BFT: A Relational-based Bit Filtration Technique for Efficient Approximate String Joins in Biological Databases

S. Alireza Aghili, Divyakant Agrawal, and Amr El Abbadi
Department of Computer Science,
University of California, Santa Barbara, CA 93106.
{aghili, agrawal, amr}@cs.ucsb.edu

Abstract. Joining massive tables in relational databases have received substantial attention in the past decade. Numerous filtration and indexing techniques have been proposed to reduce the curse of dimensionality. This paper proposes a novel approach to map the problem of pairwise whole genome comparison into an approximate join operation in the well-established relational database context. We propose a novel Bit Filtration Technique (BFT) based on vector transformation and furthermore conduct the application of DFT(Discrete Fourier Transformation) and DWT(Discrete Wavelet Transformation, Haar) dimensionality reduction techniques as a pre-processing filtration step to effectively reduce the search space. BFT reduces the search space and the running time of the join operation drastically. Our empirical results on a number of Prokaryote and Eukaryote DNA contig databases, demonstrate up to 99.9% filtration ratio to efficiently prune non-relevant portions of the database, incurring no false negatives, with up to 50 times faster running time compared with traditional dynamic programming, and q-gram extraction approaches. BFT may easily be incorporated as a pre-processing step for any of the well-known sequence search heuristics as BLAST, QUASAR and FastA, for the purpose of pairwise whole genome comparison. Additionally, we discuss the integration of our proposed techniques for more efficient approximate join in the text databases, data integration, and data cleansing. We analyze the precision of applying BFT and other transformation-based dimensionality reduction techniques, and finally discuss the imposed trade-offs.

1 Introduction

Traditional query languages and relational databases have been mainly designed for exact query search, and the problem of similarity search and the corresponding applications have been extensively studied within the past decade, especially in the context of biological databases. However, not enough advances have been made to address the need for approximate queries. In particular, mainstream database research has not paid

substantial attention to the issue of approximate pairwise whole genome similarities. Errors and modifications are observed in a variety of applications originating from typographical mistakes(*Data cleansing*), inconsistent attribute design conventions(*Data integration*), or even being part of a natural mutational mechanism(*Genomics*). Each of these events may result in a series of changes on the original strings from a global point of view. The approximate search seeks the sequences close enough to a given query sequence either through direct alignment[15, 17] or using other heuristics [1, 16, 18]. For instance, approximate sequence analysis has enabled the detection of certain strains of the *Escherichia coli(E.coli)* bacteria responsible for infant *diarrhea* and *gastroenteritis*. Similarly in large and modern enterprises, it is inevitable that different branches of the organization would need a large amount of external data, retrieved from other resources, to be integrated into the existing database. Such data would most probably use a different schema and/or tuple representation conventions, probably generated by different database engines. Integration of such data sources (approximate join to suppress the duplicates) leads to an enormous challenge since the corresponding database relations might each include hundreds of millions of records(e.g. digital libraries). Looking for pairwise whole genome homology search, or very large scale string joins, neither the dynamic programming algorithms nor the heuristics [1, 16, 18, 13, 14, 7] may practically be applied. In such cases, the entire database should be searched, although most of the inspected strings may not actually result in the answer set. As a result, the expensive inspection of non-relevant strings impacts the performance dramatically.

The mentioned applications, motivations, and shortcomings trigger the necessity of incorporating efficient filtration techniques to leverage the complexity and scalability of the problem. In this paper we propose a novel approach to map the problem of pairwise whole genome comparison into an approximate join operation in the well-established relational database context. Furthermore, we apply the proposed *BFT(Bit-Filtration Technique)*, and additionally, *DWT(Discrete Wavelet, Haar)* and *DFT(Discrete Fourier Transformation)* as pre-processing filtration techniques. Our simulations study the approximate join operation and the corresponding filtration efficiency gained by the proposed techniques while dealing with relations with up to 1.3 billion tuple comparisons in the worst case.

The rest of the paper is organized as follows: Section 2, discusses the background and related work. Section 3 introduces the proposed techniques. Section 4 demonstrates a concise empirical performance analysis

and the simulation results followed by section 5, which concludes the work.

2 Background, Related Work

In a typical application of approximate join, given two string datasets S and T, and range r, all the string tuples of S are compared against all string tuples of T, in search for pairs of string tuples which are at most r edit operations far from each other. However as mentioned before, because of the quadratic time involved, the dynamic programming[17, 15] algorithms are not feasible. Several heuristics [4, 11, 3, 1, 16] have been proposed to speed up the similarity search phase of the procedure in the case of range query and k-nearest neighbor search. Most of these heuristics need to inspect the entire database while only a very small part of it might actually be of interest. To the best of our knowledge, this study is the first effort to *i*) facilitate efficient filtration for approximate join queries using discrete transformation techniques, and *ii*) map the problem of pairwise whole genome comparison into a relational approximate join operation of the database context.

Jin, Li, and Mehrotra[9] map the strings of database into Euclidean space and use d dimensions to represent each string in the feature space. Furthermore, a new range threshold δ for the new feature space is empirically found and all pairs of strings whose feature vector distances are greater than δ are pruned. However, *i*) the number of dimensions d is found empirically, which is very much data dependent, *ii*) range threshold δ, is found empirically by sampling random subsets of the database which results in *false negatives*! Gravano et al.[7] target the problem of approximate join in textual relational databases. They extract positional q-grams[10, 14] from each of the strings and apply count, positional, and length filtering to prune *out-of-range* string pairs. Furthermore, the SQL equivalents of the proposed operations are represented, and the work is also extended for edit distances with block shifts. Multi-Resolution index Structure(MRS)[11] uses a sliding window of size $|w|$ and extracts the first and second *Haar wavelet* coefficients of the corresponding windows. Given a range query (Q, r), MRS seeks the result set in different resolution levels of maximum postfix segments. However, *i*) MRS only addresses the problem of range query and k-nearest neighbor, and *ii*) the focus of the work is on the cost of their index structure/procedure, rather than the analysis of the filtration efficiency, and precision of the proposed approach.

Chavez and Navarro[4] 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, and further using triangle inequality to prune non-relevant sequences using Suffix Tree[2] as an index structure. No empirical analysis is conducted to evaluate this approach on real biological data. SST[6] uses overlapping sliding windows of size w over the database sequences and maps them into a $\mathbb{R}^{4^w}$-dimensional frequency vectors. Furthermore, SST uses k-means clustering algorithm to hierarchically cluster 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 [19] provide a concise study of DFT and DWT transformations, but only in the context of time-series databases.

3 Proposed Techniques

The traditional database join($\bowtie$) operation is based on the *exact matching* of string tuples, while the approximate join($\approx$) is based on the *approximate matching* of the string tuples. Initially, we perform the *block-based mapping*[1] procedure as depicted in Fig. 1. Given string databases T and S: **i)** perform the block-based mapping and extract the relational equivalent of T and S: A window of size b traverses each of the databases and extracts b-sized blocks overlapped by $b/2$ characters*(Steps 1,2)*, **ii)** the extracted blocks are then represented as attribute values in a relational database with their corresponding location in the original string database as the primary key*(Step 3)*, **iii)** given range r, the *approximate join* operation seeks all the corresponding tuple pairs of T and S, which are at most r far from each other based on a well defined distance function, usually being *Edit Distance*(ED)[1]*(Step 4)*. Block-based mapping facilitates the initial step of mapping the problem of pairwise whole genome comparison into an approximate table join.

[1] Note that, in our implementation, the database designer has the freedom of choosing non-uniform blocking factors across relations.

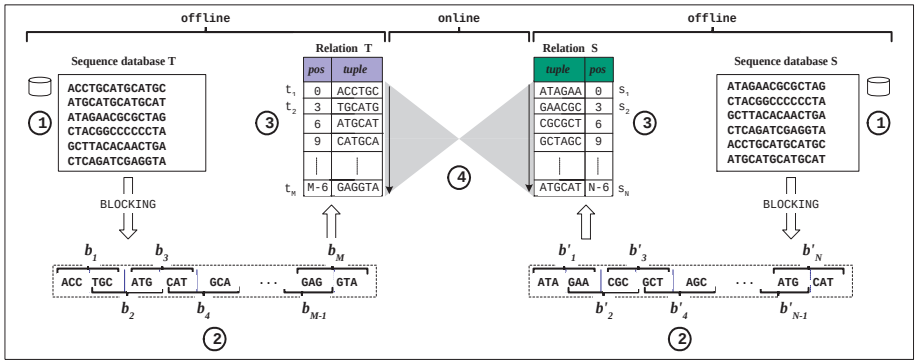

Fig. 1. The relational database conversion or block-based mapping procedure for $b = 6$

3.1 Terminology, Formulation

Following definitions[2] introduce the steps in transforming the *original domain(set of strings)* to *frequency domain(set of feature vectors)*:

Definition 1 ***(frequency vector)*** *Let $S = s_1, \ldots, s_n$ be a string 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 each $f_i (\geq 0)$ corresponds to the occurrence frequency of α_i in S, and $\sum_{i=1}^{k} f_i = |S| = n$.*

Definition 2 ***(frequency quantization)*** *Let $S = s_1, \ldots, s_n$ be a string from the alphabet Σ_k. The frequency quantization of S, $S^F = [\xi_{s_1}, \ldots, \xi_{s_n}]$, is a $(|\Sigma| \times n)$-dimensional matrix, where each orthonormal vector ξ_{s_i} represents the corresponding $(|\Sigma| \times 1)$-dimensional basis vector for s_i character, for $1 \leq i \leq n$.*

For instance, for S = AGGTTGCAATTA:

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

Definition 3 ***(approximate join)***
Let $T = [a_1^T, \ldots, a_g^T]$ and $S = [a_1^S, \ldots, a_h^S]$ be two database relations with their corresponding attributes. Suppose a_i^T and a_j^S are the two non-numerical attributes over some joint alphabet Σ, upon which we would like to perform approximate join. Given range r and distance function d,

[2] Due to space limitations, further definitions, proofs are provided in the Appendix.

approximate join of T and S, $T \bowtie_d^r S$*, returns all pairs of tuples* $(t, s) \in (T \times S)$ *such that* $d(t[i], s[j]) < r$*, for* $i \leq g$*, and* $j \leq h$ *.*

One way to solve the *aproximate join* problem, $\mathrm{T} \bowtie_d^r S$, is as follows: Given the relation S, compare all tuples of S against all tuples of relation T using ED as the distance measure, either through direct application of dynamic programming[15, 17] or other popular heuristics[1, 16, 3]. 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[12] 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 $|\mathrm{T} \bowtie_d^r \mathrm{S}| \ll |\mathrm{T} \times \mathrm{S}|$. A solution could be mapping the string similarity of the ED domain($\mathrm{T} \bowtie_{ED}^r \mathrm{S}$), into a vector difference in an acquired *Frequency Distance (FD)*[3] domain($\mathrm{T} \bowtie_{FD}^r \mathrm{S}$), to benefit from much more time/space-efficient numerical methods in the literature. One way is to use a mathematical transformation to map each *string* S_i, into its corresponding n-dimensional *frequency vector* $f(S_i)$, and use a lower-bound frequency distance function to approximate the edit distance of the original string domain.

The property shown in Theorem 1(Appendix), is the main driving force behind using transformation-based filtration. The calculation of distance in the frequency domain is linear in time/space, which is much more efficient compared to the calculation of the distance in the original string domain which is quadratic in time/space, hence, approximate join is much more efficiently evaluated in the frequency domain. Given a set of strings $\mathrm{S} = \{S_1, \ldots, S_n\}$ with their corresponding frequency vectors $f(\mathrm{S}) = \{f(S_1), \ldots, f(S_n)\}$, and range r, let T be a relation having only one tuple t and the corresponding frequency vector $f(t)$. Suppose we want to calculate $\mathrm{T} \bowtie_{ED}^r \mathrm{S}$, then all the strings S_i, for which $FD(f(t), f(S_i)) > r$ may be pruned from the answer set without the need to further calculate the edit distance. This property dramatically reduces the *computational cost*[11], and the required amount of *search space* for $\mathrm{T} \bowtie_{ED}^r \mathrm{S}$, while $(\mathrm{T} \bowtie_{ED}^r \mathrm{S}) \subseteq (f(\mathrm{T}) \bowtie_{FD}^r f(\mathrm{S}))$. However, a very important requirement is to guarantee that the *Filtration Ratio(FR)* $= \frac{|f(T) \bowtie_{FD}^r f(S)|}{|T \bowtie_{ED}^r S|} \geq 1$, not to incur any *false negatives*. A better filtration technique should lead to a smaller filtration ratio.

[3] We applied L_1-norm as the preferred FD, please refer to the details in Appendix...

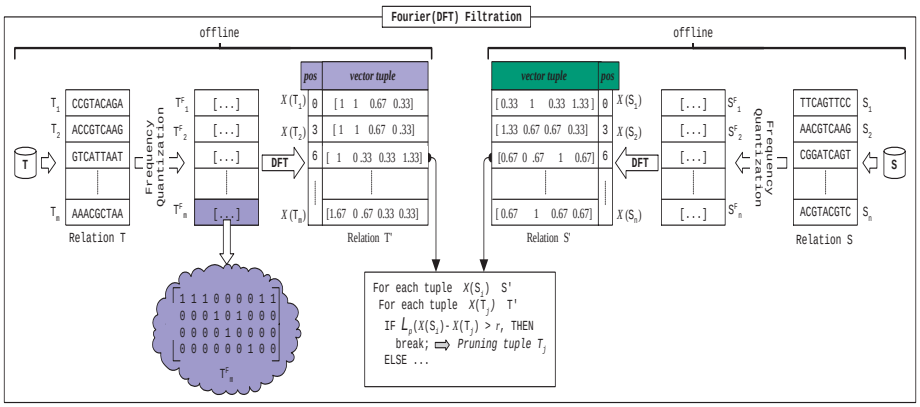

Fig. 2. The DFT filtration procedure for only one coefficient, and $b = 9$

3.2 Transformation-based filtration techniques

Discrete Fourier and Wavelet Transformation: The transformation-based filtration on both of DFT(Fig. 2) and DWT techniques, is identical except on the choice of transformation and the number of incorporated coefficients. The general process is shown under Algorithm 1. Given the genome databases S and T, the approximate join procedure is performed in two different stages: *offline* and *online*. In the offline stage, all the blocks/tuples S_i and T_j are extracted, and each tuple is mapped onto its corresponding feature vector(s) using DFT or DWT. Following this procedure, the S and T datasets would be mapped into S$'$ and T$'$ database relations, respectively. The tuple extraction procedure is very fast and needs only a single scan for each of the given databases. The actual approximate join operation is performed in the online stage. All the feature vector tuples of the relation S$'$ are compared against their T$'$ counterparts, and all those tuple pairs whose distance is greater than the given range r, are pruned from the resulting candidate set. Furthermore, a refinement step using dynamic programming is performed to remove the false positives. Additionally, a multidimensional indexing structure[5] could be built on the extracted relations for more efficient tuple pruning. However, we intend to study the impact of the various indexing schemes in our future work.

Bit Filtration Technique(BFT): Given two string databases T and S, we first construct the relational equivalent of each database as T= $\{T_1, \ldots, T_m\}$ and S= $\{S_1, \ldots, S_n\}$(Fig. 1), respectively. In the second step, the corresponding frequency vector(s), $f(T_i)(f(S_j))$, are extracted and as a result two new relations T$' = \{f(T_1), \ldots, f(T_m)\}$ and S$' =$

$\{f(S_1), \ldots, f(S_n)\}$ are constructed, respectively. The schema of T′(S′) has two attributes: ***i)*** The *index* of the tuple in the original database as the primary key, and ***ii)*** $|\Sigma|$-dimensional *frequency feature vector*. This process may be applied offline to the bigger relation(T′), and the smaller relation's frequency vectors may be extracted on the fly. Furthermore, the *Most Fluctuating Bit (MFB*, or *comparison bit)* of the frequency vector tuples of relation T′ is calculated. MFB corresponds to the k^{th} entry(column) of the frequency vector, holding the frequency of alphabet $\alpha_k \in \Sigma$, whose entry value across all frequency tuples $f(T_j)$ demonstrates the most discriminating deviation from the mean value(for $1 \leq k \leq |\Sigma|$ and $1 \leq j \leq m$).

Figure 3 depicts the steps of BFT procedure. BFT clusters the bigger relation T′, on its $1^{st}, ..., |\Sigma|^{th}$ MFB bit, in p multiple passes, for $1 \leq p \leq |\Sigma|$. When the algorithm starts, the entire relation is considered as one cluster. In the first pass, the frequency tuples are clustered based on their MFB entry value, in an increasing order. Given block size β, after the first pass, the relation T′ would potentially be clustered into $\beta + 1$ clusters$(c_0, \ldots, c_\beta)$ with no special order within each cluster, typically forming clusters of size $m/(\beta+1)$. The tuples belonging to c_j cluster, have the same value j, on their corresponding MFB entry. The second phase, subdivides each cluster into potentially $\beta + 1$ new clusters based on the values of their second MFB in an increasing order, and so on. Therefore, the maximum total number of clusters generated by BFT would be C= $\prod_1^p(\beta + 1)$, for the choice of p sequential passes. The value of p can be tuned according to the requirements of the application and the filtration threshold imposed by the user. BFT also benefits from a neighbor cluster joining mechanism to make sure that the clusters have roughly similar load of tuples. It is interesting to observe that BFT orders the frequency tuples on their MFB bits. Finally, the Algorithm 2 as the filtration step, is performed.

The intuition behind BFT is the fact that, given $f(S_i) \in c'_{k'}$ and $f(T_j) \in c_k$ and range r, if the absolute difference between their corresponding MFB entry is bigger than the given range: $(|k' - k| > r) \Rightarrow (\text{FD}(f(S_i), f(T_j)) > r) \Rightarrow (\text{ED}(S_i, T_j) > r)$, and hence all the clusters $c_k, \ldots, c_{|\Sigma|}$ $(T_j \ldots T_m)$ may be pruned from the candidate set. We could further represent each of the clusters with a single frequency vector or build a tree index on T′, to reduce the space and time needed to perform the approximate join operation. These issues would be further discussed in the simulation result section.

Algorithm 1 Approximate join processing:

Offline preprocessing phase, Given the string database, T(and S)$\in \Sigma^*$:

• (*Block-based partitioning*) Slide the blocking window of size b on the original DNA dataset T and extract the corresponding b-sized tuples, partitioning T on positions $0, \frac{b}{2}, \frac{2b}{2}, \ldots$ into a total of $\frac{|T|-b+1}{\lfloor \frac{b}{2} \rfloor}$ blocks. Let T_j denote the block/tuple extracted from T, at position j, where $0 \leq j < |T| - b$.
• Represent the dataset with its corresponding relational representation with its tuples being the extracted blocks, and index j as the primary key of the relation.
• Perform *Frequency Quantization*(Def. 2) on each tuple T_j, constructing T_j^F,
• Use the desired DFT or DWT transformation on each *Frequency-Quantized* tuple T_j^F, and calculate the corresponding transformed vector $X(T_j^F)$ or $\varpi(T_j^F)$ coefficients in the frequency domain, respectively.
• Extract and store only a few coefficients to represent the original string tuple. For the case of DFT, we keep the highest *energy-concentrated* coefficients as, first, last and the second[19]. For the DWT, we keep the first and second coefficients.
• For each relation T, build an offline index structure, relation T′[*index#*, *tuple vector(s)*].

Online filtration phase, Given a distance function $d(FD$ or $L_p)$, and range r:

for all tuple vectors in S′: **do**
  for all tuple vectors in T′: **do**
    if $d(X(S_i) - X(T_j)) > r$ **then**
      Prune T_j from the resulting candidate set;
      Break;
    end if
  end for
end for
▷ *Refinement step:* Apply dynamic programming on the remained tuple pairs, to find the strings S_i and T_j, where $ED(S_i, T_j) < r$.

Algorithm 2 Bit filtration procedure:

for i = 1 to n: **do**
  for j = 1 to m: **do**
    if $|(f_{MFB}(S_i) - f_{MFB}(T_j))| > r$ **then**
      // means that $ED(S_i, T_j) > r \Rightarrow$ *prune all remaining tuples* $T_j \ldots T_m$*;*
      Continue with S_{i+1};
    end if
  end for
end for

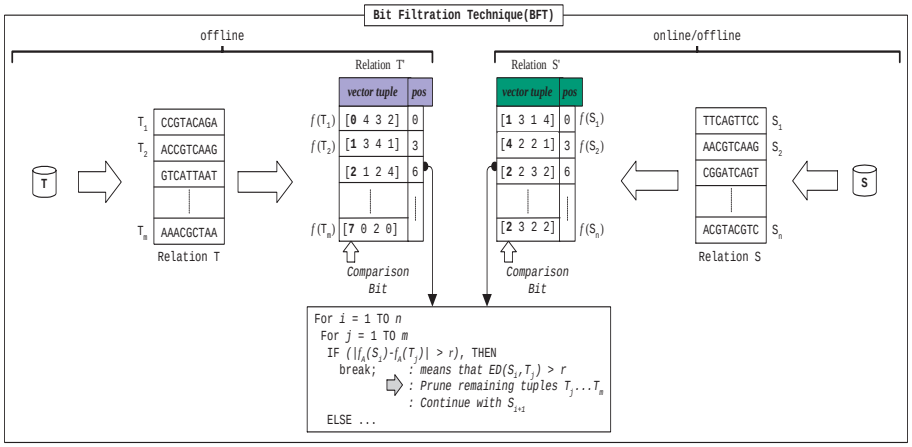

Fig. 3. The BFT filtration procedure, for blocking factor $b = 9$

4 Performance Analysis

4.1 Implementation

We compared the performance of BFT, DFT, and DWT as preprocessing filtration techniques, against and in conjunction with dynamic programming [15, 17], and q-gram [7] approaches. Our implementation closely follows the depicted procedures of Figures 1-3.

We incorporated different blocking methods for string dataset to relational database conversion procedure: *i*) *Consecutive* partitioning: Each of the consecutive blocks of length b, overlap by $b-1$ residues, *ii*) *Overlapped* partitioning: Each of the consecutive blocks of length b, overlap by $b/2$ residues(Fig. 1), and *iii*) *non-Overlapped* partitioning, where the whole data is chopped into $l = \theta(log_{|\Sigma|}|\mathrm{T}|)$ [14] partitions of various length. On various block partitioning methods, more the blocks extracted, we observed higher computational cost, and better filtration ratio(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 1.4.1*, and ran our simulations on an *Intel Xeon 2.4GHz* processor with *2GB* of main memory.

i) Frequency Distance (FD) should be a tight chosen in the frequency domain, *cost*. For instance, in the case of DFT, we had to use $1^{st}, 2^{nd}$, and n^{th} however, its corresponding running time became impractical After applying DWT, or DFT, we clustering the extracted frequency vectors as incorporated in BFT procedure, domain should be, computationally,

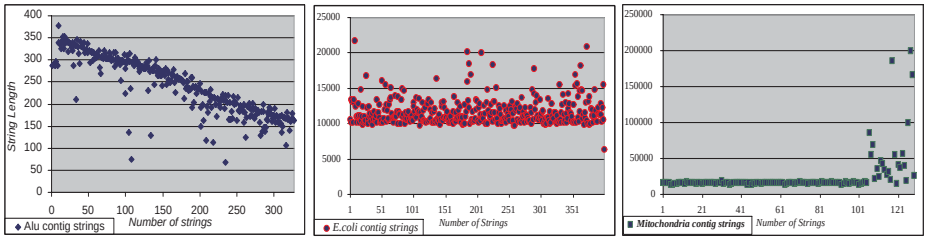

Fig. 4. Distribution of string lengths for *Alu*, *E.coli*, and *Mitochondria* datasets

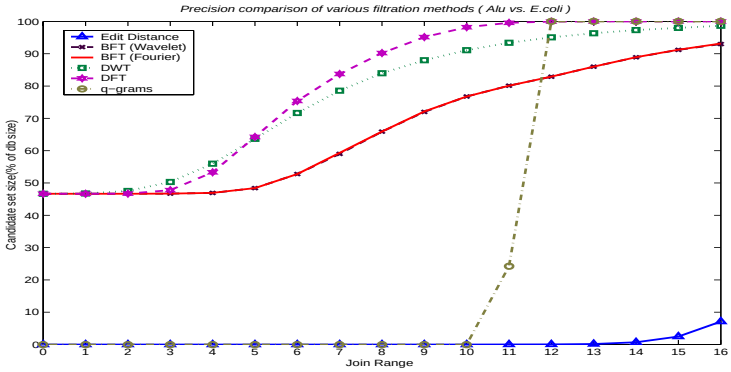

Fig. 5. Resulting candidate set of $Alu \bowtie E.coli$ for $b = 32$

as while maximized, incurring no *false negatives*, however the efficiency of pruning depends on the approximate join range(*application requirements*).

Table 1. The statistics ($max\ b = 32$) for the datasets used in our simulations

Dataset	A	C	G	T	Total	Tuple quantity	MFB
Alu	24301	18271	22192	15742	80506	4530	G
Mitocondria	1024379	647278	502392	989164	3163213	197566	A
E. coli	1148707	1184392	1181731	1147409	4662239	290779	A
imdb	N/A	N/A	N/A	N/A	788020	54000	B

4.2 Simulation Results

We ran our experiments on three Prokaryote and Eukaryote genome databases (*Alu*, *Escherichia coli*, and *Mitochondria*) [12], and one actor name database [8]. The Statistics of the incorporated data are depicted in

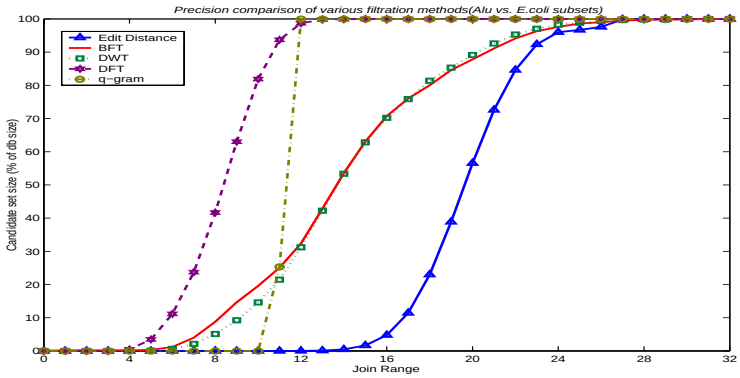

Fig. 6. Resulting candidate set of $Alu^R \bowtie E.coli^R$ for $b = 32$

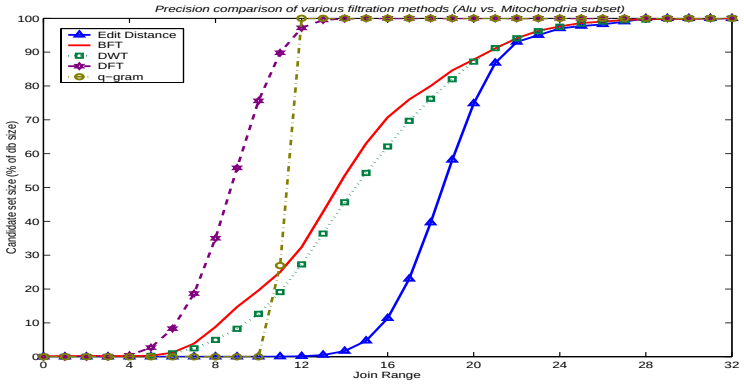

Fig. 7. Resulting candidate set of $Alu^R \bowtie Mitochondria^R$ for $b = 32$

Fig. 4, and Table 1. Due to the large computational cost of applying the *inner-loop-join* for the ED comparison, we had to reduce the size of each DNA database relation T into 4K tuples, named T^R, to be able to run the *all-pair-all local alignment*[17] in a reasonable amount of time. However, Fig. 5 demonstrates the filtration efficiency on the original files with 1.3 billion tuple comparisons. Initially, we performed block-based mapping on the DNA contig datasets to build their relational equivalents. In the blocking process, we applied a uniform tuple length($b = 32$) for all the DNA databases of the choice, however, tuple lengths in the movie database[8] were variable($8 \leq b \leq 32$) by nature. Additionally, we could incorporate variable block lengths on our DNA datasets but, we only included the result for the uniform blocking for the sake of simplicity. We incorporated *three* coefficients($1^{st}, 2^{nd}$, and n^{th}) for DFT, and *two* coefficients (1^{st} and 2^{nd}) for DWT. The results of BFT were based on only

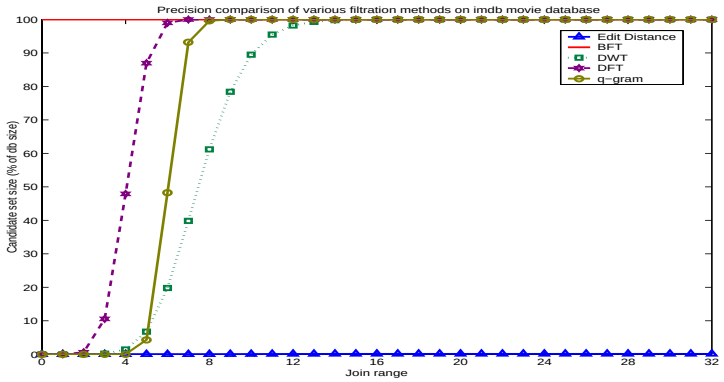

Fig. 8. Resulting candidate set of the approximate join *imdb*[8] database

using the *first* MFB, hence, only *1-pass* clustering of the relation. Due to the limitations of the space, the results of more than one MFB are not shown in this paper. Figures 5-8, demonstrate the filtration efficiency of running BFT, DWT, and DFT compared with q-gram[7], and *local alignment*[17] techniques on various databases.

Given two database relations S and T, let B denote the total number of tuple comparisons needed in an approximate join operation. Vertical axis demonstrates the candidate set($\propto \frac{1}{precision}$), the fraction of comparisons that is left for further refinement ($\frac{|f(S) \ \bowtie_d^r \ f(T)|}{B}\%$), as a function of join range. Smaller candidate set is the result of a higher filtration efficiency. Figures 5, and 6-8 demonstrate the filtration efficiency, out of 1.3 billion and 16 million total number of comparisons, respectively. In Fig. 5, we used two different ways of performing BFT: using the 1^{st} coefficient of DWT versus the 1^{st} coefficient DFT for frequency vector extraction. However, the results were identical, which can be explained by the fact that the first coefficient of DFT is identical to the first coefficient of DWT with a multiplication factor of $1/\sqrt{n}$. Therefore, for the rest of experiments, we only demonstrate the classical BFT using frequency vectors extraction. BFT, and DWT demonstrated very similar trend on the filtration ratio. Given relation S and target relation T, all the techniques except BFT, need to inspect all the tuples of the target relation T for any given tuple of S, while in contrary, BFT incorporates the pruning phase and hence, does not need to scan the entire database T. The portion of database inspected by BFT takes its best values on the lower ranges. Lower the join range, a better filtration ratio was expected and observed, while potentially a larger portion of the relation is expected to be out of range. BFT demonstrated up to 99.9% effective filtration ratio on all

DNA datasets. In very low ranges, q-gram[7] provides efficient filtration, however, it needs to scan all the target tuples for a possible *within-bound* q-gram count(or positional/length filtering).

Figure 8, demonstrates the result of running our proposed techniques(for $|\Sigma| = 32$), on two disjoint subsets of *imdb*[8] movie database. DWT achieves a reasonable filtration efficiency, however BFT works really bad! The reason lies on the fact that ***i)*** the chosen alphabet size was far too sparse and insensitive to the context, and hence MFB was not able to perform an efficient filtration on the tuples. We would need to use *word*s as a finer granularity representation of name tuples rather than just single alphabet characters. We are planning to investigate this issue more in our future work, ***ii)*** the actual portion of the database which was within the range(ED), was very small, and hence BFT had to scan the whole database to find the candidate matches. On the other hand, DFT and DWT perform efficient filtration for 0-5 range. Not surprisingly, this is the desired range within the area of data cleansing or data integration, while only very few typographical errors on each single word/block are allowed.

Table 2 shows the average running timing comparisons(in *seconds*) of approximate join for range $r = \{0, 1, \ldots, 8\}$. Figures 9-10, demonstrate the "*Crème de la Crème*" of applying BFT. We first applied BFT as a preprocessing filtration step to extract the candidate sets calculated from the frequency join $f(\mathrm{S}) \bowtie^r_{FD} f(\mathrm{T})$, incurring *no false negatives*, which is a superset of the actual result set $\mathrm{S} \bowtie^r_{ED} \mathrm{T}$. Furthermore, we used local alignment, and q-gram techniques on the remaining candidate set as the next pruning step to *possibly* narrow the search space either to the actual answer set(alignment), or *possibly* a narrower candidate set(q-gram). As a result, the recall[4] of applying BFT, DWT, DFT were all 100%, as earlier expected by Theorem 1(FD$\leq$ED), as *no false negatives* were created. Figures 9-10 show up to 52-times speed-up on the overall process, should BFT is used as a filtration preprocessing step.

Table 2. Average run time(in seconds) of approximate join, for $r = 0 \ldots 8$ and $b = 32$

	Edit Distance	**DFT**	**q**-gram	**DWT**	**BFT**
Alu $\bowtie^r$ *E.coli*	109546	409156	101872	93337	4881

5 Conclusion

In this paper, we proposed a novel, yet simple, *Bit Filtration Technique(BFT)* for more efficient filtration of undesired tuple comparisons, and studied

[4] $Recall = \frac{CandidateSet \cap AnswerSet}{AnswerSet}$

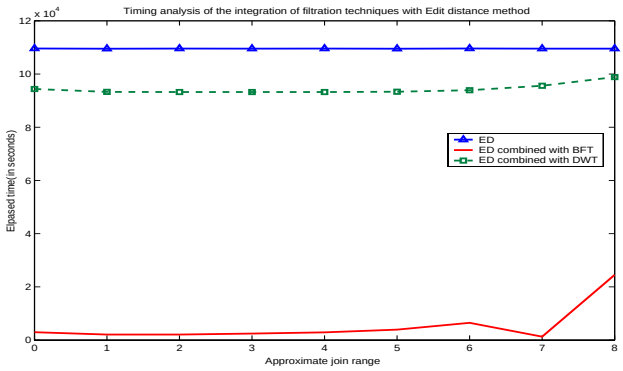

Fig. 9. Running time comparison of *Alu* ⋈ *E.coli* with the integration *BFT* on *Edit Distance*[17] approach as a function of approximate range, for $b = 32$

the similar integration of DFT and DWT on biological databases and evaluated the specific problem of approximate join. BFT and other proposed transformation methods, may be applied as a pre-processing filtration step for any of the known heuristic techniques like BLAST[1], QUASAR[3], FastA[16], and even the dynamic programming sequence alignment[17, 15], and q-gram[7]. Our results show that applying the proposed techniques, a high accuracy and faster database pruning is achieved. 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 on the integration of BFT, for up to 99.9% filtration, incurring no false negatives, and 50-times faster running time.

References

1. Altschul, S., Gish, W., Miller, W., Myers, E., Lipman, D.J.: Basic Local Alignment Search tool. Journal of Molecular Biology **215** (1990) 403–410
2. Apostolico, A.: The Myriad Virtues of Subword Trees. Combinatorial Algorithms on Words, NATO ISI Series, Springer-Verlag (1985) 85–96
3. Burkhardt, S., et al.: q-gram Based Database Searching Using a Suffix Array (QUASAR). RECOMB (1999) 77–83
4. Chavez, E., Navarro, G.: A Metric Index for Approximate String Matching. LATIN (2002) 181–195
5. Gaede, V., Günther, O.: Multidimensional Access Methods. ACM Computing Surveys **30** (1998) 170–231
6. Giladi, E., Walker, M.G., Wang, J.Z., Volkmuth, W.: SST: An Algorithm for Finding Near-Exact Sequence Matches in Time Proportional to the Logarithm of the Database Size. Bioinformatics **18** (2002) 873–877

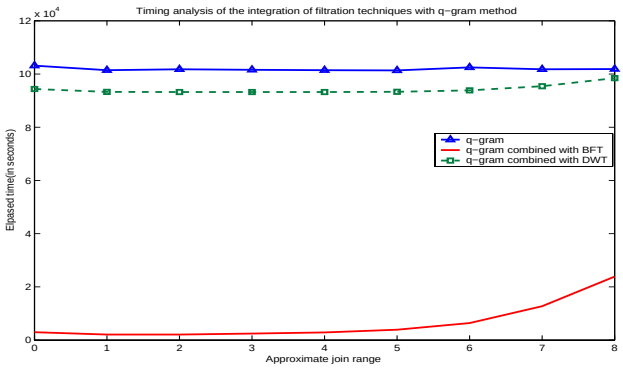

Fig. 10. Running time comparison of $Alu \bowtie E.coli$ with the integration BFT on q-gram[7] approach as a function of approximate range, for $b = 32$

7. Gravano, L., et al.: Approximate String Joins in a Database (Almost) for Free. VLDB (2001) 491–500
8. Internet Movie DataBase (IMDB). http://www.imdb.com
9. Jin, L., Li, C., Mehrotra, S.: Efficient Similarity String Joins in Large Data Sets. UCI ICS Technical Report, TR-DB-02-04 (2002)
10. Jokinen, P., Ukkonen, E.: Two Algorithms for Approximate String Matching in Static Texts. MFCS **16** (1991) 240–248
11. Kahveci, T., Singh, A.K.: Efficient Index Structures for String Databases. VLDB (2001) 351–360
12. National Center for Biotechnology Information(NCBI). http://www.ncbi.nih.gov/
13. Navarro, G., Baeza-Yates, R.A.: A Hybrid Indexing Method for Approximate String Matching. J. of Discrete Algorithms **1** (2000) 205–239
14. Navarro, G., Baeza-Yates, R.A., Sutinen, E., Tarhio, J.: Indexing Methods for Approximate String Matching. IEEE Data Engineering Bulletin **24** (2001) 19–27
15. Needleman, S.B., Wunsch, C.D.: General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. J. of Mol. Biol. **48** (1970) 443–453
16. Pearson, W.R.: Using the FASTA Program to Search Protein and DNA Sequence Databases. Methods Mol Biol **25** (1994) 365–389
17. Smith, R., Waterman, M.S.: Identification of Common Molecular Subsequences. J. Mol. Biol. **147** (1981) 195–197
18. Thompson, J.D., Higgins, D.G., Gibson, T.J.: CLUSTAL W: Improving the Sensitivity of Progressive Multiple Sequence Alignment Through Sequence Weighting, Position Specific Gap Penalties and Weight Matrix Choice. Nuc. Acids Res. **22** (1994) 4673–4680
19. Wu, Y., Agrawal, D., El Abbadi, A.: A Comparison of DFT and DWT based Similarity Search in Time-Series Databases. CIKM (2000) 488–495

Appendix:

Corollary 1 *Let* S= $s_1, \ldots, s_n$ *be a string over the alphabet* Σ_k *with the corresponding frequency vector* $f(\mathrm{S}) = [f_1, \ldots, f_k]$. *Any single edit distance operation at the position* i *of the string* S, *corresponds to an equivalent frequency distance operation:*

- *(Deletion of a character)* $\equiv$ *(*$f_i \leftarrow f_i - 1$*), for some* i,
- *(Insertion of a character)* $\equiv$ *(*$f_i \leftarrow f_i + 1$*), for some* i,
- *(Replacement of a character)* $\equiv$ *(*$f_i \leftarrow f_i - 1$ *and* $f_j \leftarrow f_j + 1$*),* $\exists\ i \neq j$

For instance, let S and T be two strings from Σ^*, with their corresponding frequency vectors $f(\mathrm{S}) = [2\ 3\ 0\ 4]$ and $f(\mathrm{T}) = [0\ 4\ 2\ 1]$. Then, $FD(f(\mathrm{S}), f(\mathrm{T})) = 5$ with $3(\pm 1)$ operations(*Replacements*) and $2(-1)$ operations(*Deletions*). It is interesting to observe that f :S$\rightarrow$ f(S) is a *many-to-1* function and $5 \geq \mathrm{ED(S,T)} < 9$.

What is the geometric explanation behind FD? Given two frequency distance vectors $u, v \in \mathbb{R}^{|\Sigma|}$, the Frequency Distance FD(u, v), as defined above, relocates the origin of the $\mathbb{R}^{|\Sigma|}$ vector space to the far end of the vector u, and returns the sum of the magnitudes of those portions of the vector v entries which are in the positive region of the new relocated coordinate space.

Definition 4 ***(Frequency Distance, FD)*** *Given two frequency vectors* U= $[u_1, \ldots, u_k]$ *and* V= $[v_1, \ldots, v_k]$, *The frequency distance* FD(U,V), *is defined as the minimum number of (+1), (-1), and (*$\pm$*1) operations(corollary 1) needed, on the entries of* U, *to transform* U *to* V, *or vice versa.*

Theorem 1 *Let* S *and* T, *be two strings from alphabet* Σ, *with their corresponding frequency vectors* f(S) *and* f(T). *The frequency distance* FD$(f(\mathrm{S}), f(\mathrm{T}))$, *is a lower bound on the edit distance* ED(S, T) *[11]:* FD$(f(\mathrm{S}), f(\mathrm{T})) \leq \mathrm{ED(S, T)}$. *Furthermore, given range* r, (FD$(f(\mathrm{S}), f(\mathrm{T}))$ $> r) \Rightarrow \mathrm{ED(S, T)} > r$.

Proof. As explained by corollary 1, frequency distance (FD) substitutes each of the edit operations by at most one $(+1)$, (-1), or (± 1) operation. Hence, the number of FD operations provide a lower-bound on the number of ED operations, hence FD$\leq$ ED. Similarly, it is easy to show that FD satisfies the criteria of being a metric, while the triangular inequality holds be default. □

Definition 5 *The k^{th}-level Haar Wavelet Transformation(DWT) [11] of a frequency quantized string 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 S = AGGTTGCAATTA , the 3^{rd}-level *Haar Wavelet* transformation of S is $\varpi_3(AGGTTGCAATTA)$= $\{\alpha_{3,0}, \beta_{3,0}\} = \{[4, 1, 3, 4], [-2, -1, 3, 0]\}$, represents the set of first and second wavelet coefficients.

Definition 6 *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 $(|\Sigma| \times 1)$-dimensional 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 $(|\Sigma| \times 1)$-dimensional vectors.

For instance, for $S' = ACCT$, the first and second DFT coefficients of S' are: $X_0(S') = [\frac{1}{2}, 1, 0, \frac{1}{2}]$ 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 that needs to be answered: *What would be the proper* FD *distance to deploy in the frequency domain to provide a better/tighter approximation on the Edit Distance of the original space, when using* DFT?

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. 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. Higher the FD, constrained that FD $\leq$ ED, a better estimate on ED is achieved. For any two *DFT-transformed* vectors X, Y: we deploy L_1-norm as the desired frequency distance which is defined as $L_1(X, Y) = \sum_j |x_j - y_j|$. L_1-norm facilitates a tighter lower bound and is also expected to have a lower computational cost compared with the other L_p-norms.