MBRs. It only maintains their span.

These two problems are resolved by choosing a representative frequency vector for each MBR and assuming that an MBR contains a number of iids based on the representative frequency vector.

The representative frequency vector of an MBR has to be the vector which is closest to the rest of the vectors in the MBR. Furthermore, the sum of the entries of this vector must be equal to w, where w is the resolution of the MBR. This vector is found in two steps as follows: First, the centroid of the MBR is calculated as the average of the lower and higher coordinates of the MBR. Later, this centroid is projected onto the plane which contains frequency vectors for resolution w. Formally, the representative frequency vector of an MBR is defined as follows.

Definition 2 *Let L and H be the lower and higher end points of an MBR, B, of resolution w, then the representative frequency vector of B, v_B, is*

$$v_B = \frac{L+H}{2} - \frac{(\frac{L+H}{2} - W) \cdot N}{||N||_2^2} \cdot N,$$

where $N = [1, 1, ..., 1]$ *and* $W = [w, 0, 0, ..., 0]$. □

Once the representative frequency vector of an MBR has been chosen, its *GED* distribution with respect to the given query can be estimated as explained in Section 3.1. If the MBR contains c frequency vectors at resolution w, then the *GED* distribution of the query to the MBR is approximated by assuming $\lceil c/w \rceil$ iids at the representative frequency vector. This is justified since the MBRs of the MRS index structure contains $\approx c/w$ independent substrings.

The *GED* distribution between a frequency vector and a set of iid frequency vectors can be calculated using order statistics. The k^{th} order statistics is defined as follows.

Definition 3 *Let* X_1, X_2, ..., X_n *be iid random variables, where* n *is a positive integer. Let the sequence* Y_1, Y_2, ..., Y_n *be the values these random variables take sorted in ascending order. The* k^{th} *member of this sequence (i.e.,* Y_k*) is defined as the* k^{th}*-order statistics of this sequence.*

The extreme values Y_1 and Y_n are also called the *MIN* and *MAX* values of this sequence. Note that, the k^{th}-order statistics, Y_k, is actually a random variable. It is shown in [14] that the cumulative density function of this variable can be calculated as

$$P(Y_k \leq d) = \sum_{j=k}^{n} C_j^n \cdot P(X \leq d)^j \cdot (1 - P(X \leq d))^{n-j},$$

and the probability density function can be computed as

$$P(Y_k = d) = \\ k \cdot C_k^n \cdot P(X \leq d)^{k-1} \cdot (1 - P(X \leq d))^{n-k} \cdot P(X = d),$$

where X is a random variable having the same distribution as X_is ($1 \leq i \leq n$).

Let M_k be the number of X_is ($1 \leq i \leq n$ and $0 \leq k$) which takes value less than or equal to k, then it is proven in [14] that

$$P(M_k \leq m) = \sum_{j=0}^{m} C_j^n \cdot P(X \leq d)^j \cdot (1 - P(X \leq d))^{n-j}.$$

Using these formulas for order statistics, one can achieve the combined *GED* distributions for all the frequency vectors in an MBR.

The distance distribution between a query object and a set of data objects can be formulated for any type of database similarly. Such a generalization requires changes in two steps: 1) The representative for a set of data objects must be chosen based on data type and the clustering algorithm used. 2) An appropriate distance distribution function needs to be employed for a data object pair as discussed in Section 3.1.

4 Using frequency statistics for interactivity

In this section, we discuss how to achieve interactivity using the statistical model we built on frequency vectors in Section 3. We utilize available non-interactive substring search tools, and make them interactive. We employ BLAST and the classic Smith-Waterman technique as the non-incremental search tools in this paper. This is because BLAST is the industrial standard and is highly optimized to work efficiently. Smith-Waterman algorithm is also widely used in both string alignment and pattern matching problems. Note that our techniques can be used with other search tools to make them interactive as well.

4.1 Local statistics-based interactive search: LIS

Our interactive search technique uses statistical information about the database substrings to improve the quality of the partial results. We call this technique the *Local statistics-based Interactive Search (LIS)* technique. LIS provides the user partial results along with confidence intervals. As the user waits longer, the results and the confidence intervals are updated iteratively to achieve more precise results.

LIS takes a query string and an *update rate* as input. The *update rate* parameter affects the sensitivity and the interactivity of the algorithm. We will elaborate on this parameter later in this section. The *estimator* defines an ordering of the MBRs of the MRS index structure. Database substrings are then searched iteratively in this order, and results are reported to the user along with confidence intervals.

$/* INPUT \quad q$: *query sequence*
k : *number of nearest neighbors*
R : *root of the MRS index structure.*
Let W be the set of resolutions available in the index structure. */
Algorithm LIS(q, k, R)

- For all $w \in W, w < |q|$
  $q_w :=$ centroid of the MBR that covers the frequency vectors of all substrings of q of length w.
- $M := \emptyset$; // initialize heap
- *HEAP-INSERT(*$M, R, 0$*)*; // insert root node to heap
- While $M \neq \emptyset$
  1. $B :=$ *EXTRACT-MIN(*M*)*;
  2. If B is a leaf level MBR then
     (a) Search q in the string contained in B;
     (b) Update results found so far and the confidence level, and report to the user;
  3. else /* The *Estimator* starts here */
     For all children B_i of B
     - $M_i := \infty$;
     - $v :=$ representative vector of MBR B_i;
     - For all $w \in W, w < |q|$
       (a) $\mu_{q,v} :=$ mean of *GED*$(q_w, \mathcal{S}_v)$;
       (b) $\sigma^2_{q',v} :=$ variance of *GED*$(q_w, \mathcal{S}_v)$;
       (c) M_i+= (k^{th} order statistics$(q_w, \mathcal{S}_v)$)/w;
     - *HEAP-INSERT(*M, B_i, M_i*)*;

Figure 3. Pseudocode of LIS.

Figure 3 presents the pseudocode of LIS. The algorithm takes a query string q, number of nearest neighbors k and the root of the MRS index structure as input. The algorithm starts by constructing an MBR that covers the frequency vectors of all possible query substrings of length w, for all resolutions in the MRS index structure that is less than $|q|$. Later, the centroid of these MBRs are determined. These centroids represent the query string at different resolutions. The algorithm maintains a min-heap which initially contains only the root node. Later, the MBRs in the min-heap are extracted one at a time (step 1). If current MBR is a leaf node, then q is aligned with the string in that MBR (step 2.a). The results are then updated, and reported to the user along with confidence level (step 2.b). Confidence level exhibits *how confident the user should be with the current partial results*. We will discuss the computation of the confidence levels later in this section. If the current MBR is not at leaf level, then LIS goes into *estimator* mode (step 3). The estimator iteratively inspects all the children of the current node. At each iteration, the corresponding database region is inspected at all possible resolutions; mean and variance

Dataset	Size	Index Size	Num. of Random I/Os
chr. 18	4.2M	17M (135K)	3.4M (4.2K)
chr. 21	34M	138M (1M)	33M (34K)

Table 1. The sizes of the index structures and the number of random disk I/Os for *chromosome 18* and *chromosome 21* (human genome) for R-tree and the MRS index structure. The values for the MRS index structure are shown in parentheses.

of the *GED* distribution are evaluated for each resolution as presented in Theorems 1 and 2 (Steps 3.a and 3.b). Using these values, the expected value of the k^{th} order statistics is then computed based on the formulas presented in Section 3.2 and accumulated by multiplying with a weight of $\frac{1}{w}$(Step 3.c). Finally, this MBR is inserted into min-heap along with the expected value of the k^{th} order statistics.

Step 2 takes $O(\log n)$ time per MBR. Steps 3.a through 3.c can be computed in $O(\sigma)$ time, where σ is the alphabet size. The for loop is iterated once per MBR. Therefore, the complexity of Step 3 (except last line) is $O(N \cdot \sigma)$, where N is the number of MBRs. Heap insertion can take up to $O(\log n)$ time per MBR. Therefore, step 3 takes $O(N \cdot \log N)$ time. Let D be the size of the database and c be the box capacity, then the time complexity of LIS (except the actual search phase in step 2.a) is $O(\frac{D}{c} \cdot (\log(\frac{D}{c}) + \sigma))$.

Confidence intervals are computed based on the results discovered so far and the *GED* distributions of the uninspected MBRs as follows. Let d_k be the distance to the k^{th} closest string reported so far. $P(Y_1 \geq d_k)$ for an MBR represents the probability that that MBR does not contain any match whose distance is less than d_k. This value can be easily computed for each uninspected MBR using the formulas given in Section 3.2. Later, these results are multiplied to find the probability that the k^{th} closest distance in the remaining MBRs is not better than the k^{th} closest distance found so far. We define the confidence formally as follows.

Definition 4 *Let d_k be the distance to the k^{th} closest string reported so far, where $k > 0$ is an integer. Let B_1, B_2, $\cdots$, B_r be the uninspected MBRs. Let $P_{B_i}(Y_1 \geq d_k)$ represent the probability that B_i does not contain any match whose distance is less than d_k, for $1 \leq i \leq r$, then the confidence is defined as*

$$\text{confidence} = \Pi_{i=1}^{r} P_{B_i}(Y_1 \geq d_k).$$

The *update rate* parameter is a real number in the interval $[\frac{1}{N}, 1]$. It defines the number of MBRs searched per iteration. Two extreme cases for update rate are $\frac{1}{N}$ and 1. If update rate is $\frac{1}{N}$, then only one MBR is processed per iteration. For small update rates, LIS has higher interactivity; the initial results are reported to the user faster, and the results are updated more frequently. On the other hand, the confidence in the results is lower. If the update rate is 1, then the technique converges to traditional non-interactive search techniques (i.e. the entire database is searched before reporting any result).

Currently, we have a beta-version of LIS with BLAST running at `http://amazon.cs.ucsb.edu:8080/sequence.php`. A snaphot of this fron page of this server is given in Figure 4 The user selects a target dataset among a set of DNA datasets and enters a query. The beta-version of the program considers moderately sized queries (i.e. queries that have 4K to 10K nucleotides.). This is because we employed only a small subset of index structures on the online version for now. Furthermore, moderately sized queries are sufficient to observe the difference in response time of the interactive and the non-interactive versions of BLAST. Interested readers are welcome to compare the original version of BLAST (by NCBI) and our interactive version of BLAST.

4.2 Analysis of LIS

We utilize the MRS index structure to cluster the frequency vectors of database substrings for several reasons. 1) The size of the MRS index structure is very small (typically 1-2 % of the database size.). 2) The index structure

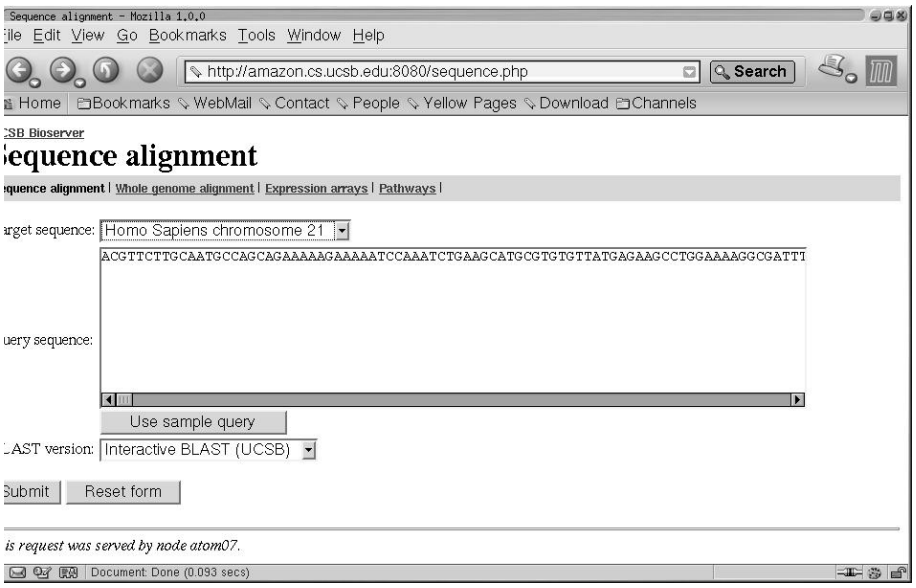


Figure 4. A snapshot of our LIS program on UCSB Bioserver.

stores information at multiple resolutions. 3) Since the MBRs of the index structure are generated by sliding a window on the database string, the union of all the substrings contained in an MBR are located in a single contiguous block in the database string. Therefore, reading the contents of an MBR does not require more than one random disk seek. Using R-tree [19] or some other similar multidimensional indexing techniques [11, 12, 15, 21, 29, 40, 44] has two major drawbacks. First, pointers to the corresponding substrings must be stored along with the frequency vectors in the index structure. As a result of this, the size of the index structure increases dramatically. Second, postprocessing incurs a high amount of random disk I/O since the frequency vectors in an MBR can correspond to database substrings in any order. Table 1 displays the dramatic difference between R-tree and the MRS index structure for two datasets.

The presented technique scales to large datasets since statistics are maintained at varying resolutions (corresponding to different window sizes), and for a given resolution, the order statistics are inspected hierarchically.

One possible way to improve LIS is to use the average of the frequency vectors instead of the centroid as the representative of an MBR. In our experiments, we had the same results when we used the average of the frequency vectors as the representative. Using the average of the frequency vectors has one disadvantage over centroid: The space requirement of the MRS index structure increases by 50% since one more value is stored per dimension for each MBR. Therefore, we will use the centroid as the representative. Other ordering techniques based on *MINDIST* (i.e. minimum distance to the MBRs), or distance to the centroid, or *MAXDIST* did not perform as well.

Although we use the MRS index structure for clustering the frequency vectors, our theory can be extended to other index structures and other kinds of datasets by modifying three steps of LIS: 1) Cluster data blocks using an appropriate technique. 2) Find small signatures for each block. A signature must represent all the objects in the corresponding block. 3) Predict the distance distribution between two objects using only their signatures.

4.3 Reducing the search cost of LIS

LIS ranks database substrings according to their similarity to the given query string in the frequency domain. These substrings are then examined in the computed order, thus potentially finding better results earlier. However, LIS still has to search the whole database in order to ensure the absence of false dismissals. The total search time of LIS can be reduced by pruning the lower ranking substrings that do not contain any result better than the

k^{th}-NN found so far. We propose to prune low quality substrings in two steps:

1. Let S_k be the score of the k^{th} best match found so far. An upper bound, L, on the length of the k^{th} best match is estimated. Define w to be the minimum resolution available in the MRS index structure, for which $w \geq L$.

2. An upper bound, S_{upper}, to the score of the best alignment between the query and database substrings contained in the next uninspected MBR at resolution w is computed. If $S_{upper} < S_k$, then this MBR is pruned. Otherwise, it is inspected for alignments.

These two steps are inserted at the beginning of the *for loop* in Step 3 of the LIS algorithm in Figure 3. The resulting method is called *LIS-prune* Next, we discuss the implementation of Steps 1 and 2. Step 1 is based on the following lemma:

Lemma 1 *An alignment of length n with a non-negative score in BLAST's default scoring scheme contains at most $n/3$ indels (insert/delete). Similarly, such an alignment contains at most $n/4$ mismatches.*

The proof of Lemma 1 follows from BLAST's default scoring scheme for matches, mismatches, and indels (see Example 1). This lemma implies that the total length of the alignment (including gaps) is at most $\approx 33\%$ more than the number of matches. Let S be the score of an alignment in BLAST's default scheme. If we approximate the number of matches of an alignment as the score of the alignment, then the total length of the alignment is at most $S \cdot 4/3$. This length provides us the appropriate resolution at which to carry out Step 2 of pruning[1].

Once we know the resolution at which to inspect the MBRs for pruning, we examine each unprocessed MBR B. We compute an upper bound to the score of the best alignment (in the affine gap model) of a query string q to the strings contained in B. This computation is shown in Figure 5. The algorithm takes a frequency vector v and a box B as input. It starts by finding the number of mismatches (Steps 1 and 2). Later, the scores for matches and mismatches are computed (Steps 3 and 4), and the cost of insertions/deletions are added (Steps 5 and 6). Note that the algorithm assumes a single gap for all insertions and deletions in order to maximize the score. In the worst case, $3 \cdot \sigma + 7$ integer additions, 5 integer multiplications and $\sigma + 1$ integer comparisons are sufficient to compute $SCORE_w(v, B)$, where σ is the alphabet size. For DNA strings this number is 19 integer additions, 5 integer multiplications, and 5 integer comparisons regardless of the length of the strings.

5 Using BLAST statistics model for interactivity

The LIS method is based on the statistics of the distribution of the distance (in the frequency domain) between a query and the set of substrings within an MBR. In this section, we will explore a different way to compute statistics and support interactivity. Instead of examining the distribution of substrings within an MBR, we will examine the frequency of different letters (of a given alphabet) within an MBR and use existing BLAST theory [27] to estimate the expected score between a query and a database string with a specified alphabet distribution.

5.1 BLAST statistics model

Let $\Sigma = \{\alpha_1, \alpha_2, \cdots, \alpha_\sigma\}$. Assume that the letters are sampled with probabilities $\{p_1, p_2, ..., p_\sigma\}$ in the data string, and $\{q_1, q_2, ..., q_\sigma\}$ in the query string. Let the score of a match of α_i and α_j be $s(\alpha_i, \alpha_j)$ such that

1. $\sum_{i,j} p_i p_j s(\alpha_i, \alpha_j) < 0$,
2. $max_{i,j}\{s(\alpha_i, \alpha_j)\} > 0$.

[1] A similar reasoning can be made to estimate an upper bound on the length of an alignment in any other scoring scheme.

/* w is the resolution of used to find the score. */
/* v is σ dimensional integer point. */
/* B is σ dimensional integer box of lower and higher coordinates $B.L$ and $B.H$. */
Procedure $SCORE_w(v, B)$

1. $inc := dec := sum := 0$;
2. for i:= 1 to σ
   - if $v[i] < B.L[i]$ then
     inc += $B.L[i] - v[i]$;
     sum += $B.L[i]$;
   - else if $B.H[i] < v[i]$ then
     dec += $v[i] - B.H[i]$;
     sum += $B.H[i]$;
   - else sum += $v[i]$;
3. $ScoreInc :=$
   $(min\{sum, w\} - inc) \cdot S_{match} + inc \cdot S_{mismatch}$;
4. $ScoreDec :=$
   $(min\{sum, w\} - dec) \cdot S_{match} + dec \cdot S_{mismatch}$;
5. if $w < sum$ then
   $ScoreInc$ +=
   $S_{gap\_open} \cdot (sum - w - 1) + S_{gap\_extend}$;
6. else if $sum < w$ then
   $ScoreDec$ +=
   $S_{gap\_open} \cdot (w - sum - 1) + S_{gap\_extend}$;
7. return $min\{ScoreInc, ScoreDec\}$;

Figure 5. Procedure $SCORE_w(v, B)$ for computing the best score of the alignment between a string x and a set of strings $\mathcal{X}$, where w is the resolution used to compute the score, v is the frequency vector of s, and B is the MBR that covers the frequency vectors of the strings in $\mathcal{X}$.

These conditions must be satisfied for any valid scoring scheme. If first condition is not met, then the best matching substring of two random string would always tend to be the whole sequence [28]. The second condition implies that there are at least two letters with a positive score match.

Let $f(\lambda) = \sum_{i,j} p_i p_j e^{\lambda s(\alpha_i, \alpha_j)}$. BLAST uses the unique positive solution, λ^* to the equation $f(\lambda) = 1$ in the statistical computation [27]. Karlin and Altschul [27] show that the expected value for the score of the maximal alignment of two strings can be approximated as

$$\frac{ln(mn)}{\lambda^*},$$

where m and n are the lengths of the two compared strings. Finding this expected value requires the computation of λ^*. We propose to use a direct solution technique based on the following properties:

a) $f(0) = 1$, b) $f'(0) < 0$, c) $f''(0) > 0$,

where f' and f'' are the first and second derivatives of f. These three conditions imply that f is a convex function. One of the solutions of $f(\lambda) = 1$ is obviously at $\lambda = 0$, and the other root is positive. We define $g(\lambda) = f(\lambda) - 1$, and use the Newton-Raphson Method [20] to find the positive root of $g(\lambda) = 0$. In our experiments, this method converged in a few iterations.

5.2 GIS: Interactivity using BLAST statistics

Here, we develop our second interactive search technique based on BLAST statistics [27, 28]. We call this technique *Global statistics-based Interactive Search (GIS)*. Similar to LIS, GIS partitions the database strings into overlapping blocks of length $w+c-1$ with an overlap of $w-1$ letters between consecutive blocks, where w is the window length and c is the box capacity of each block. Each block in this partitioning corresponds to the union of all the substrings contained in an MBR of the MRS index structure. We store the frequency of all the letters for each partition separately.

The score of the maximal alignment for each partition s_i is approximated by $\frac{ln(|s_i|\cdot|q|)}{\lambda_i^*}$. Since the length of substrings in all the partitions are fixed (i.e. $|s_i| = w+c-1$ is fixed), $\frac{ln(|s_i|\cdot|q|)}{\lambda_i^*}$ becomes larger when λ_i^* is smaller.

Given a query string q, GIS works in 3 steps:

1. Compute λ_i^* for q and the substring s_i in partition i using the Newton-Raphson method, for all i.
2. Sort λ_i^* in ascending order.
3. Search the partitions s_i in ascending λ_i^* order using an efficient search tool like BLAST. Report intermediate results to the user for each partition.

The first step of this algorithm calculates λ_i^* for each q and s_i using the Newton-Raphson method. Usually, this method converges to solution within a negligible error in less than 10 iterations. Therefore, the time complexity of this step is $O(n)$, where n is the number of partitions. The sorting step takes $O(n \log n)$ time. Therefore, the total time complexity of the preprocessing step is $O(n \log n)$.

Since BLAST statistics is limited to a small subset of scoring schemes, extending GIS to arbitrary types of databases requires a new statistical model. This can be done in three steps: 1) Split the data space into bins and map objects to bins. 2) Define a score for each bin pair (this scoring scheme must satisfy the requirements of BLAST's statistical model). 3) Find the frequency of each bin for each data block.

6 Experimental results

We used two classes of string datasets in our experiments:

• **DNA dataset** contains chromosome 02 (*chr-02*), chromosome 18 (*chr-18*), chromosome 21 (*chr-21*), and chromosome 22 (*chr-22*) from *homo sapiens* database and the genetic code of *E.Coli* [6]: chr-18 and E.Coli datasets contain more than 4M base pairs, and the other datasets contain more than 30M base pairs.

• **Protein dataset** contains all the proteins in the SWISSPROT database [5]. We created one large protein dataset by appending all the sequences, resulting in a single sequence of length approximately 68M . The alphabet here contains 20 letters.

We downloaded the source code of BLAST, and implemented the MRS index structure for window sizes w = $\{256, 512, 1024, 2048\}$, and box capacity $c = 1000$. We used BLAST for alignments of DNA strings and the standard Smith Waterman algorithm for alignments of protein strings. We implemented LIS, LIS-prune and GIS as discussed in Section 4 and Section 5.

We extracted four query sets from chr-18 dataset for $|q| = \{500, 1000, 2000, 4000\}$ and three query sets from chr-22 dataset for $|q| = \{1000, 2000, 4000\}$ each containing 100 queries. Later, we generated six new query sets from each of these query sets by modifying these queries with 5%, 10%, 20%, 30%, 40% and 50% mutation probability using three edit operations (i.e. insert, delete, and modify). We generated two query sets from the protein dataset for $|q| = \{256, 512\}$. Later, we created three more query sets by modifying these queries with

mutation rates 5%, 10% and 20%. We performed k-NN queries for various values of k using these query sets on a 1.4 GHz AMD Athlon MP+ computer with 1 GB memory.

We used BLAST's default scoring scheme for DNAs in our experiments: score of +1 for a match, -3 for a mismatch, -5 gap open, and -2 gap extend (see Example 1). For protein dataset, we use the standard pattern matching constants: +1 for a match, and -1 for mismatch, insert, and deletes.

We calculate the *accuracy* of the partial results for k-NN queries as the sum of the scores of the k best matches found so far divided by the sum of the scores of the actual best k matches. Formal definition of accuracy is as follows.

Definition 5 *Let S_1, S_2, $\cdots$, S_k be the best k scores found so far, where $k > 0$ is an integer. Let S_1^*, S_2^*, $\cdots$, S_k^* be the actual best k scores, then we define the accuracy as*

$$\text{Accuracy} = \frac{\sum_{i=1}^{k} S_i}{\sum_{i=1}^{k} S_i^*}.$$ □

Intuitively, the accuracy shows *how good the current results are with respect to the actual results*.

We set the *update rate* to $1/N$ in our experiments, where N is the number of MBRs. In other words, we process one MBR at each iteration.

6.1 Varying the mutation rate

Our first experiment set inspects the performance of GIS and LIS for varying proximity of query strings to database. We use the query sets generated from chr-18 in this experiment set. We generate a larger dataset, namely *chr-18/E.Coli* by appending two dissimilar datasets chr-18 and E.Coli, and perform queries on this dataset. The purpose of this experiment is to see whether our techniques can distinguish the distant regions in E.Coli from the homologous regions of chr-18.

Figures 6(a) to 6(b) show the average score found by LIS, LIS-prune, and GIS on 1-NN queries at different iterations for 5%, 10%, 20%, and 30% mutation rates of $|q|$ = 4000 query set. Since chr-18 and E.Coli datasets have approximately same number of base pairs, they have similar number of MBRs. As evident from these figures, all of our techniques find the optimal results before half of the database is inspected. This means that, our techniques can distinguish the distant regions in E.Coli from closer regions in chr-18. This is more evident for smaller mutation rates (i.e. when the query strings are closer to chr-18). As the mutation rates increases to 50%, our techniques find optimal results later. This is because for high mutation rates, the query strings are not close to one of the datasets anymore. Hence, all datasets have similar results.

For small mutation rates, the graphs are more steep at the beginning. This means that our techniques work better when the query string is close to a subset of the database. However, even for high mutation rates, like 20 or 30%, our techniques can find high scoring results before 30% of the database is inspected.

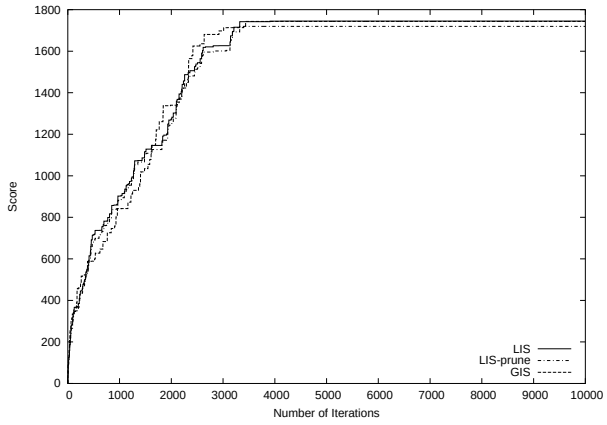
Although there is no clear winner among these three techniques, GIS finds high scoring results slightly faster than LIS and LIS-prune on the average. LIS and LIS-prune overlap for mutation rates greater than 10%.

Table 2 summarizes the number of iterations performed by LIS, LIS-prune, and GIS to achieve accuracies of 0.75 and 0.9 for various mutation rates of the query set. For all mutation rates, all the techniques can obtain 0.9 accuracy after 25-31% of the MBRs are inspected, and 0.75 accuracy after 15-24% of the MBRs are inspected.

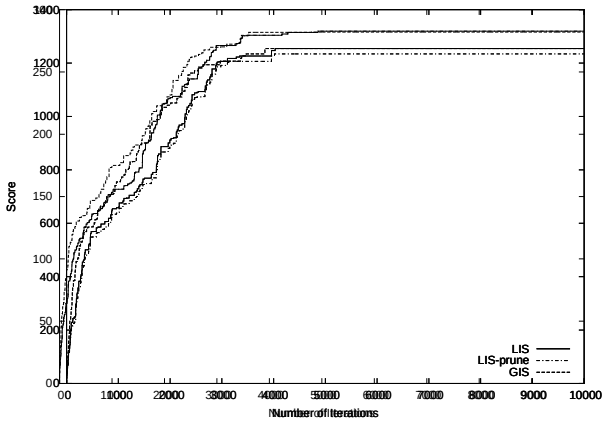
6.2 Varying number of NN

Our second experiment set evaluates our techniques for different number of nearest neighbors. In this experiment, we used the $|q|$ = 4000 query set from chr-18 dataset and modified it with 5% mutation rate. We performed queries on chr-18/E.Coli dataset.

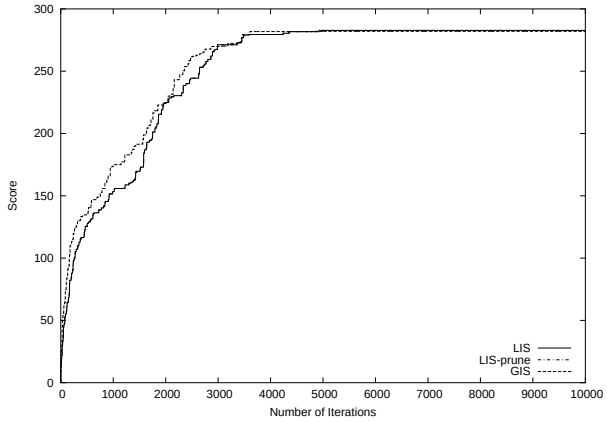
Table 3 displays the number of iterations performed to achieve an accuracy of 0.75 and 0.9. Both LIS and GIS can achieve 0.75 accuracy after only 31-35% of the MBRs are inspected for all values of k. As the number of



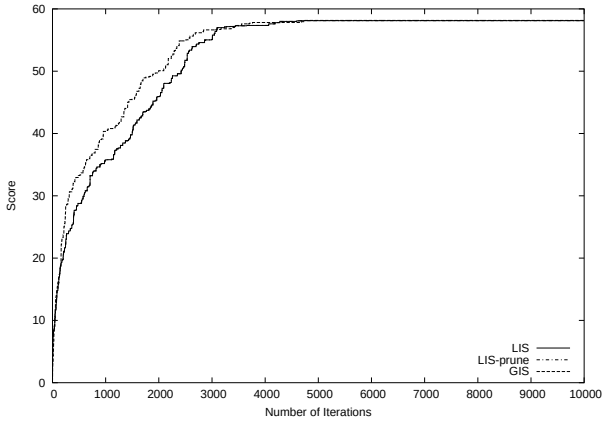
(a) 5% mutation



(b) 10% mutation



(c) 20% mutation



(d) 30% mutation

Figure 6. The score found by LIS, LIS-prune, and GIS at different iterations for query set generated from chr-18 on chr-18/E.Coli dataset.

	Mutation Ratio (%)					
	5	10	20	30	40	50
LIS	2095 (2588)	2123 (2719)	1866 (2739)	1771 (2537)	1903 (2758)	1766 (2917)
LIS-prune	2096 (2560)	2140 (2719)	1866 (2739)	1771 (2537)	1903 (2758)	1766 (2917)
GIS	1846 (2401)	1613 (2347)	1759 (2431)	1362 (2236)	1583 (2834)	1857 (2811)

Table 2. The number of iterations for LIS, LIS-prune, and GIS to achieve 0.75 and 0.9 accuracy for various mutation rates. The number of iterations for non-interactive techniques is 8863. The results for accuracy = 0.9 are shown in parenthesis.

	Number of NN (k)			
	1	2	4	8
LIS	2095 (2588)	1992 (2819)	2167 (3254)	2807 (4372)
LIS-prune	2095 (2560)	1933 (2637)	2196 (3209)	2798 (4336)
GIS	1846 (2401)	1842 (2338)	2336 (3057)	3085 (3790)

Table 3. The number of iterations for LIS, LIS-prune, and GIS to achieve 0.75 and 0.9 accuracy for various number of nearest neighbors. The number of iterations for non-interactive techniques is 8863. The results for accuracy = 0.9 are shown in parenthesis.

nearest neighbors increases, the number of iterations required by LIS to achieve a certain accuracy increases. This is because of two reasons. First, the size of the result set increases with the number of nearest neighbors. Second, as k increases, the scores of the best k matches in the resulting set contains many alignments of similar score. This means that the results become indistinguishable. As a result of this, our techniques can not distinguish the alignments. Note that this is an intrinsic problem for any k-nearest neighbor algorithm in very high dimensions [30]. However, this increase in the number of iterations is not abrupt.

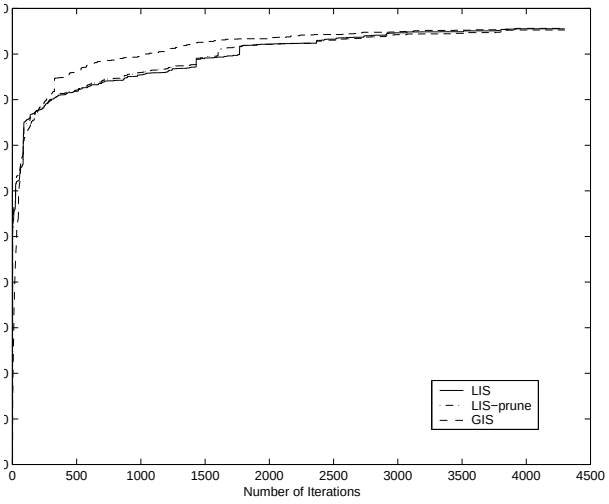


Figure 7. The score found by LIS, LIS-prune, and GIS at different iterations for query set generated from chr-22 on chr-18 dataset.

6.3 Comparison of distant strings

Our third experiment set inspects the performance of GIS and LIS when the query strings are much different from the database string. We use the query sets generated from chr-22 and search the chr-18 dataset in this experiment.

Figure 7 shows the average score and the accuracy of LIS, LIS-prune, and GIS on 1-NN queries for $|q|$ = 4000 query set for different number of iterations. We do not report the results for different length query sets separately here since they all had the same pattern. The maximum score in this experiment is slightly more than 90. This means that the length of the best matching substring is approximately 2.3% of the query length for this dataset. In

Mutation Ratio (%)	5	10	20	30	40	50
Pruning Ratio (%)	38.9	17.3	12.1	0.0	0.0	0.0

Table 4. The percentage of database pruned for various mutation rates.

Number of NN (k)	1	2	4	8
Pruning Ratio (%)	38.9	38.9	37.5	17.5

Table 5. The percentage of database pruned for various number of NN.

this experiment, both GIS and LIS achieved high scores extremely fast. Both techniques obtained 75% accuracy within the first 100 iterations. Since the total number of iterations to search the entire database is approximately 4300, both techniques can report 75% accurate result after processing only 2.5% of the database, and 90% accurate result after processing 12-27% of the database.

6.4 Evaluation of pruning rate

Unlike LIS-prune, both LIS and GIS search the entire database to ensure that there is no better result. LIS-prune reduces total search time by pruning (potentially) low scoring regions of the database. Table 4 shows the percentage of database pruned by LIS-prune during the experiments in Figure 6. As can be seen, LIS-prune prunes a huge percentage of the total search space for small mutation rates. For example, the total run time of LIS-prune is 64% less than both LIS and GIS when the mutation rate is 5%.

Table 5 lists the percentage of database pruned by LIS-prune during the experiments in Table 3. The amount of database pruned reduces slowly as the number of NN (k) increases. Even when k = 8, LIS-prune is 21% faster than both LIS and GIS.

6.5 Evaluation of confidence levels

Figure 8 shows how the confidence of LIS changes over iterations for 1-NN queries for the $|q|$ = 4000 dataset. LIS achieves 90% confidence after performing only 30% of the iterations. Another important point is that the confidence of LIS corresponds closely to its accuracy in Figure 6(a). That is, LIS can accurately report the user the quality of the partial results with respect to the final results before all the final alignments are found. We observed the same property in other experiments too. This experimentally substantiates the theoretical developments of Sections 3.1 and 3.2: the theoretical computation of accuracy (Figure 8) matches to the observed accuracy in experiments (Figure 6(a)). This is very important, because the user can estimate the score of the best matches of the final alignment at intermediate steps by inspecting the partial results and the confidence intervals.

6.6 Increasing the alphabet size

In this experiment we evaluate the quality and performance of LIS, LIS-prune, and GIS for larger alphabet size. We run queries on the protein dataset for this experiment. The alphabet size is 20 in this experiment.

Figure 9 plots the average score found during various iterations for the query set $|q|$ = 256 with 10% and 20% modification. We do not plot the results for other query lengths and mutation rates since they have similar behavior. The experiments show that both LIS and LIS-prune are better than GIS. LIS-prune is slightly better than LIS since it prunes some of the data blocks in advance. For 20% mutation rate, LIS and LIS-prune achieved 75% accuracy after only 31% of the data is processed. On the other hand GIS processes 66% of the dataset to achieve the same

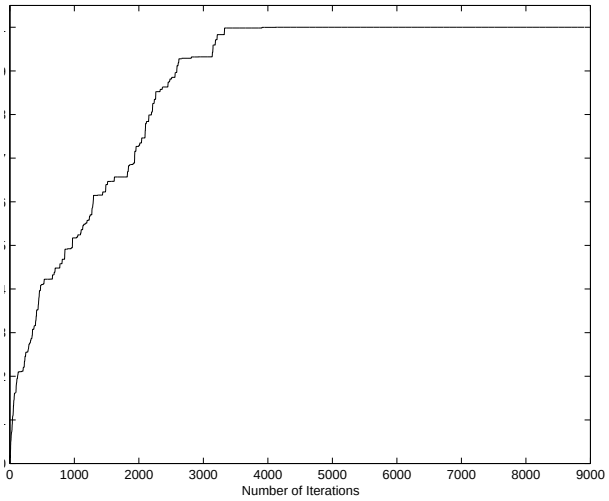


Figure 8. The confidence of LIS for 1-NN queries for the experiment in Figure 6(a).

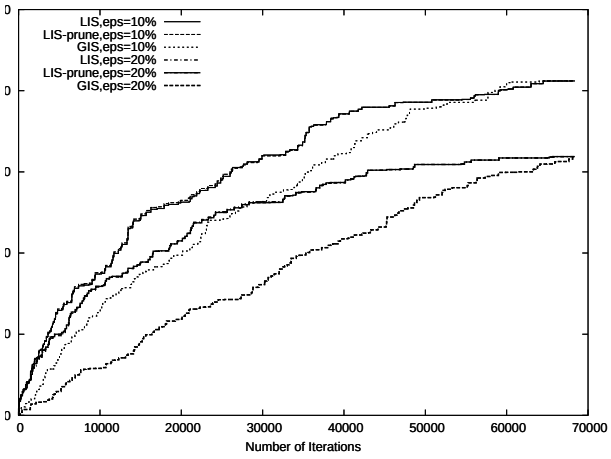


Figure 9. The score found by LIS, LIS-prune, and GIS on protein dataset for $|q| = 256$, and mutation rates 10% and 20%.

	Mutation Ratio (%)			
	0	5	10	20
$\|q\| = 256$	63.9	29.6	11.8	1.4
$\|q\| = 512$	70.1	73.6	69.6	50.1

Table 6. The percentage of database pruned for various query lengths and mutation rates.

quality. This means that BLAST's statistical model fails to predict the high scoring regions for larger alphabets while our model still works well.

Table 6 lists the percentage of data blocks pruned by LIS-prune for all the query sets. As the mutation rate increases, the amount of pruning drops. This is because the queries become more distant as the mutation rate increases. The alignments usually have lower score for distant queries. Therefore, the amount of error tolerance for pruning increases as the mutation rate increases. The amount of pruning is much better for $|q| = 512$ query sets. This can be explained as follows. The data space contains $O(|q|^{\sigma-1})$ possible locations for frequency vectors [26], where σ is the alphabet size. Therefore, as $|q|$ increases, the ratio of the volume of the MRS index structure to the volume of the data space drops dramatically. As a result of this, the probability that a given random query point happens to be far from an MBR increases. This property also implies that the LIS-prune technique works better for large alphabets.

7 Discussion

In this paper, we considered interactive substring searching for k-NN queries. We defined a new model, based on frequency vectors, for the analysis of the distance distributions between a query and a set of strings based on their frequency vectors. We presented formulas to compute this distribution in $O(\sigma)$ time, where σ is the alphabet size.

We proposed two novel interactive substring search techniques called *LIS* and *GIS*. LIS organizes the database as a set of MBRs with the help of the MRS index structure. The MBRs of the index structure are reordered based on the order statistics of their representative frequency vectors at various resolutions. These MBRs are then searched iteratively using any traditional string search tool. The partial results are reported at each iteration along with their confidence intervals. Later, we reduced the total run time of LIS by pruning the unpromising MBRs. This technique is called *LIS-prune*. GIS considers the average frequency of all the letters in each MBR. It employs BLAST's statistics model to predict the maximal score for each MBR, and ranks the MBRs based on this prediction.

The strategy used by LIS can be extended to other novel scientific databases in three steps: 1) The distance distribution between a pair of objects is determined using their signatures. 2) Database is organized into clusters using an appropriate index structure. 3) A representative for each cluster is defined based on the underlying distance function.

GIS can also be extended to arbitrary types of databases in three steps: 1) Split the data space into bins and map objects to bins. 2) Define a score for each bin pair (this scoring scheme must satisfy the requirements of BLAST's statistical model). 3) Find the frequency of each bin for each data block.

Our experimental results show that the proposed techniques achieve high accuracies quickly. In our experiments on DNA strings, (although there was no clear winner) GIS found high scoring results slightly faster than LIS. Both techniques achieved 75% accuracy within the first 2.5-35% of iterations. In our experiments on protein strings, LIS and LIS-prune were much better than GIS. LIS and LIS-prune achieved 75% accuracy within the first 31% of the iterations while it took 66% of the iterations for GIS. Our pruning strategy eliminated up to 38% of the database in these experiments.

Our techniques work better when the query string is similar to a subset of the database. These types of queries have many important applications, like determining the alignment of ESTs. Even when the queries are modified with high mutation rates, both of our techniques can find high scoring results after inspecting only 30% of the database.

The confidence value computed by LIS corresponds closely to its accuracy. This confirms the precision of the approximation formulas derived in the paper. Highly accurate confidence value provided by LIS enables user to predict the score of the final alignments even after the first few iterations.

In summary, the explosive growth of the scientific databases and the complexity of computing similarities makes

traditional non-interactive search tools undesirable since the user has to wait a long period of time to get the results. The methods presented in this paper provides the user instant response for string databases. These strategies can also be easily extended to other types of databases to make them interactive. Furthermore, the statistical models developed in this paper enable better query planning when the data is distributed.

We are currently building a web server which enables interactive queries. Our initial tests on the web server show that our techniques reduce the response time of queries on human chromosome database from minutes to a few seconds. In the future, we would like to test our algorithms on other types of databases. We also would like to develop a query execution plan for complex queries which involve more than one type of database.

References

[1] http://www.sdss.org.

[2] http://www.rcsb.org/pdb.

[3] http://ecocyc.org.

[4] http://www-unix.griphyn.org/projinfo/physics/astro.php.

[5] ftp://ftp.expasy.ch/databases/swiss-prot.

[6] ftp://ftp.ncbi.nih.gov.

[7] S. Altschul and W. Gish. Basic local alignment search tool. *J. Molecular Biology*, 1990.

[8] S. Altschul and W. Gish. Local alignment statistics. *Methods in Enzymology*, pages 460–480, 1996.

[9] R. Arratia and M. Waterman. A phase transition for the score in matching random sequences allowing deletions. *Ann. Appl. Prob.*, pages 200–225, 1994.

[10] D. Benson, I. Karsch-Mizrachi, D. Lipman, J. Ostell, B. Rapp, and D. Wheeler. GenBank. *Nucleic Acids Research*, January 2000.

[11] B. Berchtold, C. Böhm, and H. Kriegel. The pyramid technique: Towards breaking the curse of dimensionality. In *SIGMOD*, pages 142–153, 1998.

[12] B. Berchtold, D. Keim, and H. Kriegel. The X-tree: An index structure for high dimensional data. In *VLDB*, 1996.

[13] S. Burkhardt, A. Crauser, P. Ferragina, H.-P. Lenhof, E. Rivalsd, and M. Vingron. q-gram based database searching using a suffix array (QUASAR),. In *RECOMB*, Lyon, April 1999.

[14] E. Castillo. *Extreme Value Theory in Engineering (Statistical Modeling and Decision Science Series)*. Academic Press, New York, 1988.

[15] K. Chakrabarti and S. Mehrotra. The hybrid tree: An index structure for high dimensional feature spaces. In *ICDE*, pages 440–447, February 1999.

[16] S. Davidson, G. Overton, V. Tannen, and L. Wong. BioKleisli: A digital library for biomedical researchers. *Int. J. on Digital Libraries*, 1(1):36–53, April 1997.

[17] A. Delcher, S. Kasif, R. Fleischmann, J. Peterson, O. White, and S. Salzberg. Alignment of whole genomes. *Nucleic Acids Research*, 27(11):2369–2376, 1999.

[18] 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 *RECOMB*, Japan, 2000.

[19] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.

[20] R. Hamming. *Numerical Methods for Scientists and Engineers*. Dover Pubns, 2 edition, April 1987.

[21] J. Hellerstein, J. Naughton, and A. Pfeffer. Generalized search trees for database systems. In *VLDB*, pages 562–573, Zürich, September 1995.

[22] X. Huang and W. Miller. A time-efficient, linear-space local similarity algorithm. *Adv. Appl. Math*, 12:337–357, 1991.

[23] E. Hunt, M. P. Atkinson, and R. W. Irving. A database index to large biological sequences. In *VLDB*, pages 139–148, Roma, Italy, September 2001.

[24] H. V. Jagadish, N. Koudas, and D. Srivastava. On effective multi-dimensional indexing for strings. In *SIGMOD*, 2000.

[25] T. Kahveci and A. Singh. An efficient index structure for string databases. In *VLDB*, pages 351–360, Roma, Italy, September 2001.

[26] T. Kahveci and A. Singh. MAP: Searching large genome databases. In *PSB*, pages 303–314, January 2003.

[27] S. Karlin and S. Altschul. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proc. Natl. Acad. Sci.*, 87:2264–2268, March 1990.

[28] S. Karlin and S. Altschul. Applications and statistics for multiple high-scoring segments in molecular sequences. *Proc. Natl. Acad. Sci.*, 90:5873–5877, June 1993.

[29] N. Katayama and S. Satoh. The SR-tree: An index structure for high-dimensional nearest neighbor queries. In *SIGMOD*, pages 369–380, 1997.

[30] N. Katayama and S. Satoh. Distinctiveness-sensitive nearest-neighbor search for efficient similarity retrieval of multimedia information. In *ICDE*, pages 493–502, Heidelberg, Germany, April 2001.

[31] S. Kurtz and C. Schleiermacher. REPuter - fast computation of maximal repeats in complete genomes. *Bioinformatics*, 15(5), 1999.

[32] U. Manber and E. Myers. Suffix arrays: A new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993.

[33] S. Muthukrishnan and S. C. Sahinalp. Approximate nearest neighbors and sequence comparison with block operations. In *STOC*, Portland, Or, 2000.

[34] E. Myers. An O(ND) difference algorithm and its variations. *Algorithmica*, pages 251–266, 1986.

[35] E. Myers. A sublinear algorithm for approximate keyword matching. *Algorithmica*, pages 345–374, 1994.

[36] S. Needleman and C. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Molecular Biology*, 48:443–53, 1970.

[37] J. Ogasawara and S. Morishita. Practical software for aligning ESTs to human genome. In *CPM*, pages 1–16, 2002.

[38] W. Pearson and D. Lipman. Improved tools for biological sequence comparison. In *Proc. Natl. Acad. Sci*, volume 85, pages 2444–2448, 1988.

[39] P. Pevzner and M. Waterman. A fast filtration algorithm for the substring matching problem. In *CPM*, pages 197–214, Padova, Italy, 1993.

[40] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R+-tree: A dynamic index for multi-dimensional objects. In *VLDB*, pages 507–518, Brighton, England, 1987.

[41] T. Smith and M. Waterman. Identification of common molecular subsequences. *J. of Molecular Biology*, March 1981.

[42] D. States and P. Agarwal. Compact encoding strategies for DNA sequence similarity search. In *ISMB*, 1996.

[43] T. Tatusova and T. Madden. BLAST 2 sequences, a new tool for comparing protein and nucleotide sequences. *FEMS Microbiol. Lett.*, pages 247–250, 1999.

[44] D. White and R. Jain. Similarity indexing with the SS-tree. In *ICDE*, pages 516–523, 1996.

[45] H. Williams and J. Zobel. Indexing and retrieval for genomic databases. *TKDE*, 14(1):63–78, January/February 2002.

[46] Z. Zhang, S. Schwartz, L. Wagner, and W. Miller. A greedy algorithm for aligning DNA sequences. *J. of Computational Biology*, 7(1-2):203–214, 2000.