

CoMRI: A Compressed Multi-Resolution Index Structure for Sequence Similarity Queries

Hong Sun, Ozgur Ozturk, Hakan Ferhatosmanoğlu
Ohio State University
Department of Computer and Information Science
Columbus, OH 43210
Phone: (614) 292-6377
sun.82@osu.edu
{ozturk,hakan}@cis.ohio-state.edu

ABSTRACT

We present CoMRI, *Compressed Multi-Resolution Index*, our system for fast sequence similarity search in DNA sequence databases. Since the size of such sequence databases is usually large and varying lengths of queries are possible, it's impractical to index each single subsequence by any already known algorithm. In this paper, we employ Virtual Bounding Rectangles (VBRs) concept to build a compressed, grid style index structure. An advantage of grid format over trees here is sub-sequence location information is given by the order of corresponding VBR in the VBR list. VBRs are index cells of the proposed structure, each containing and approximating MBRs (Minimum Bounding Rectangles) and they can be represented rather compactly. Taking advantage of VBRs, our index structure fits into a reasonable size of memory easily. Together with a new optimized multi-resolution search algorithm, the query speed is improved significantly. Extensive performance evaluations on Human Chromosome sequence data show that VBRs save 80%-93% index storage size compared to MBRs and new searching algorithm prunes almost all unnecessary VBRs which guarantees efficient disk I/O and CPU cost. According to the results of our experiments, the performance of CoMRI is at least 100 times faster than MRS which is a grid index structure introduced very recently.

Keywords

Sequence similarity, Range Search, I/O Optimization, Quantization, Indexing for Similarity, DNA Similarity Query

1. INTRODUCTION

Biology has increasingly become a data driven science. The emergence of high throughput data acquisition technologies for investigating biological phenomena has been an important factor in this process [9]. Managing string databases

has been particularly important especially for the past few years with the explosion of sequence data, such as genome and protein databases. Such databases are available online and increasing rapidly. They are used by many researchers to generate new knowledge. Queries asking subsequence similarity are widely used to capture interesting and useful information from these large databases.

There have been several approaches for similarity search on sequences. An excellent survey of sequential algorithms that do not use indexing mechanisms can be found in [10]. There have been previous studies on data management and indexing the genome using various strategies [4, 7, 13]. For example BLAST [1], the most widely used tool in genomic research, first performs a linear scan of the whole collection of sequences searching for some exact hits to certain constant length subsequences of the query. The search is done using distinctive subsequences of the query. Candidate hits are then extended in two directions until the similarity between two sequences falls below some threshold. With the exponential growth in the database size, complete scan of the whole database increasingly causes scalability problems. In the near future techniques performing large scans on the database will be too slow to use, and will be impractical especially if multiple queries are required for an analysis of data. Naturally, fast and interactive retrieval is highly desired and difficult to achieve. In order to provide a satisfying searching speed, an efficient in-memory index structure has to stand behind every high quality search engine.

One of the widely used similarity query type is ϵ -range query. In an ϵ -range query over a string database, the user poses a query sequence and a similarity threshold, ϵ , then asks all subsequences within ϵ -range of the query sequence. To be able to state and run DNA sequence similarity queries, we need a similarity metric between string s_1 and s_2 , to decide if they are similar enough or not. For an ϵ -range query, the metric distance value between substrings is compared with the range threshold, ϵ , given by the user as the query parameter. One of the popular metrics is Edit Distance between two strings, which is in fact a measure of difference (i.e. higher the edit distance, more dissimilar the strings.) However, edit distance is an $O(n^2)$ algorithm both in terms of time and space requirements, where n is the length of the strings compared. This is highly inefficient, especially

for genomic sequence queries since relatively long sequences are of particular interest to researchers. Without an index the total query time (i.e. total time for comparing query string with all entries) is linearly proportional to database size. This total query time can be made constant by making the preprocessing in time linear to database size instead.

Effective transformations in the preprocessing time can be employed to reduce the time required to make each comparisons in the runtime of queries. We will follow the methodology of clustering/indexing the subsequences after transforming them into a multi-dimensional vector space and defining a distance metric for the new space. The queries in the transformed space are accurate approximations of the actual queries, and in the case of large datasets, I/O operations may be needed to further improve the query results. With the help of the distance functions, our framework achieves accurate approximations and effective pruning during the search. We have been studying different multi-dimensional data indexing methodologies like SS-tree [14], A-tree [12] VA-file and IQ-tree [3], and developed a new indexing method that would suit better for transformed genome data.

In the transformations we used, each string transforms into a d dimensional representative of feature vector regardless of the string size, which means comparison in constant time whereas in string domain this similarity measure takes time proportional to n^2 . The transformations we used are lossy transformations, so the distance in this new space cannot be expected to keep the exact relationship between the originating strings. There are two ways we envision of using the distances in the transformed domains, first as a lower bound to edit distance, second as an approximation to the edit distance between strings. The approach others followed is merely the first one [7] [5]. They try to find a distance function in the vector domain that will be a lower bound to the edit distance between the strings represented by those vectors. The search-space pruning algorithm using such functions have the good property that, they never misprune, i.e. never leave out a point that is actually in range, out of candidate set. However as you can see in Figure 1, these functions are far from being tight lower bounds. Since their average is very low compared to edit-distance, when the query range, ϵ , is high, they are mostly in the range. As a result of this their pruning rate abruptly decreases as ϵ increases. Some of the current distances are even proved to be the tightest bound that will guarantee to be lower-bound. To achieve the accuracy of this pruning method, we should strive to find different transformation domains, where better, tighter lower bounds is possible, but in the mean time we believe, we can use some approximation methods, where we gain big performance improvements together with an ignorable loss in accuracy.

Since DNA sequence is string data, feature based index structure is emerging as an important search paradigm in this kind of databases. The technique used is to map subsequences to points in a multi-dimensional feature space and index them via multi-dimensional data structures such as R-tree family [2,6,14]. DNA sequence databases and the transformations we use, have special properties and requirements, such as large size of the datasets, high dimensional feature

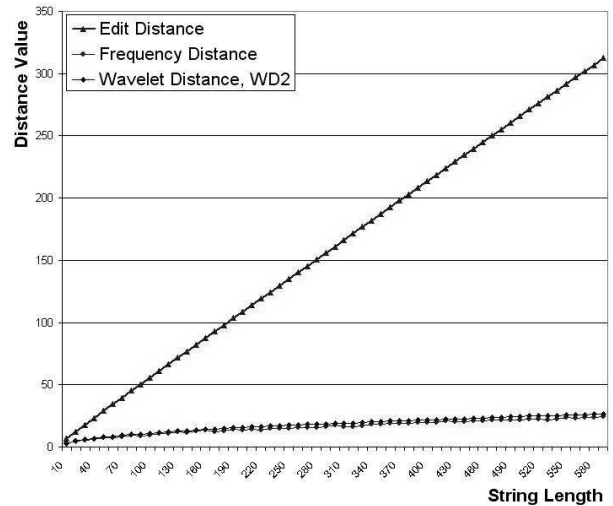


Figure 1: Average values of 2 distance functions to compare with edit-distance for samples of different length genome sequences

vectors and the need to support different query lengths. So those well-known tree structures don't fit the problem at hand well. For instance, R-tree is not good for high dimensional datasets and SS-tree, circular analogue of R-trees, uses too much space to fit into memory. In this paper, we introduce a non-tree/grid-style compressed index structure that offers high searching performance for DNA sequence similarity query problem. We have addressed the problem of transforming the sequence into vector space in previous studies and found that wavelet transformation on nucleotide 2-grams in the sequence was a good compromise to preserve enough features using reasonable space [11]. Here we propose a quantized, multi-resolution index structure on top of 2-gram based wavelet transformation of subsequences.

The organization of the paper is as follows: Section 2 lists the requirements of the targeted index structure, describes the ideas that our index structure is based on and defines the basic components building CoMRI. Then in Section 3 the search algorithm traversing this index structure to get the query results is described. After showing experiment results in Section 4, we end the paper in Section 5 with conclusion and future work.

2. DESIGN OF A COMPRESSED INDEX STRUCTURE

2.1 overview

Optimizing the number of I/O operations should be among the first things to consider for a tool for querying large datasets of genomes. However, it was not given enough priority even by highly popular programs like Blast [1]. Only recently Kahveci and Singh [7] designed their system with this aim and proposed a wavelet-based Multi Resolution String (MRS) index structure. They map the sub-sequences into an 8-dimensional space and then index these spatial points with a grid. Each grid-cell contains vectors of all substrings of a string for each resolution. The content of grid-cells are organized into MBRs each having same capac-

ity of vectors.

After analyzing mentioned specific properties of genetic sequence data, the transformations applied to substrings and the interesting queries for the researchers, we come up with the requirements for efficiently and effectively indexing DNA sequence data, which are still not solved by existing systems. The proposed approach in this paper have managed to address most of these requirements, listed as follows:

- In order to load the whole index into memory, the index has to be effectively compressed without losing much accuracy. Quantization methods should be applied when representing data nodes to save space. In section 4.1 we show how much our method succeeds to compress the index size.
- The capacity of the data nodes (grids, MBRs, etc.) used in the data structure has to be decided carefully. Because here we face the typical tradeoff between space and time efficiency. If we choose a small capacity for nodes, then the population of the nodes increases to the level where we will have difficulty to maintain the index in memory for the whole genome. If we choose a very large capacity for nodes this will inflate the number of I/O operations and computation, because we prune *nodes* from the candidate set and then for each candidate node, we retrieve all corresponding strings and check if it is in the ε -range. In section 4.1 and 4.2 we display the results of the experiments with different node capacities about number of I/O operations and pruning efficiency.
- The size of the researchers' queries has a large possible set. We need to supply to the demand of different query sizes except outliers, without restricting the user, or ignoring some part of their query. Having a multi-resolution approach gives the user this flexibility. In this approach we have subindexes, called resolutions, for certain lengths of strings. Then a query is partitioned into subqueries corresponding to these resolutions.
- The heuristic distance functions that are used to prune the search space, i.e. that are used to eliminate some of the candidates, are successful only in the query cases with small range values. However, frequently the researchers are interested in distant relative molecules, that accumulated many mutations in them. In those cases the range of the query submitted is relatively high. The search algorithms that have acceptable performance with small ε values might not perform well in those cases. A search algorithm should work for queries with both short and long ranges.

CoMRI is designed to satisfy the crucial technical requirements listed above. Our approach first transfers each subsequence of DNA sequence dataset into a wavelet vector and approximately represent a set of vectors, corresponding to adjacent overlapping subsequences, in a data structure called VBR (Virtual Bounding Rectangle). The set of VBRs in one resolution representing the substrings of a fragment is called VBR-stream. Finally, a list of VBR streams make

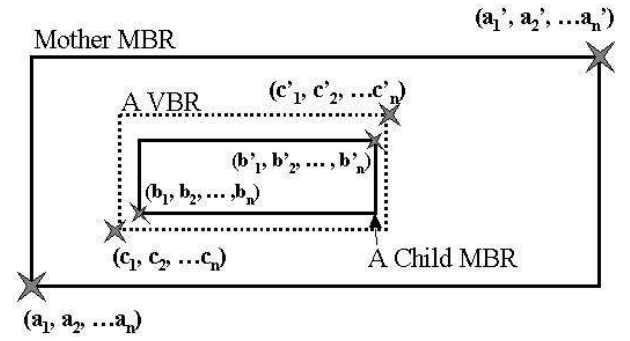


Figure 3: Representation of Mother MBR, MBR, and VBR

up the index structure. The rest of this section describes the definitions of VBRs, two distance functions and details about structure of the CoMRI.

2.2 VBR: a basic block of proposed structure

In many indexing methods [7, 8, 14] Minimum Bounding Boxes are used to represent a group of entries in the database. This saves space since instead of representing each and every data point in the index, similar points are grouped together in a box, and used as the entry of the database. When it is time to run the similarity query, MBRs that can not possibly contain any point entry in the range of search are eliminated and point to point comparisons are only done between the query point and the points in the remaining or *candidate* MBRs.

MBRs are denoted by their minimum and maximum coordinates. Consequently MBR size in memory is sum of the sizes of two vectors. Given that a vector has n dimensions and an integer data type is represented using four bytes, for the MBR we spend $2 \cdot 4 \cdot n$ bytes in the heap. A more compact representation can improve the performance substantially. VBR [12] is a compact representation based on quantization. The vector points are concentrated more in a certain region in the space. Representing the boxes in reference to a box containing them all, Mother MBR, is much more compact than representing them in reference to whole space.

Definition 1: Mother MBR is the smallest rectangle that covers all spatial points in n -dimension space. Mother MBR is defined by two edge-points a and a' , where $a = [a_1, a_2, \dots, a_n]$, $a' = [a_1', a_2', \dots, a_n']$ and $\forall i a_i \leq a_i'$. All MBRs in the space are the children of Mother MBR.

Definition 2: A VBR (Virtual Bounding rectangle) is a quantized differential expression of a child MBR. Let $A = (a, a')$ be the mother MBR, let $B = (b, b')$ $b = [b_1, b_2, \dots, b_n]$, $b' = [b_1', b_2', \dots, b_n']$ be a child MBR contained in A . Hence, $a_i b_i b_i' a_i'$ where $i = 1, 2, \dots, n$ holds. A new rectangle $C = (c, c')$ $c = [c_1, c_2, \dots, c_n]$, $c' = [c_1', c_2', \dots, c_n']$ exists, c_i and c_i' where $i = 1, 2, \dots, n$ are the quantized values b_i and b_i' relative to the interval (a_i, a_i') . We call C is a VBR if the following property holds: $a_i \leq c_i \leq b_i \leq b_i' \leq c_i' \leq a_i'$ Figure 3 shows an example VBR. We define the quantization functions specifically as following. Let $q \geq 1$ be an

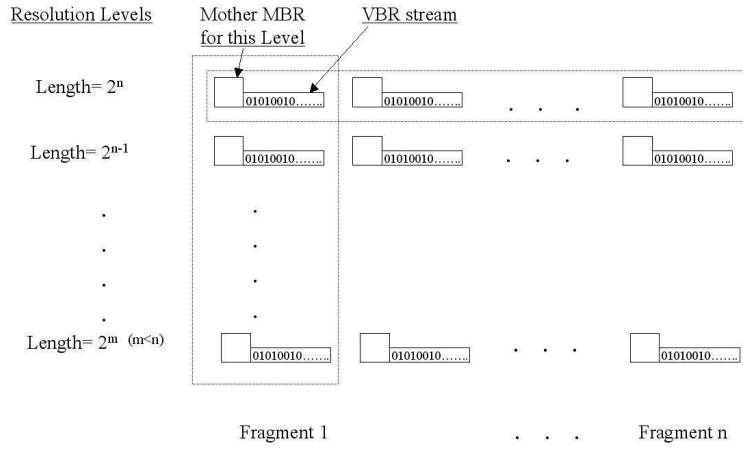


Figure 2: The overview of the index structure

integer, such as: $c_i = a_i + \frac{(a_i' - a_i)h_s(b_i)}{q}$ where

$$h_s(b_i) = \begin{cases} q - 1 & \text{if } b_i = a_i'; \\ \lfloor \frac{b_i - a_i}{a_i' - a_i} \cdot q \rfloor & \text{otherwise.} \end{cases}$$

Similarly $c_{i'} = a_i + \frac{(a_i' - a_i)h_e(b_i)}{q}$ where

$$h_e(b_i) = \begin{cases} 1 & \text{if } b_i = a_i'; \\ \lfloor \frac{b_i - a_i}{a_i' - a_i} \rfloor & \text{otherwise.} \end{cases}$$

Definition 3: A Resolution Level of VBR-index is actually an index in itself that is only capable of querying strings of certain length. We need to have multi-levels of resolution to be able to handle different query lengths, but not a resolution for every possible query length as this would cause a really large index size. In Section 3, we describe how we support a certain range of queries by partitioning it into subqueries of lengths matching to resolution levels.

2.3 Wavelet Transformation WT#N(s) and Wavelet Distance WD#N(v_1, v_2)

Here the transformation and distance definitions are given in a more general way for n-tuples. However, in CoMRI we used WD2, the distance we had defined for vectors in WT2 domain (wavelet transformation for 2-tuples) [11]. Since, through our previous studies, we had found WT2, to preserve enough features of the sequence. The definition we give for WD2 uses Frequency Transformation for n-grams, so first we define them.

Definition (Frequency Transformation, FT#N(s)): For a string s of length n consisting of characters from an alphabet Σ , we keep track of frequencies of occurrences of each character, using a vector v of $|\Sigma|$ dimensions. Sum of those frequencies will eventually be n . This vector is the frequency transformation of this string s , FT1(s). [7] Here we give a more generalized definition, FT#N(s) which keeps track of frequency of N-grams (instead of 1-grams, i.e alphabet symbols).

Examples: For genome data, $\Sigma = (A, C, G, T)$, so FT1("TAT") = 1,0,0,2 and FT1("GGTG") = 0,0,3,1. Transformation

length is 4 regardless of the string length. For FT2 it is 16 since there are $4^2=16$ 2-tuples possible (i.e. AA, AC, AG, AT, CA, CC ... TT).

Definition Wavelet Transformation for N-grams(WT#N(s)): After dividing s into two equal substrings s_a and s_b , we compute the transformations $v_a = \text{FT}\#N(s_a)$ and $v_b = \text{FT}\#N(s_b)$. Concatenation of addition and subtraction of those two vectors, $[v_a + v_b, v_a - v_b]$, is our N-gram Frequency Wavelet transformation, WT#N(s). This transformation is presented in a more formal way with a recursive wavelet definition in [7]. Definition given here is the first two wavelet coefficients of last recursion in that formal definition and it is more accessible to people. This definition promotes efficiency because of elimination of recursion.

Definition Wavelet Distance for N-grams(WD#N(v_1, v_2)): This distance function is designed for Wavelet Vectors defined above. First we calculate "posDist" and "negDist" (positive distance and negative distance) between v_1 and v_2 . (posDist: sum of positive values in $v_1 - v_2$, negDist: absolute value of sum of negative values in $v_1 - v_2$.) Where δ is $|\text{posDist} - \text{negDist}|/2$ and m being minimum of posDist and negDist, Wavelet Distance for N-gram wavelet transformation vector is defined as:

$$WD\#N(v_1, v_2) = \begin{cases} \delta/n & \text{if } R_i < \delta; \\ (\delta + (m - \delta)/2)/n & \text{otherwise.} \end{cases}$$

Definition Scaled WD2 (SWD2(v_1, v_2)): We scaled the WD2 function to get its average same with Edit Distance via a statistically generated the polynomial function from experiments results.

2.4 Index Structure

Figure 2 shows the layout of CoMRI structure. In the structure, each column represents a fragment in the dataset and each row represents the index content of a resolution. Each resolution contains a mother MBR represented by its two corner points and a VBR stream, represented as two points quantized with reference to mother-MBR. As described be-

fore, there is a tradeoff between time and space efficiency related to the capacity of VBRs. Furthermore, occupying more space in memory for the higher levels also degrades performance, since we have to scan all the VBRs for the subquery with highest resolution. On the other hand, as we mentioned in Observation 1, shorter queries are already more difficult to prune, so we favor a small VBR capacity for them. Another advantage of smaller VBR capacity is that less number of comparisons between query and individual sub-sequences has to be made in the final disk reading stage per candidate VBR, as less vector points are included in a VBR. To optimize in both ways, we decided to use an adaptive strategy, which we named as dynamic VBR (DVBR). A cutoff value, c is set as a parameter. Resolutions higher than this cutoff have a constant VBR capacity whereas resolution levels lower than c , have a VBR capacity proportional to the resolution level; lower the resolution, smaller the VBR capacity. A more formal definition of DVBR is as follows:

Let $R_1, R_2 \dots R_n$ are resolutions in CoMRI structure, where Minimum Resolution $\leq R_i \leq$ Maximum Resolution, Given a cutoff c

$$VBRcapacities = \begin{cases} c & \text{if } R_i \geq c; \\ R_i & \text{otherwise.} \end{cases}$$

3. ALGORITHM

In this section we propose an efficient search algorithm that is optimized for the proposed structure. In some of the previous studies also based on partitioning the query into subqueries [7], the subqueries are searched at corresponding resolution based on range value updated in the previous resolution. Previous subquery only updates the range value and then *all* of the entries in the next resolution level are evaluated for new subquery. Our algorithm, in lower resolution, requires subqueries to be run *only* for neighbors of the candidates from the upper resolution level via the assistance of location information.

Pruning iterations of our algorithm start from the highest resolution going top down, whereas the MRS-index [7] starts pruning using the most granule resolution (i.e. resolution for the shortest subsequence length) going up. As we will describe in the next paragraphs top-down search improves the efficiency of the algorithm substantially. Given a query q with random length, we always can find a partition format as following: $q = [q_1] [q_2] \dots [q_i] \dots [q_n] [r]$ where $q_i = 2^{ci}$ (Minimum Resolution $\leq ci \leq$ Maximum Resolution) and r is the remaining sequence. Figure 4 presents the strategy we use to partition a query sequence. The shortest subquery is always located at start of string, then the second shortest sub-query and so on. If any remaining contents exist, it remains at the end. Each sub-query obtained from this partitioning technique matches one of resolutions of the index structure.

Observation 1: The average edit distance between two random strings is a little above half of their length, with a small variance. As the lengths of two pairs strings increase the probability of having their distance smaller than a constant ϵ value decreases.

According to Observation 1, longer strings have higher probability to get pruned than the shorter strings. q_n as the

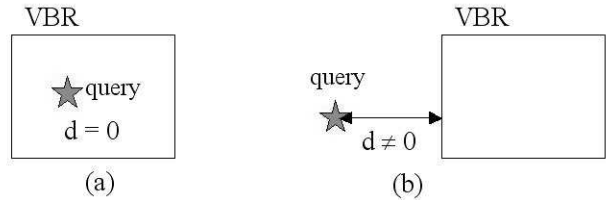


Figure 5: Two cases for the query with respect to VBR.

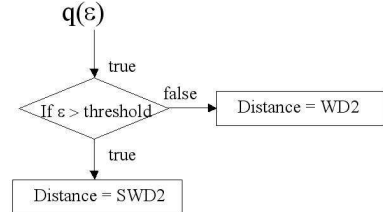


Figure 6: Flowchart for the decision of which distance function to use.

longest sub-query of q has the largest pruning rate among all resolutions. In our range query algorithm, we start the search from the highest resolution with the subquery q_n and then continue searching using q_{n-1}, q_{n-2} until q_1 each at its corresponding resolution level, comparing only with the VBRs that are the neighbors of candidates of the upper level. Using q_n in the first search iteration improves pruning efficiency. Starting with higher resolution causes early pruning of most candidates, so operations on lower resolutions become much cheaper. This early pruning result will greatly reduce the search time in rest of the resolutions.

The relationship of a VBR and a sub-query on the same resolution level is shown in Figure 5. Figure 5 (a) shows the situation that a sub-query point falls into the VBR. In this situation, the distance between them is zero without doubt. Figure 5 (b) shows the situation that a sub-query point is located outside of a VBR. In this case, we have two methods to calculate the distance, WD2 is successful only in small range queries but comes with a guarantee not to misprune, SWD2 is the scaled up version using a polynomial fitting so it prunes better even for large ranged queries, in the cost of losing lower-bound property. We use the scaled function, as described in the flowchart in Figure 6, where WD2 starts to fail pruning well.

If the distance derived from above is within the allowed range, we will keep this VBR as a candidate from this resolution level. The candidate structure is as below:

```

Structure Candidate{
int Offset; /*this VBR position in the VBR stream in
this resolution */
int Range; /* the distance left for the rest of the resolu-
tions */
Candidate Structure

```

In the first search iteration, whole VBR stream has to be scanned in order to obtain the first list of candidates. How-

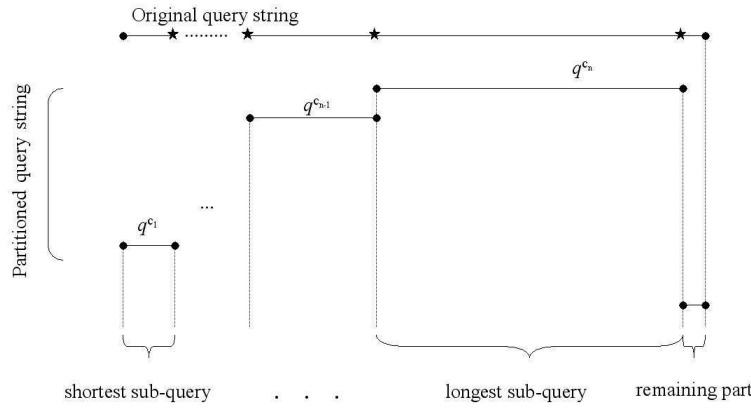


Figure 4: Partitioning the query into suitable lengths to be submitted as a query to different resolution levels to be compared with candidates

ever, once candidates are generated from the first resolution level, no scanning is necessary for the remaining resolution levels. Since each candidate contains information of VBR position and available range from upper resolution, we simply continue eliminating candidates among possible VBRs that are located as preceding neighbors of the candidates from the higher resolution level. We also keep track of a separate query range value for each candidate and they are calculated from the epsilon range and eventual distance values of upper neighbor candidate. This decreases the matching probability more and increases pruning rate.

We continue this process until either no candidates left or the shortest sub-query is reached. For the later situation, disk pages corresponding to the last set of candidates are read and the distance between the full length of query q and sub-sequences from datasets are calculated.

Since candidates are always in ordered according to the scanning in the first search iteration, the final disk I/O should be more like sequential scan rather than random access. Consequently, the cost of disk I/O is lower.

```

Structure MBR{
int[] Lowerbound; /*lower bound of this MBR*/
int[] Upperbound; /*upper bound of this MBR*/
};

Structure VBR-stream{
MBR MotherMBR; /* The mother MBR structure*/
int[] Radix; /* The bits length in each dimension*/
byte[] VBRStream; /* Hold all VBRs in a bit stream*/
int Resolution; /* the resolution this VBRStream is
represented*/
};

```

The building blocks of the VBR-index.

Algorithm RANGE-SEARCH (q, r)

- Load index from disk file to memory.
- Split the incoming query q into n sub-queries as q_1, q_2, \dots, q_n that $|q_i| = 2^{c_i}$ and $\text{MinimumResolution} \leq c_1 \leq c_2 \dots c_n \leq \text{MaximumResolution}$.
- Calculate wavelet vectors of subsequences, q_i 's, and whole query sequence, q .
- For $j \leftarrow 0$ to m
 - /*go through each fragment i.e. gene sequence*/
 - For sub-query with the highest resolution:
 - For $i \leftarrow 0$ to number of VBRs on this resolution from the index
 - /* Only the highest resolution in the query need to go through whole VBRStream.*/
 - if $\text{distance}(i) \leq r$
 - /*Let VBRCandidate be a resulting structure and Candidate be a vector holding all candidates.*/
 - $\text{distance in VBRCandidate} \leftarrow r - \text{distance}$
 - /*the distance left to the rest of possible resolution*/
 - $\text{offset of VBRCandidate} \leftarrow i$
 - $\text{Candidates.add}(\text{VBRCandidate})$
 - if Candidates not empty
 - for each VBRCandidate
 - /* If necessary, process the rest of resolution*/
 - for $j = c_n - 1$ to c_1
 - Use offset from the VBRCandidate to find the related VBR(s) in this resolution
 - if $\text{distance} \leq \text{distance from VBRCandidate}$
 - update VBRCandidate with new distance and offset
- if Candidates not empty
 - Read disk pages corresponding to offset and fragment position of each VBRCandidate.
 - Calculate the actual distance between q and sub-sequence from datasets.

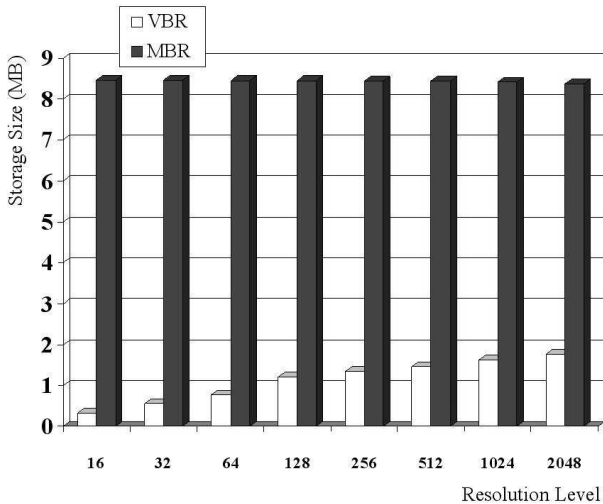


Figure 7: Size gain via using VBR

The Algorithm used to query the VBR-index.

4. EXPERIMENTAL RESULTS

In our experiments, we selected human chromosome 18, chromosome 22 as target datasets [7]. Chromosome sequences are composed of the alphabet set A, C, G, T. We implemented wavelet transforms for resolution range from 2^4 to 2^{11} . Since the alphabet size is 4 and since we are using first coefficient of wavelet transformation on 2-tuples, each transformed feature vector has 32 dimensions. As described in the above sections, we combined *distance function on wavelet transformation on 2-tuples* (WD2) and *scaled WD2* (SWD2) in pruning process to take advantages of both where they are better.

Similarity search is a data-intensive task, and disk page access is dominant factor especially for large-scale datasets. Our measures are based on the number of disk pages that has to be visited from the disk file in the final stage. The disk page size is set to 1K in our experiments in order to compare with previous studies. The rest of this section gives the results of our experiments.

4.1 Effect of VBR on size of index structure

The first experiment reports a significant shrinkage on the size of index structure by using VBR instead of using MBR as displayed in Figure 7. As we mentioned in above section, VBR quantizes the lower bound and the upper bound position with reference to Mother MBR of the resolution. Figure shows the average storage saving rate of VBR over MBR on each resolution in chromosome 18. The lowest resolution 2^4 saves up to 93% and even the highest resolution 2^{11} saves about 80%. The reduced storage space by using VBR implies we can employ a much smaller size of VBR in our index structure, which increases the pruning performance observed in 4.4.

4.2 Effect of VBR capacity

Our next experiment considers the impact of the VBR size on the disk I/O costs. As mentioned in the previous section, the pruning efficiency of an index structure formed of

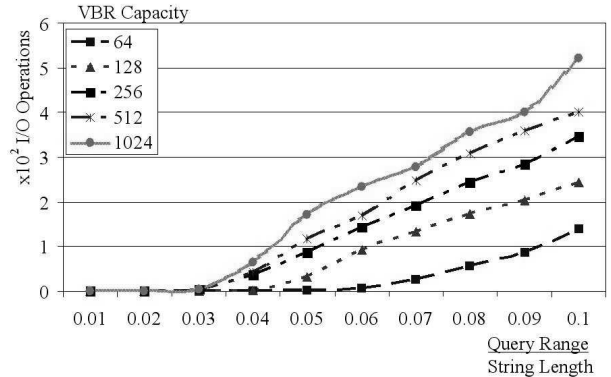


Figure 8: Costs of various DVBR capacities

data cells depends on the total number of cells. Decreasing number of cells to save space implies each cell has to contain more feature vectors, thus the performance of the index structure deteriorates as we will fetch more data for each candidate cell and the final disk I/O increased. In this experiment, we used queries with length 500, 1000, 2000 for VBR capacities 64, 128, 256, 512, 1024 for range query on chr18 dataset. The queries are randomly picked from the chr22 dataset. Figure 8 plots the effects of these VBR capacities on I/O costs of queries in our index structure. From the results, we can see smaller VBR size increases pruning rate very significantly. Such as with the settings VBR capacity 64 and range rate (Query Range/String Length) 0.05 the cost is about 7 times less than the cost of VBR size 1024.

4.3 Effect of using SWD2

The third experiment exposes the advantage of using SWD2 on high values of query range. The queries are generated randomly and the target dataset is chr18. As depicted in Figure 9, SWD2 has much higher pruning rate than WD2 on range queries with high epsilon values. The performance of SWD2 is 10 times better than WD2 on the range rate of 0.1. For low range rate values the pruning performance of SWD2 and WD2 is almost same in the low end. This is because the distance effect on the pruning is reduced in the highly similar searching. Since WD2 is successful in pruning for range queries with small ϵ values, we don't need to use the scaled distance function taking the risk of mispruning, so in the final product, we put a conditional checking if the range rate value is larger than a threshold, and using the scaled function for such cases of long subqueries.

4.4 Compare Dynamic VBR and Static VBR

Chart in Figure 10 displays the number of I/O operations required by two different design choice about VBR capacities. DVBR is the Dynamic VBR capacity choice where each resolution below a certain resolution has a decreasing VBR capacity as we go to a lower resolution. This gives us more pruning, since candidate VBRs will have less entries, but increases the index size, since we will have more number of VBRs. SVBR is the static case, where VBR capacity is same independent of each resolution level.

4.5 Comparison with MRS

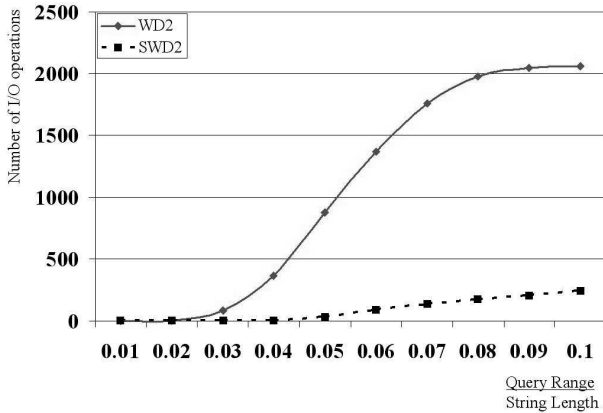


Figure 9: Comparison of the I/O costs of WD2 and SWD2

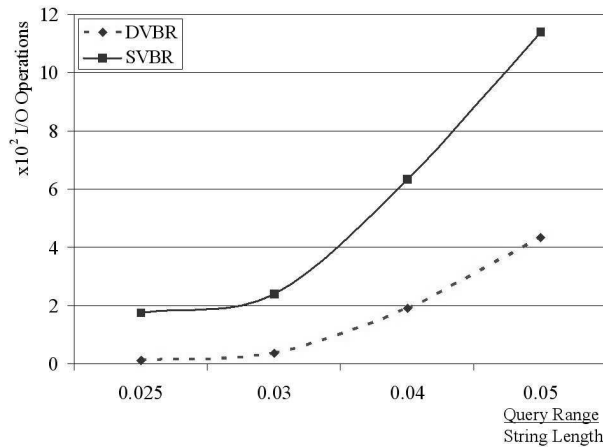


Figure 10: comparison of number of I/O operations of two different design choices

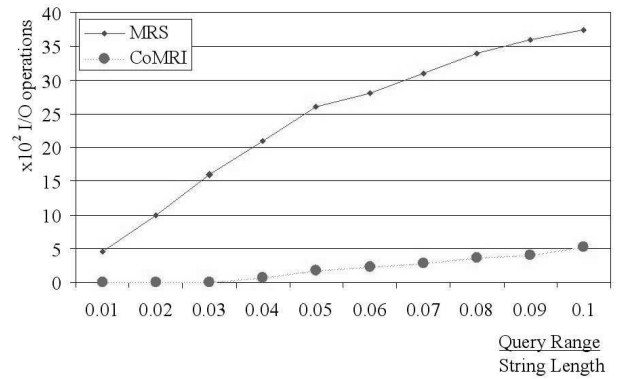


Figure 11: I/O cost comparisons for CoMRI and MRS [7]

In this last experiment, we compare the costs of range queries between CoMRI and MRI. The experimental queries are randomly picked from chr22 dataset again and the lengths are range from 500 to 2000. 11 shows that CoMRI performs 100 times faster than MRI even around the largest range rate. The advantages CoMRI gained over MRI are coming from 1) Early pruning with searching longest resolution first. 2) Selected searching instead of full searching on most of resolutions. 3) Usage of combination of WD2 and SWD2. 4) Applying concept of DVBR. 5) Shrinking index size via usage of VBRs.

5. CONCLUSIONS

Considering the problem of similarity searching on a large DNA sequence dataset, there are two important aspects we have to focus on. The first one is the index size. An optimal index structure has to provide enough information on searching stage, meanwhile its size has to be as small as possible to fit into memory. The second one is searching efficiency. Searching efficiency is determined by pruning rate in memory on the index structure. Intuitively, higher pruning rate in memory means less disk I/O and less time consuming. In this paper, we proposed a new Compressed Multi-Resolution Index structure, CoMRI, which addressed the above two aspects with a successful solution.

Firstly, we transfer all sliding sub-sequences to their wavelet form and then group them into a list of ordered VBRs. We represent CoMRI structure as a multi-resolution grid. Consequently, various resolution levels as rows in this grid. In each row, we have adjacent VBRs as VBR stream. Since VBR is the basic component of CoMRI, the compact character of VBR directly benefits the storage size of CoMRI. We show the comparison of CoMRI with other index structure based on non-compressed format in experimental section. CoMRI only takes 15% storage space on average. Thanks to this compact solution, we could easily implement an in-memory searching algorithm.

Secondly, we present searching algorithm for similarity queries. The query is partitioned into different resolutions before searching. The algorithm always starts from the highest resolution and ends at the lowest resolution composing the query. This approach guarantees most of the pruning occurs

early and less candidates are carried to the lower resolution levels. In pruning stage, we apply both WD2 and SWD2 functions according to the ε value of the current sub-query. WD2 is used for short query range preventing mispruning and SWD2 is applied on longer query range values for efficient pruning. Also, we utilize dynamic VBR to achieve even deeper pruning.

The curious reader can check our groups web-site for a demo of our system.

It will be available soon at the URL: <http://www.cis.ohio-state.edu/db/>

6. REFERENCES

- [1] S. Altschul, W. Gish, W. Miller, and J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.
- [2] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R* tree: An efficient and robust access method for points and rectangles. pages 322–331, May 23–25 1990.
- [3] S. Berchtold, C. Bohm, H. Jagadish, H. Kriegel, and J. Sander. Independent quantization: An index compression technique for high-dimensional data spaces. In *Proc. 16th Int. Conf. on Data Engineering*, San Diego, CA, 2000.
- [4] V. Gaede and O. Gunther. Multidimensional access methods. *ACM Computing Surveys*, 30:170–231, 1998.
- [5] 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.
- [6] A. Guttman. R-trees: A dynamic index structure for spatial searching. pages 47–57, 1984.
- [7] T. Kahveci and A. Singh. An efficient index structure for string databases. pages 351–360, Roma, Italy, September 2001.
- [8] N. Katayama and S. Satoh. The sr-tree: an index structure for high-dimensional nearest neighbor queries. In *Proceedings of the ACM SIGMOD international conference on Management of data*, pages 369–380, Tucson, Arizona, May 1997.
- [9] Naren Ramakrishnan Lenwood S. Heath.
- [10] G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.
- [11] Hakan Ferhatosmanoglu Ozgur Ozturk. Effective indexing and filtering for similarity search in large biosequence databases. In *International Symposium on Bioinformatics and Bioengineering BIBE 2003*, Washington, DC, march 2003. IEEE.
- [12] Y. Sakurai, M. Yoshikawa, S. Uemura, and H. Kojima. The a-tree: An index structure for high-dimensional spaces using relative approximation. In *Proceedings of 26th International Conference on Very Large Data Bases, September 10–14, 2000*, pages 516–526, Cairo, Egypt, 2000.
- [13] J. McCarthy F. Olken M. Zorn V. M. Markowitz, S. Lewis. Data management for genomic mapping applications: A case study. In *SSDBM*, pages 45–57, 1992.
- [14] D. White and R. Jain. Similarity indexing with the SS-tree. pages 516–523, 1996.