Using Transformation Techniques Towards Efficient Filtration of String Proximity Search of Biological Sequences

Alireza Aghili, Divyakant Agrawal, Amr El Abbadi
{aghili, agrawal, amr}@cs.ucsb.edu

January 14, 2003

Abstract

The problem of proximity search in biological databases is addressed. We study vector transformations and conduct the application of DFT(Discrete Fourier Transformation) and DWT(Discrete Wavelet Transformation, Haar) dimensionality reduction techniques for DNA sequence proximity search to reduce the search time of range queries. Our empirical results on a number of Prokaryote and Eukaryote DNA contig databases demonstrate up to 50-fold filtration ratio of the search space, up to 13 times faster filtration. The proposed transformation techniques may easily be integrated as a preprocessing phase on top of the current existing similarity search heuristics such as BLAST[1], PattenHunter[11], FastA[17], QUASAR[4] and to efficiently prune non-relevant sequences. We study the precision of applying dimensionality reduction techniques for faster and more efficient range query searches, and discuss the imposed trade-offs.

1 Introduction

Discovering the structure, function and evolutionary relationship of genes are the main goals of genome sequencing research, where comparative analysis of homologous sequences is a crucial part in the study of gene function, known as *genomics*. The behavior resemblance of two DNA sequences of two different organelles or species to the same external exposure, may be used to infer *functional or structural similarities*, or *mutual inclusion in the same pathway or biological mechanism*. Some of the vast applications of proximity search include *discovering the nature and functionality of human genome*, *phylogenetic analysis*, *drug discovery*, *keyword search* in databases, or user pattern analysis

in the context of *network security*, and many more. For instance, approximate sequence analysis has assisted the detection of certain strains of the *Escherichia coli(E.coli)* bacteria responsible for infant *diarrhea* and *gastroenteritis*. The researchers at the University of Chicago's Howard Hughes Medical Institute[16] discovered a protein molecule able to transmit a genetic trait without DNA or RNA in *yeast*, which is able to string itself together into a long fiber, much like those found in the brain in *mad cow* and human *Creutzfeldt-Jakob* diseases. In general, some of the typical applications of sequence proximity search include[3]:

- Identification of highly conserved residues/motifs which are likely to correspond to essential sites for the structure or function of the sequence.
- Phylogenetic analysis which relies on neighbor sequence search, at the protein or DNA level, to predict mutations from which it is possible to retrace evolutionary relationships among different genetic sequences. Similarly, phylogenetic trees provide the information to reconstruct the history of species and gene families.

The proximity search seeks the sequences close enough to a given query sequence either through direct alignment[15, 18] or using other heuristics[1, 11, 17, 19]. The alignment of biological sequences (pairwise or multiple alignment) is the operation to place nucleotide or amino acid residues in columns inferring the closest common ancestral relationships. This is achieved by introducing gaps (with predefined *cost*s), to represent insertions or deletions (so called *indels*), into sequences. Hence, an alignment is a hypothetical model of mutations on the residue level through *edit operations* (Replacement, Insertions, Deletions). The best alignment usually refers to the one demonstrating the most likely evolutionary scenario. Let $S_1, S_2 \in \Sigma^*$ to be finite ordered DNA sequences of characters(*bases*) taken from the alphabet set Σ, where $\Sigma = \{A, C, G, T\}$. Each pair of characters from Σ are assigned a replacement(substitution) *score/cost* and the substitution matrices[6, 8, 20] providing such information are built based on the structure similarity and replacement likelihood of the residues. For instance, at the DNA level, probabilities of substitution vary according to the *nature* of the base pairs. Notably, *transitions* (substitutions between two *purines*, A and G, or two *pyrimidines*, C and T) are generally more frequent than *transversions* (substitutions between a purine and a pyrimidine). Hence, the optimal alignment of sequences S_1 and S_2, named S_1' and S_2', is achieved by applying the minimum number of edit operations to transform S_1 into S_2, called *Edit Distance*, or $ED(S_1, S_2)$. The pairwise optimal alignment is based on dynamic programming

ensuring the maximal *score* (or the minimal *cost*) and requires $O(pq)$-time, and $O(pq)$-space, using dynamic programing[15, 18] algorithm. An example of the described procedure is depicted in the following example:

$$
\begin{array}{ll}
S_1 & \text{A A C T C G A G A C C C} \\
S_2 & \text{A T C C G A G A G G T C C C} \\
\Downarrow & \text{————————————————} \\
S'_1 & \text{A A C T C G A G A - - - C C C} \\
 & \quad \text{R} \quad \text{D} \qquad\qquad \text{I I I} \\
S'_2 & \text{A T C - C G A G A G G T C C C}
\end{array}
$$

where R, D, and I correspond to *Replacement*, *Deletion* and *Insertion* respectively. Computing the optimal alignment of n sequences, each of length l requires $o(2^n l^n)$-time and $o(l^n)$-space. Unfortunately, such an algorithm is neither practical nor scalable. Following is a summary of some of the problems encountered in the sequence proximity search within the context of biological sequences:

- The quadratic *computational complexity* of the optimal alignment is so high, making it *impractical*.
- Due to the limitations on the current knowledge of *mutations* and their corresponding probabilities, only approximate searches and heuristics[1, 11, 17, 19] have been practically applied for comparison of sequences.
- *Scalability* is one of the most important issues to be addressed. Neither the Dynamic programming algorithms nor the heuristics may practically be applied to a large number of sequences(across species), each of which might be composed of billions of residues.

The rest of the paper is organized as follows: Section 2, discusses the background and related work, followed by the motivation and terminology in section 3. Section 4, studies the proposed transformation techniques and their integration. Section 5 demonstrates a concise empirical performance analysis and the simulation results. Finally, section 6 concludes the work.

2 Background, Related Work

In a typical application of range query, given a protein or DNA query sequence Q and range r, it is compared with all the sequences in the database, in search for sequences which are at most

r edit operations far from query Q. However as mentioned before, because of the quadratic time involved, the dynamic programming[18, 15] algorithms may not be directly or practically applied for this purpose. Several heuristics[1, 11, 17, 4] have been proposed to speed up the homology search procedure, which are not efficient for range query over large datasets. These heuristics need to inspect the entire database while only a very small part of it might actually be of interest. The rest of this section highlights the recent research addressing the similar problem.

Multi-Resolution index Structure(MRS)[10] is a technique based on *Haar wavelet*, designed to speed-up range queries. It uses a sliding window of size $|w|$, moving over the query sequence and for each possible location, extracts the first and second *Haar wavelet* coefficients of the $|w|$-sized subsequences/windows. Hence each window is mapped to a point in $\Re^2$. Furthermore, every c trail of windows is represented with a single *Minimum Bounding Rectangle(MBR)*. The trail of MBRs are subsequently processed at different resolution levels, based on different values for $|w|$. Given a range query (Q, r), the query is first divided into the maximum 2^i-sized postfix segments, with the assumption that the query sequence is originally of length 2^j for some $j \geq i$, and furthermore, each segment is searched within the respective resolution level. However, i) the lower bound provided by their *Frequency Distance(FD)* function is not tight enough for *score* and ED estimations and using more coefficients, MRS does not guarantee a better precision, contrary to what is expected from a distance preserving transformation, ii) the focus of the work is on the performance of the index structure and does not analyze the filtration efficiency of using *wavelet* transformation on biological data.

Chavez and Navarro[5] translate the problem of approximate string search into a range query or proximity search in a metric space. The technique is based on picking k pivots randomly, and mapping each sequence with a k-dimensional vector(only keeping min and max distances), further using triangle inequality to prune non-relevant sequences using Suffix Tree[2] as index structure. No empirical analysis is conducted to evaluate this approach on real biological data. SST[7] uses overlapping sliding windows of size w over the database sequences and maps them into a $\Re^{4^w}$-dimensional frequency vectors. After that, SST uses k-means clustering algorithm, hierarchically clustering database sequences. Given a query Q, it is first divided into non-overlapping windows, pruning the database windows which are farther from the given query range, and finally studying the effect of window size on search time, and error rate of input data on true positive/negative

rates. Finally, authors in [21] provide a concise study of DFT/DWT transformations, but only in the context of time-series databases.

In this work, we study the effectiveness of different edit distances, and the application/integration of DFT/DWT for the purpose of sequence proximity search specially in the context of biological databases.

3 Motivation, Terminology

Definition 1 *Let $T = T_1, \ldots, T_N$ be a sequence database over the alphabet Σ. Let $T_{i,j}$ denote a subsequence of T_i starting at index j, for $1 \leq i \leq N$ and $0 \leq j < |T_i|$, where $|T_{i,j}| \leq |T_i| - j$. Given a query pattern $Q \in \Sigma^*$ and a range r, a **Range query** is the problem of finding the Result set $R(Q, r)$ as the set of all the subsequences $T_{i,j}$ in T such that $ED(Q, T_{i,j}) < r$, where ED corresponds to the Edit Distance.*

One way to solve the *Range query* problem is as follows: Given a query pattern Q, compare all sequences stored in the database against Q using ED, either through direct application of dynamic programming[15, 18] or other popular heuristics[1, 11, 17, 4], and determine the set $R(Q, r)$. Although this approach is correct, it is not practical/scalable for two reasons. First, sequence databases may involve a large number of very large sequences(e.g. Chr_{22} as the smallest human chromosome[14] that consists of approximately 35 million base pairs) resulting in severe performance penalty. Secondly, the prohibitive computational cost of alignment or even heuristic-based sequence comparison makes it impractical, specially when $|R(Q, r)|$ is very small, compared to the total number of subsequences in T.

A solution could be mapping the sequence similarity(using *Edit Distance(ED)* for $R_{ED}(Q, r)$) problem into a numerical/vector difference(using *Frequency Distance(FD)* for $R_{FD}(Q, r)$) problem to benefit from much more time/space-efficient numerical methods in the literature. One way is to use a mathematical transformation to map the *sequence domain*(of sequences S_i) into a *vector/frequency domain*(of frequency vectors $f(S_i)$), and use an appropriate frequency distance function to estimate the edit distances of the sequence domain. If the right transformation is applied, then *Parseval Theorem* implies that *Frequency Distance* is less than or equal to *Edit Distance*, or for short $FD(f(S_i), f(S_j)) \leq ED(S_i, S_j)$ (*Distance preserving transformation*), with

the equality holding when all the transformation coefficients are used in the frequency distance calculations. This property is the main driving force behind using transformation. Furthermore:

- The calculation of distance in the frequency domain(FD), is much more *time/space-efficient* compared to the calculation of the distance in the original sequence domain(ED).
- Range queries are much more efficiently evaluated in the frequency domain. For instance, suppose having a query pattern frequency vector $f(Q)$, range r and a set of frequency vectors $f(S_1), \ldots, f(S_n)$, then all the frequency vectors(and in turn sequences) $f(S_{1 \leq i \leq n})$, *where* $FD(f(Q), f(S_i)) > r$ may be pruned from the answer set without the need to investigate further (to calculate the original ED), at a very low cost. This would dramatically reduce *i*) the *computational cost*[10] and, *ii*) the required amount of *search space* $R_{FD}(f(Q), r)$, for a given range query (Q, r), where $R_{ED}(Q, r) \subseteq R_{FD}(f(Q), r)$. However, a very important requirement is to guarantee that the ratio $\frac{|R_{FD}(f(Q),r)|}{|R_{ED}(Q,r)|} \geq 1$, not to incur any *false negatives*, while being as small as possible for the better filtration ratio.

Following definitions introduce the steps in transforming the *original domain(set of sequences)* to *frequency domain(set of vectors)*:

Definition 2 *Let $S = s_1, \ldots, s_n$ be a sequence over the alphabet $\Sigma_k = \{\alpha_1, \ldots, \alpha_k\}$, then the frequency vector of S, called $f(S)$ is defined as: $f(S) = [f_1, \ldots, f_k]$, where $f_i (\geq 0)$ corresponds to the occurrence frequency of α_i in S, and $\sum_{i=1}^{k} f_i = |S| = n$. Similarly, ξ_{s_i} is defined to be a $|\Sigma| \times 1$ vector where the entry corresponding to $\alpha_i \in \Sigma_k$, for $i \leq k$, is 1 and all other entries filled with 0, or in other words:*

$$\xi_{s_i} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_k \end{bmatrix}, \; where \; \forall i \leq k, \; f_i = \begin{cases} 1 & \alpha_i \\ 0 & \text{otherwise.} \end{cases}$$

For instance, let $S = AGGTTGCAATTA$ be a sequence over $\Sigma_4 = \{A, C, G, T\}$, then $f(S) = [4, 1, 3, 4]$, and $\xi_{s_1}^T = [1, 0, 0, 0]$, $\xi_{s_2}^T = [0, 0, 1, 0]$, ..., $\xi_{|S|}^T = \xi_{12}^T = [1, 0, 0, 0]$, respectively.

Definition 3 *Let $S = s_1, \ldots, s_n$ be a sequence from the alphabet Σ_k, Frequency-Quantization of S, called $S^F = [s^1, \ldots, s^n]$ is a sequence of $|\Sigma| \times 1$ vectors s^i, one for each ordered base of the sequence, where $s^i = \xi_{s_i}$, for $1 \leq i \leq n$.*

Let S be the same sequence as given above, then

$$S^F = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}.$$

Following definitions, capture two of the famous distance preserving transformations which we deployed in our study.

Definition 4 *The k^{th}-level Haar Wavelet Transformation(DWT)[10] of a frequency-quantized sequence S, $\varpi_k(S)$, for $0 \leq k \leq log_2 n$, is defined as $\varpi_k(S) = [v_{k,0}, v_{k,1}, \ldots, v_{k,\frac{n}{2^k}}]$, where $v_{k,i} = [\alpha_{k,i}, \beta_{k,i}]$, for*

$$\alpha_{k,i} = \begin{cases} f(c_i) & k = 0 \\ \alpha_{k-1,2i} + \alpha_{k-1,2i+1} & 0 < k \leq log_2 n, \end{cases}$$

$$\beta_{k,i} = \begin{cases} 0 & k = 0 \\ \alpha_{k-1,2i} - \alpha_{k-1,2i+1} & 0 < k \leq log_2 n, \end{cases}$$

where for $k = log_2 n$: $\alpha_{log_2 n,0} = f(S[0 : n-1])$ and $\beta_{log_2 n,0} = f(S[0 : \frac{n}{2} - 1]) - f(S[\frac{n}{2} : n-1])$ represent the first and second Haar wavelet coefficients, respectively.

For instance, for the same S as given above, the 3^{rd}-level DWT of S: $\varpi_3(AGGTTGCAATTA) = \{\alpha_{3,0}, \beta_{3,0}\} = \{[4, 1, 3, 4], [-2, -1, 3, 0]\}$, represents the set of first and second wavelet coefficients.

Definition 5 *The n-point Discrete Fourier Transformation(DFT) of a sequence $S = [S_t]$, for t=0,...,n-1 is defined to be a sequence X of n complex numbers x_f of $|\Sigma| \times 1$ vectors, for $f = 0, \ldots, n-1$, and is given by*

$$x_f = \frac{1}{\sqrt{n}} \sum_{t=0}^{n-1} S_t e^{\frac{-j2\pi ft}{n}},\ f = 0,\ 1,\ \ldots,\ n\text{-}1,$$

where $j = \sqrt{-1}$ *is the imaginary unit. The original sequence* S *can be restored by the inverse transform:*

$$S_t = \frac{1}{\sqrt{n}} \sum_{f=0}^{n-1} x_f e^{\frac{j2\pi ft}{n}},\ t = 0,\ 1,\ \ldots,\ n\text{-}1,$$

where x_f *is a complex number and its real and imaginary parts are* $|\Sigma| \times 1$ *vectors.*

Let $S' = ACCT$, the first and second DFT coefficients of S' are calculated as: $X_0(S') = [\frac{1}{2}, 1, 0, \frac{1}{2}]$ and $X_1(S') = \{[\frac{1}{2}, \frac{-1}{2}, 0, 0], [0, \frac{-1}{2}, 0, \frac{1}{2}]\}$, respectively.

Meanwhile, There is one question to be answered: *What would be the proper FD distance to deploy in the frequency domain to provide a better/tighter approximattion of the Edit Distance of the original space?*

Within the context of frequency transformations in multi-dimensional indexing, L_p-norm$(p > 0)$ distance measures are usually the popular choice for frequency distance function. However, this choice is very much application-dependent. Hence, we decided to run exhaustive experiments on DNA sequences using L_1-norm, L_2-norm, (FD_1, FD_2)[10] distance functions, to analyze the accuracy level of each of them. We should use the FD which demonstrates a tighter bound estimate of the ED in the original domain, or in other words "*more precisely*" reflecting the similarity/distance across the sequences.

Figure 1, shows the average distance comparison resulting from running an All-Pair-All exhaustive frequency sequence comparison[18] on the contig sequences of *Alu*, $Mitochondria$, *Escherichia coli*(*E.coli*), $Yeast$ and $Drosophila$[14]. All the contigs are first transformed into frequency domain using wavelet transformation and then the distance across the extracted vectors for the first and second coefficients are calculated using the mentioned distance functions. L_1-norm empirically demonstrates a tighter bound(higher distance), compared with (FD_1,FD_2)[10] and L_2-norm on all the databases. Higher the FD, a better estimate of ED is achieved, while $FD \leq ED$ by *Parseval Theorem*. For any two frequency vectors X, Y: L_1-norm as the desired frequency distance is defined as $L_1(X, Y) = \sum_j |x_j - y_j|$, and may be expressed as the minimum number of increment/decrement(± 1) operations needed(on the entries) to transform vector X into vector Y, which is a lower bound on the original domain's ED distance. L_1-norm is also expected to have a lower computational cost compared with the others.

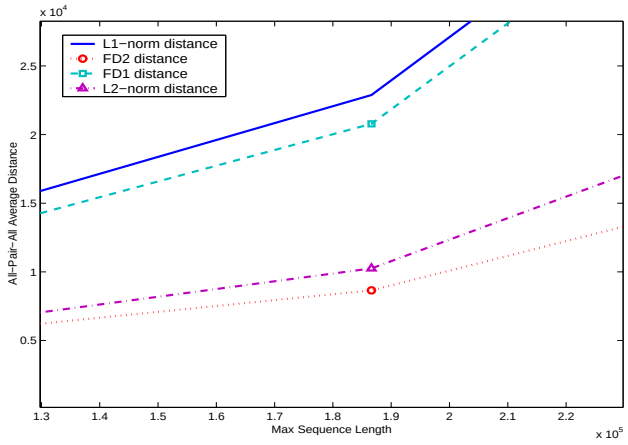

Figure 1: Average distance comparison resulted from contig DNA databases of *Alu*, *Mitochondria*, *Escherichia coli*(*E.coli*), *Yeast* and *Drosophila*[14], using L_1-norm, L_2-norm and (FD_1, FD_2)[10].

4 Transformation procedure

We perform the process in two different stages(depicted in Figure 2), *offline* and *online*, as follows:.

- *Preprocessing phase(Offline)*: Given the database of sequences $T = \{T_1, T_2, \ldots, T_n\}$, $\forall\ T_i \in \Sigma^*$:

  1. Let T_s be the sequence with the minimum length, say m. Choose the window size $|w|$ to yield an optimal total number of blocks on that sequence as $j = \theta(m/log_\Sigma N)$[12, 13], for database size N. This optimal j has been suggested for pattern partitioning, however the length of the pattern might not be known in advance, so we restrict the maximum size of the pattern by the size of the minimum sequence in the database, to be able to benefit from the optimal partitioning.

  2. Slide the window on each of the original DNA sequences T_i, and extract the corresponding $|w|$-sized blocks. Partition each T_i on the starting positions of $0, \frac{|w|}{2}, \frac{2|w|}{2}, \ldots$ into a total of $\frac{|T_i|-|w|+1}{\lfloor \frac{|w|}{2} \rfloor}$ blocks. Let $b_{i,j}$ denote the block extracted from sequence T_i, at position j, where $1 \leq i \leq n$, and $0 \leq j < |T_i| - |w|$.

  3. Perform *Frequency-Quantization*(Def. 3) on each of the subsequences/blocks $b_{i,j}$, extracting $b^F_{i,j}$,

4. Use the desired DFT/DWT transformations on each of the *Frequency-Quantized* blocks $b^F_{i,j}$, and calculate the corresponding transformed vector $X(b^F_{i,j})$ or $\varpi(b^F_{i,j})$ coefficients in the frequency domain.

5. Extract and store only a few coefficients(at most *three*) to represent the original subsequence/block. For the case of DFT, we keep the highest energy-concentrated-coefficients as, first, last and the second[21]. For the DWT, we keep the first and second coefficients, in which the energy of the sequence is expected to be mostly concentrated.

6. Build an offline index structure as follows: For each of the sequences in the database, T_i, keep a list of block vectors contained in that sequence. We keep only an index for the location of the extracted block, and the corresponding fixed-size frequency vector(s), which are at most two for DWT, and three for DFT. The higher precision might be achieved by choosing more coefficients, as needed, which is a built-in feature in our implementation.

- *Query processing(Online)*: Given a query pattern $Q \in \Sigma^*$ and range r:

1. Slide the $|w|$-sized window on the pattern sequence Q, partitioning it into non-overlapping segments of length $|w|$, for a total of $j' = \lfloor |Q|/|w| \rfloor$ partitions. Let Q_l denote the partition of Q starting at index l, where $1 \leq l \leq |Q| - |w| + 1$.

2. For each of the extracted partitions Q_l:
   - Perform *Frequency-Quantization*(Def. 3) on Q_l partition/block, and calculate Q^F_l,
   - Use DFT/DWT transformations on the *Frequency-Quantized* blocks Q^F_l, and extract the corresponding $X(Q^F_l)$ or $\varpi(Q^F_l)$ coefficient vectors,

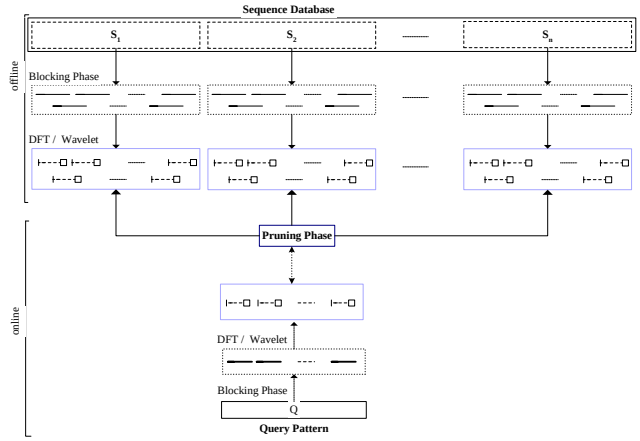

Figure 2: The transformation procedure.

– Search each of the coefficient block vectors, $b^F_{i,j}$, stored in the *offline* index, and prune all the subsequences/blocks which,

  * DFT: $FD(X(Q^F_l), X(b^F_{i,j})) > \frac{r}{j'}$, or
  * DWT: $FD(\varpi(Q^F_l), \varpi(b^F_{i,j})) > \frac{r}{j'}$.

3. Calculate ED only for the candidate result set(those not pruned), to find the subsequences S_i, where $ED(Q, S_i) < r$.

5 Performance Analysis

5.1 Implementation

We compared the application of the transformation techniques with two other approaches. The first one so called *String* is the q-gram indexing method used by QUASAR[4], and the second one so called *Vector*, is a more space efficient(trading for time) version of *String* for the inverted table index structure that is used in the approach.

The *String*, uses a *block addressing scheme* as follows: Each of the contigs of the database, and the pattern sequence, are partitioned into blocks of fixed size $|w|$(as calculated in the previous section), for a total of B blocks in database, and a counter C_{b_i} is associated with each block b_i of the database, respectively. For the q-gram of size q, an index structure of size $|\Sigma|^q$ is maintained($q = 3$). Each entry corresponds to a unique q-gram, followed by a list of blocks containing that particular

q-gram. All the q-grams of the blocks of a pattern are inspected(consecutive q-grams of the blocks overlap in $q-1$ bases), and the counter C_{b_i} is incremented whenever a search for that q-gram reports *"existing"* in the b_i. After processing all the blocks and the corresponding q-grams of the pattern, each counter C_{b_i}(where $0 < i \leq B$) indicates how many unique q-grams(ignoring the positional information) from Q are contained in that block b_i, of the database. Thereafter, all the block counters are stored in an array of size $|T|/B$, and the blocks b_i, whose counters C_{b_i} contain(share) less than $max(|Q|, |b_i|) + 1 - (r+1) \cdot |q|$ of q-grams with the pattern, are pruned from the candidate set[9, 4]. We used a uniform blocking method across all different methods. Note that *String* q-gram method is an approximate method, in contrary with dynamic programming algorithm, and potentially suffers from false positives.

The second approach called *Vector*, is very similar to the *String* method. However, the index structure saves on the total amount of space needed to keep for each block. Accordingly, for each block b_i, the corresponding frequency vector(Def. 2) $f(b_i)$ is calculated. Each $f(b_i)$ is mapped into an integer as follows: Let $f(b) = [f_1, f_2, f_3, f_4]$, then the corresponding identifier for b_i, called I_{b_i}, is defined as

$$I_{b_i} = \sum_{k=1}^{4} f_{k-1} |b_i|^{|\Sigma|-1}.$$

The same identifier translation was used to map each q-gram to an integer. This would decrease the number of entries in the index structure exponentially(for $q \gg 3$) which would be of size $q^{|\Sigma|}$. Each q-gram/block vector υ is mapped into an integer value I_υ, where $|\upsilon| \leq I_\upsilon \leq |\upsilon|^{|\Sigma|}$. However, this is not a 1-to-1 mapping, therefore only the nonzero entries are kept. The rest of *Vector* process is exactly like *String*. However, as expected, it is supposed to incur more false positives, on average, compared with *String*, due to the degeneracy of our mapping and loss of the positional information during translation.

We also implemented a third improvement, called *Tuple*. Each of the blocks are stored as a $|\Sigma|^q$-dimensional frequency vector(zero entries might be neglected), where each entry i corresponds to the quantity of the unique q-gram q_i in that block. As for the index structure, we would not need any space to keep for the q-gram index as in *String*[4], while the block transformed vectors already include its containing q-gram information and the corresponding counters per q-gram is implied by each entry. This technique provided exactly the same result as *String*, however was

much more time-efficient. Hence, it is not included in our pruning graphs but is included in our timing comparison table(Table l).

For a database of N sequences, containing B blocks, *String* constitutes $(N+\frac{B}{2})(\text{int}) = O(B)(\text{int})$ space($B \gg N$), meaning linear in space. However its computational cost is $O(|Q|B)$. Similarly, *Vector* has the same computational complexity, however, being exponentially more space-efficient (only for $q \gg 3$) at the cost of more false positives on average. The amount of saved space would be approximately equal to

$$\lim_{q\to\infty} \frac{|\Sigma|^q}{q^{|\Sigma|}}.$$

Tuple is $O(|\Sigma|^q \cdot B)$-space and $O(B)$-time which is linear in total number of blocks. We could as well use a tree-based approach[7] and reduce the search time to $O(logB)$. With regards to all of the above methods, we also incorporated different blocking and q-gram partitioning methods:

- *Incremental* partitioning: Each of the consecutive q-grams/blocks of length t, overlapping by $t-1$ residues,
- *HalfOverlap* partitioning: Each of the consecutive q-grams/blocks of length t, overlapping by $t/2$ residues, and
- *non-overlapping* partitioning.

On both of q-gram and block partitioning, more the q-gram/blocks extracted, we observed higher computational cost, better filtration ratio, tighter FD bound and a smaller candidate set. This choice is a trade-off between cost versus precision. However, due to the limitation of the space, we did not include those results in this study. We implemented all the desired algorithms and transformations using $Java$, and ran our simulations on a PIII-800Mhz with 1GB of main memory.

5.2 Considerations

Following observations should be expected regarding any transformation technique:

- Frequency Distance(FD) should be a fair approximation to the original Edit Distance(ED).
- More representative coefficients chosen in the frequency domain, then a higher precision on the real distance approximation and more filtration efficiency, is to be observed.

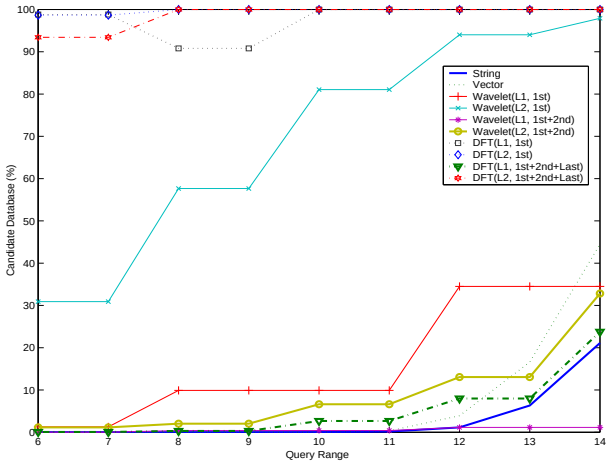

Figure 3: The resulting candidate answer set as a function of query range on *Alu* database.

- Less the FD, a *more compact space* is to be observed. This property relies upon the fact that a decrease in FD value, would result in sequence vectors being located closer to each other in frequency space. We could as well store a trail of quantized vectors as a *Minimum Bounding Rectangle(MBR)*, in which case, the number of *MBR*s needed to store the sequence vectors of the data in frequency domain will be minimized, at the cost of less efficient filtration ratio.

- The calculation of FD should be, computationally, as cheap as possible, and keeping a minimum number of coefficients, should satisfy achieving a resonably efficient filetration.

- Filtration Ratio(FR) is to be maximized(incurring no false negatives), however the efficiency of pruning depends on: i)the structure of sequences, ii) Query sequence, and iii) Query range.

5.3 Simulation Results

Let $\circ$, $\diamond$ and $\star$ denote the use of 1^{st}, $(1^{st} + 2^{nd})$ and $(1^{st} + 2^{nd} + Last)$ coefficients of the transformed sequences, respectively. Figures 3-5, demonstrate the result of running *String*, *Vector* and application of DFT/DWT transformation techniques for the choice of different coefficients on *Alu*, *Mitochondria*, and *Escherichia coli(E.coli)* contig databases[14] for a random query pattern of length 16, as the query range varies from 4 to 14.

Let B denote the total number of blocks in the database. Vertical axis shows the fraction of the database that is left for further investigation(those not pruned), that is $\frac{|R_{FD}(f(Q),r)|}{B}\%$, and horizontal axis shows the corresponding query range. In Figure 3, as expected *Vector* gives more

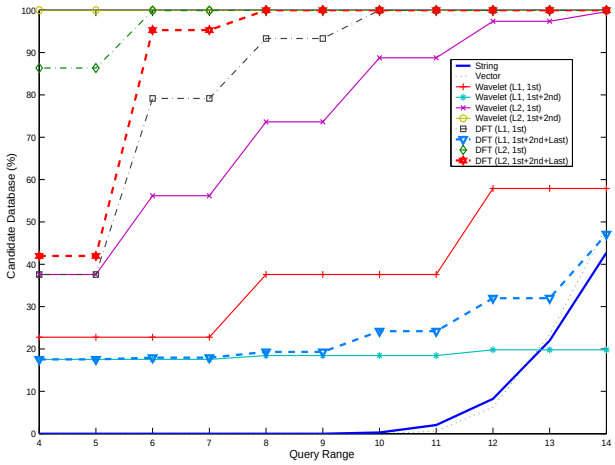

Figure 4: The resulting candidate answer set as a function of query range on *Mitochondria* database.

Species	**String**	**Vector**	**Tuple**	$DWT_{L_1,\diamond}$	$DWT_{L_2,\diamond}$	$DFT_{L_1,\star}$	$DFT_{L_2,\star}$
Alu	**4.67**	5.09	5.5	4.62	4.5	**5.21**	6.92
Mitocondria	**1921.7**	3209.9	176.74	152.3	152.6	**175**	236.2
E. coli	**534.7**	929	259.59	221.3	224.9	**289**	369.9

Table 1: The timing comparison(in *seconds*) of running 11 different range queries on the described techniques, for three *contig* datasets[14], for a random query of length 16.

false positives, and $DFT_{L_1,\star}$ demonstrates the best filtration, for $(23.37)^{-1} \leq FR_{Alu} \leq (0.07)^{-1}$ incurring no false negatives for the inspected query range. On the other hand, $DWT_{L_1,\diamond}$ gives suspicious false negatives compared with *String*, which indicates that it does not have a very good performance. We investigate this behavior later in the section. A similar behavior is observed in Figure 4, $DFT_{L_1,\star}$ gives the best filtration with no false negatives, for $(47.13)^{-1} \leq FR_{Mito} \leq (17.54)^{-1}$, but $DWT_{L_1,\diamond}$ incurs suspicious false negatives as before. Figure 5, depicts the best expectations, no false negatives of any kind on any of the transformations. Again $DFT_{L_1,\star}$ gives the best estimates with $(29.21)^{-1} \leq FR_{E.Coli} \leq (0.02)^{-1}$. In all the figures, using more coefficients(or L_1 over L_2) leads to more efficient filtration, validating our study in Figure 1.

Table-1 shows the timing comparison, resulting from running 11 different range queries(4-14) on three real datasets. Compared with *String* and *Vector* q-gram methods, transformation always works faster(including the offline index construction overhead), and gets even more explicit as the dataset grows, for a 11-13 times faster run, while very closely approximating the *String* q-gram

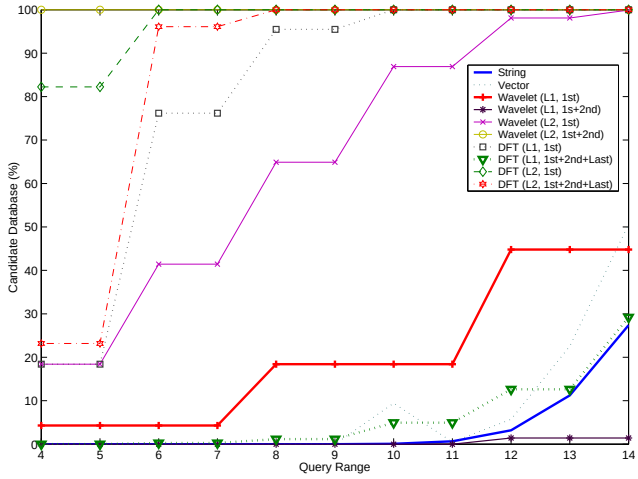

Figure 5: The resulting candidate answer set as a function of query range on *Escherichia coli(E.coli)* database.

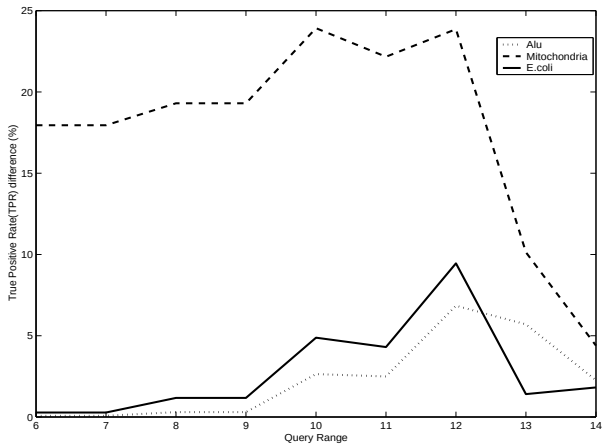

Figure 6: *True Positive Rate(TPR)* of $DFT_{L_1,\star}$ compared with *String* q-gram method on *Alu*, *Mitochondria* and *E.coli* datasets.

method. Figure 6 shows the distribution of *True Positive Rate(TPR)* of $DFT_{L_1,\star}$ compared with *String*, on the very same range queries as investigated earlier. For instance, joining the results of Figures 3-6 and Table-1, for the case of *Mitochondria* dataset, $DFT_{L_1,\star}$ may effectively prune up to $100 - (17.54)^{-1} \simeq 99\%$ of the database, for $\frac{1921.7}{175} \simeq 9\%$ of the total time needed for *String* q-gram method, while incurring no false negatives. However, as we mentioned before, the transformation application is unfortunately very much data dependent. Therefore, we ran the experiments on a much wider range of environments and inspect the performance improvements. Figures 7-9, demonstrate the results of producing 100 random query patterns Q, $(8 \leq |Q| \leq 64)$, and performing the range queries for $1 \leq r \leq |Q|$, however due to space limitations, only the results for pattern

sizes of length 16 and 64 are shown. For all the datasets and on all the experiments, it can be observed that the $DFT_{L_1,\star}$ transformation does not produce any false negatives up to the range query $r \leq |Q| - \varepsilon$, for $\varepsilon \leq 3$, however due to the blocking method used in *String* method, more false positives are expected[4], meaning that what has been seen as false negative for $DFT_{L_1,\star}$ might not be false negative after all. So apparently, we may effectively use the transformation as a preprocessing phase to prune irrelevant sequences for proximity ranges not very close to the query pattern's length($r \leq |Q| - \varepsilon$). Within this domain, $DFT_{L_1,\star}$ performs very well. This questionable behavior needs further investigation. For this purpose, we investigated every single candidate block produced by *String*, $DFT_{L_1,\star}$ and $DWT_{L_1,\diamond}$ on a random portion of *Alu* database, for some random query patterns of length 16, and performed a range query of 14, which had caused the suspicious behavior on all the datasets, and inspected the corresponding precision and recall. On the inspected configurations, *String*, $DFT_{L_1,\star}$ and $DWT_{L_1,\diamond}$ filtrations, reduced the database size to 7.75%, 14.08% and 1.41%, respectively. $DWT_{L_1,\diamond}$ produced false positives, in addition to a couple of false negatives! This would need further investigation on a larger scale. However, $DFT_{L_1,\star}$ caught all the actual k-distant blocks of the database. Thereafter we inspected the recalls. *String* depicted a better performance which was expected, by producing less false positive compared with $DFT_{L_1,\star}$. However, $DFT_{L_1,\star}$ did not miss any actual k-distant block, while $DWT_{L_1,\diamond}$ generated false negatives and misses the correct result. Both $DFT_{L_1,\star}$ and *String* resulted in a precision of 1, not missing any correct result, in contrary with $DWT_{L_1,\diamond}$. These results are depicted in Figures 10-13. The calculation of the distance in *String* was based on the difference on the number of shared q-grams, however, we used the frequency vector difference for DFT and DWT. For this reason, none of the sets of false positives produced by DFT and *String* were subset of one or another. The intersection of the results produced by them, consisted of all the k-distant blocks in addition to a few false positives.

6 Conclusion

In this paper, we studied the application of *Fourier(DFT)* and *Haar Wavelet(DWT)* transformations on biological sequences and evaluated the specific problem of range query. Transformation methods (e.g. DFT), may be applied to prune most of the non-desired sequences, and reduce the real search problem to only a fraction of the database, as a preprocessing phase for any of the

known heuristic approaches like BLAST[1], PatternHunter[11], QUASAR[4], FastA[17], and even the dynamic programming sequence alignment[18, 15]. Our results show that applying the transformation technique, a high accuracy and faster database pruning is achieved, when the behavior of the studied transformations is taken into account before applying the appropriate range query. The filtration ratio is very much data dependent and no generalization on the min/max filtration ratio or true positive rates can be suggested. However, the empirical results show a promising performance behavior, specially on $DFT_{L_1,\star}$, incurring no false negatives, high filtration ratio, while being considerably faster than the q-gram method. We plan to explore these observations on a larger scale in our future work.

References

[1] S. Altschul, W. Gish, W. Miller, E. Myers, and D. J. Lipman. Basic local alignment search tool. *J. Mol. Biol.*, 215:403–410, 1990.

[2] A. Apostolico. The myriad virtues of subword trees. *Combinatorial Algorithms on Words, NATO ISI Series, Springer-Verlag*, pages 85–96, 1985.

[3] A. D. Baxevanis and B. F. Francis Ouellette. *Bioinformatics: A Practical Guide to the Analysis of Genes and Proteins*. Wiley Interscience, second edition, April 2001.

[4] S. Burkhardt, A. Crauser, P. Ferragina, H.P. Lenhof, E. Rivals, and M. Vingron. q-gram based database searching using a suffix array (quasar). In *RECOMB*, pages 77–83, 1999.

[5] E. Chavez and G. Navarro. A metric index for approximate string matching. In *LATIN*, pages 181–195, 2002.

[6] M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt. A model of evolutionary change in proteins. *Atlas of Protein Sequence Structure. National Biomedical Research Foundation, Washington, DC*, 5:345–352, 1978.

[7] E. Giladi, M. G. Walker, J.Z. Wang, and W. Volkmuth. Sst: an algorithm for finding near-exact sequence matches in time proportional to the logarithm of the database size. 18(6):873–877, 2002.

[8] S. Henikoff and J.G. Henikoff. Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci.*, 89:10915–10519, 1992.

[9] P. Jokinen and E. Ukkonen. Two algorithms for approximate string matching in static texts. *Proc. MFCS'91*, 16:240–248, 1991.

[10] T. Kahveci and A.K. Singh. Efficient index structures for string databases. *VLDB*, pages 351–360, 2001.

[11] B. Ma, J. Tromp, and M. Li. Patternhunter: faster and more sensitive homology search. *Bioinformatics*, 18(3):440–445, March 2002.

[12] G. Navarro and R.A. Baeza-Yates. A hybrid indexing method for approximate string matching. *J. of Discrete Algorithms*, 1(1):205–239, 2000.

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

[14] NCBI. National center for biotechnology information(ncbi) website. http://www.ncbi.nih.gov/.

[15] S. B. Needleman and C.D. Wunsch. General method applicable to the search for similarities in the amino acid sequence of two proteins. *J. of Mol. Biol.*, 48:443–453, 1970.

[16] University of Chicago's Howard Hughes Medical Institute. New type of dna-free inheritance in yeast is spread by a madcow mechanism. http://www.uchospitals.edu/news/1997/19970530-prion-fibers.html, May 1997.

[17] W. R. Pearson. Using the fasta program to search protein and dna sequence databases. *Methods Mol Biol*, 25:365–389, 1994.

[18] R. Smith and M. S. Waterman. Identification of common molecular subsequences. *J. Mol. Biol.*, 147:195–197, 1981.

[19] J. D. Thompson, D. G. Higgins, and T. J. Gibson. Clustal w: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position specific gap penalties and weight matrix choice. *Nucleic Acids Res.*, 22:4673–4680, 1994.

[20] David Wheeler. Weight matrices for sequence similarity scoring. http://www.techfak.uni-bielefeld.de/bcd/Curric/PrwAli/nodeD.html, May 1996. Department of Cell Biology, Baylor College of Medicine, Houston, Texas.

[21] Y. Wu, D. Agrawal, and A. El Abbadi. A comparison of dft and dwt based similarity search in time-series databases. *CIKM*, pages 488–495, 2000.

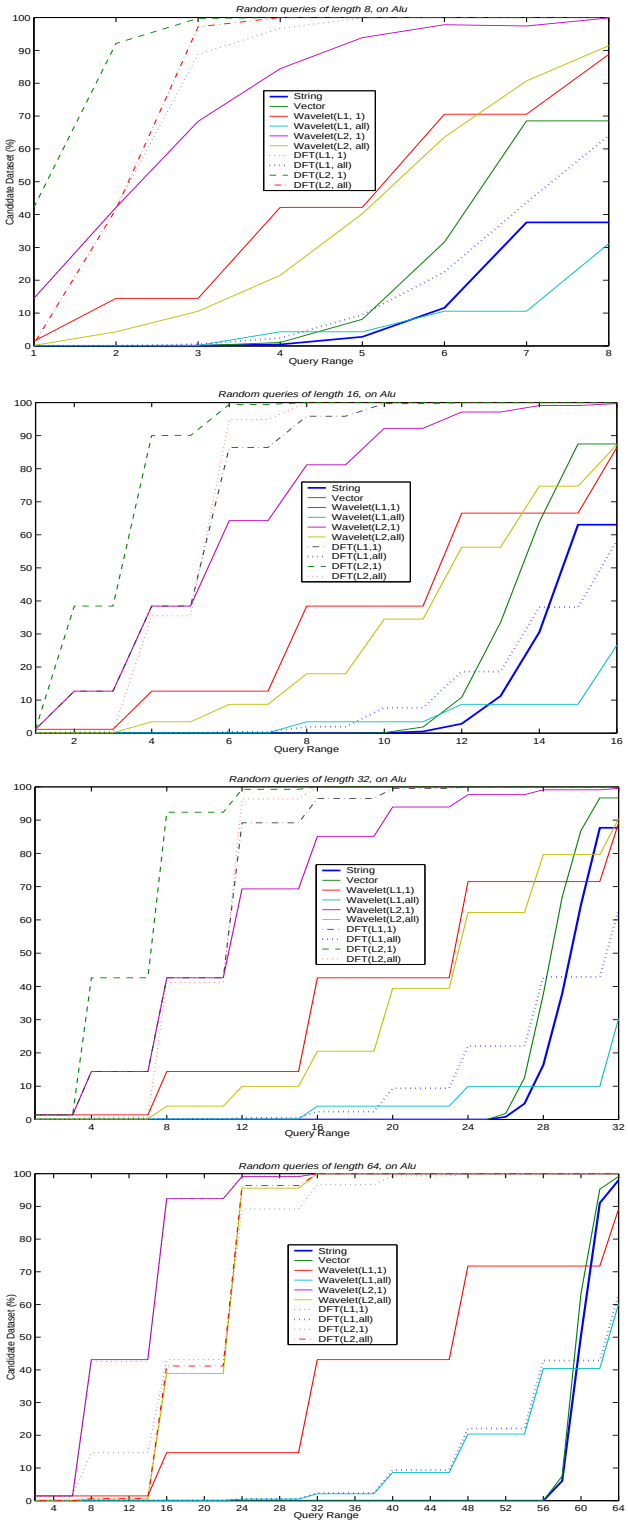

Figure 7: Random query patterns of various length, and range queries on *Alu*.

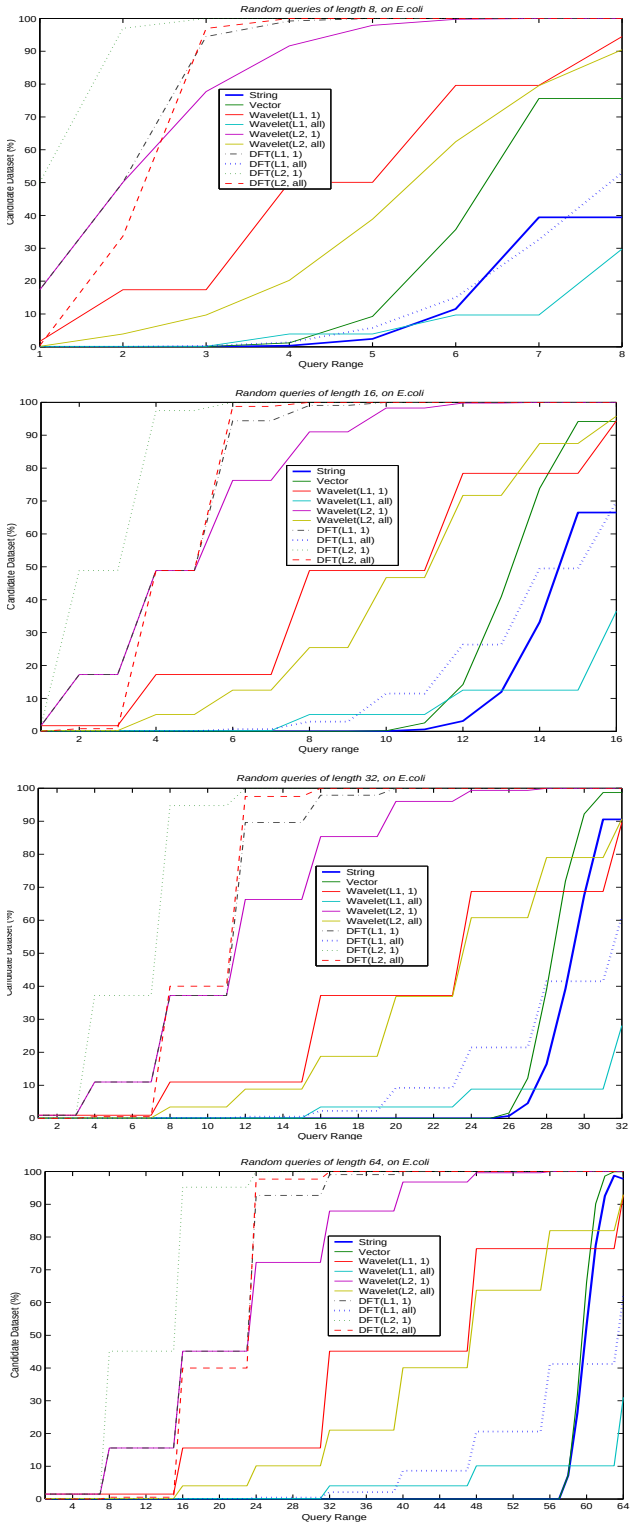

Figure 8: Random query patterns of various length, and range queries on *Escherichia coli(E.coli)*.

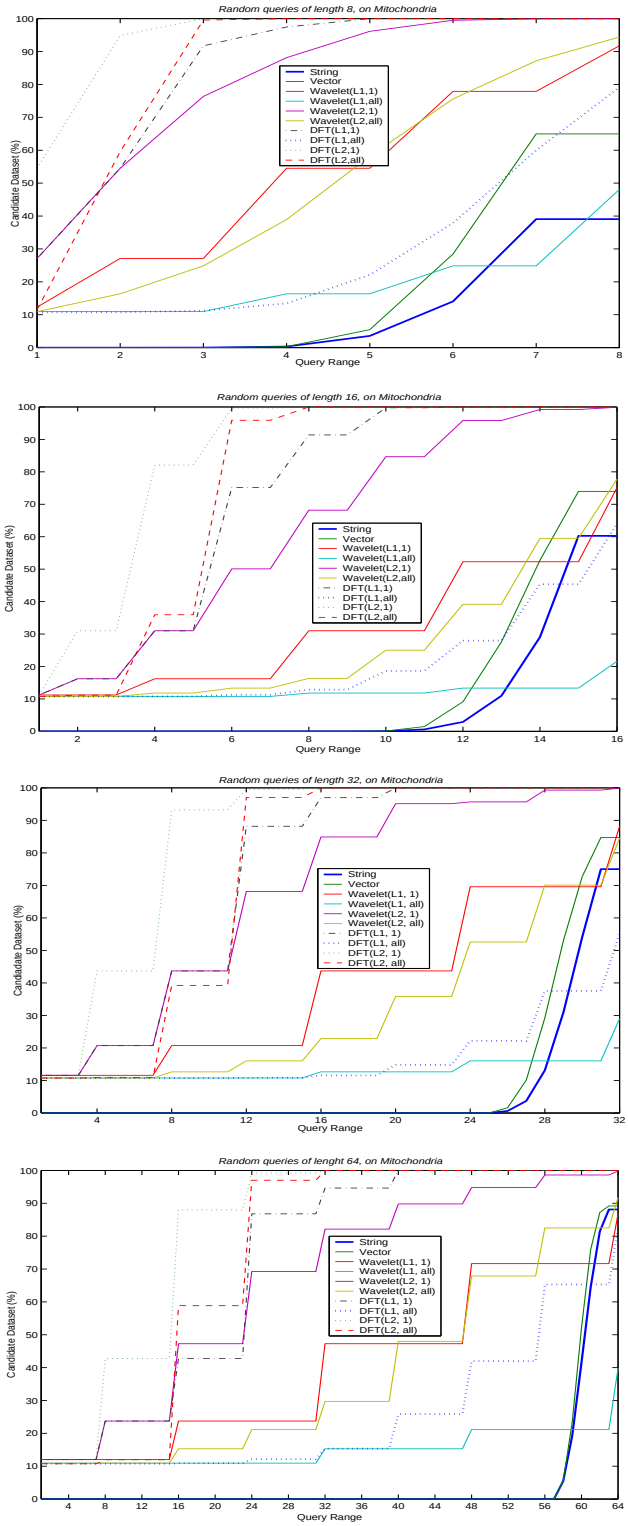

Figure 9: Random query patterns of various length, and range queries on *Mitochondria*.

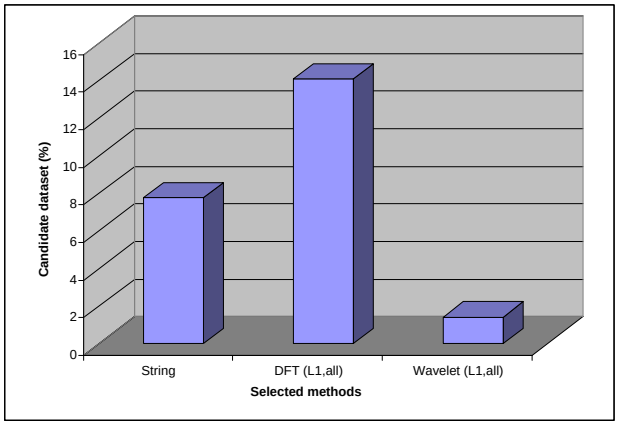

Figure 10: Resulting candidate percentages of the *Alu* database for *String*, $DFT_{L_1,\star}$, and $DWT_{L_1,\diamond}$.

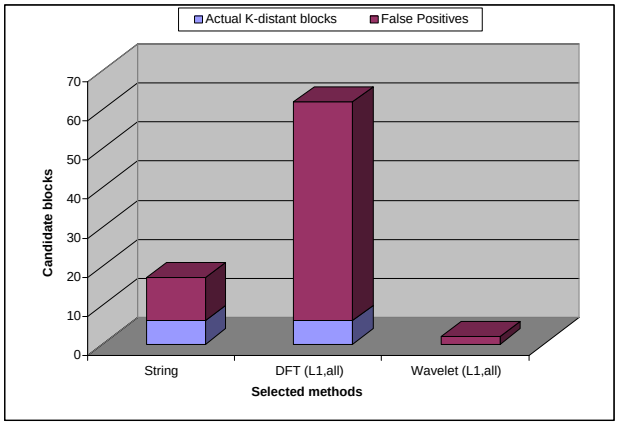

Figure 11: Resulting candidate block distribution of *String*, $DFT_{L_1,\star}$, and $DWT_{L_1,\diamond}$.

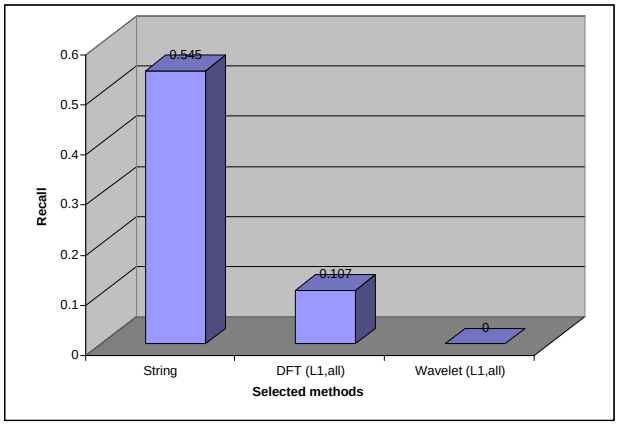

Figure 12: Resulting recalls for *String*, $DFT_{L_1,\star}$, and $DWT_{L_1,\diamond}$.

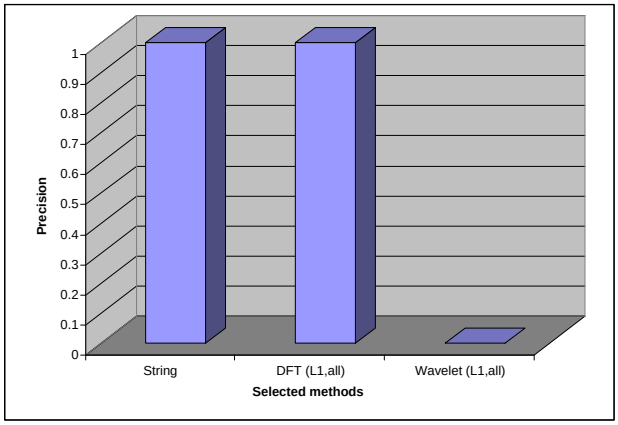

Figure 13: Resulting precisions for *String*, $DFT_{L_1,\star}$, and $DWT_{L_1,\diamond}$.