EFFICIENT FILTRATION OF SEQUENCE HOMOLOGY SEARCH THROUGH SINGULAR VALUE DECOMPOSITION

S. ALIREZA AGHILI, ÖZGÜR D. ŞAHİN,
DIVYAKANT AGRAWAL, AMR EL ABBADI
Department of Computer Science,
University of California-Santa Barbara,
Santa Barbara, CA 93106
{*aghili,odsahin,agrawal,amr*}*@cs.ucsb.edu*

Similarity search in textual databases and bioinformatics has 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 whole-genome sequence homology search into an approximate vector comparison in the well-established multidimensional vector space. We propose the application of Singular Value Decomposition(SVD) dimensionality reduction technique as a pre-processing filtration step to effectively reduce the search space and the running time of the search operation. Our empirical results on a Prokaryote and a Eukaryote DNA contig dataset, demonstrate effective filtration to prune non-relevant portions of the database with up to 2.3 times faster running time compared with q-gram approach. SVD filtration may easily be integrated as a pre-processing step for any of the well-known sequence search heuristics as BLAST, QUASAR and FastA. We analyze the precision of applying SVD filtration as a transformation-based dimensionality reduction technique, and finally discuss the imposed trade-offs.

1 Introduction

The problem of similarity search and the corresponding applications have been extensively studied within the past decade, especially in the context of textual and biological databases. 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*). The approximate k-Nearest Neighbor(kNN) seeks the k-closest strings of the database to the given query string using an appropriate distance function. Approximate sequence analysis has enabled the detection of certain strains of the *Escherichia coli(E.coli)* bacteria responsible for infant *diarrhea* and *gastroenteritis*. Meanwhile, looking for pairwise whole-genome homology search most of the 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. Similarly, the problem of keyword search in textual and web data mining has been well-studied within the past

years. Mathematical techniques such as Singular Value Decomposition(SVD) have been used to replace the massive term-by-document matrices with much smaller matrices and perform the approximate search in the reduced matrix space. Using such techniques, unimportant words get discarded and the search process is only singled out to highly relevant words. In this paper, we consider the integration of a textual data mining technique as an efficient filtration on genomic data to leverage the cost and scalability of the approximate search process. We map the problem of pairwise whole-genome sequence comparison into an approximate vector comparison in the well-established relational database context. We propose the integration of *Singular Value Decomposition(SVD)* as a pre-processing filtration step towards whole-genome k-Nearest Neighbor(kNN) search. Our simulations study the corresponding filtration efficiency gained by the proposed technique on a Prokaryote and a Eukaryote DNA contig dataset.

The rest of the paper is organized as follows: Section 2, discusses the background and related work. Section 3 introduces the terminology and formulation of the problem, followed by the proposed technique in section 4. Section 5 demonstrates a concise empirical performance analysis and the simulation results followed by section 6, which concludes the work.

2 Background, Related Work

In a typical application of kNN, given a string dataset S and a query string q, all the string tuples of S are compared against q, in search for the k-closest substring tuples S_i to query q. However, because of the quadratic time involved, the dynamic programming[16,18] algorithms are not feasible. Several heuristics[3,5,6,12,17] have been proposed to speed up the similarity search phase of the procedure in the case of range query and k-nearest neighbor search. To the best of our knowledge, this study is the first effort to facilitate efficient filtration for genome-wide approximate sequence search using *Singular Value Decomposition(SVD)*.

Jin, Li, and Mehrotra[10] map the strings of the database into the 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 may potentially result in a large number of *false negative*s. Gravano et al.[9] target the problem of approximate join in textual relational databases. They extract positional q-grams[9,11,15] 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)[12] 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, the authors[12] only focus on the cost of MRS, and do not evaluate the filtration efficiency of their proposed technique.

Chavez and Navarro[6] 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[4] as an index structure. No empirical analysis is conducted to evaluate this approach on real biological data. SST[8] uses overlapping sliding windows of size w over the database sequences and maps them into 4^w-dimensional frequency vectors. Furthermore, SST uses k-means clustering algorithm to hierarchically cluster database sequences. It first divides the database sequences into non-overlapping windows. Given a query P, it prunes the database windows which are further from the given query range. The authors[8] finally study the effect of window size on search time, error rate, and true positive/negative rates of the proposed technique. Most similarly, Aghili et al.[1,2] provide a concise study of *Discrete Fourier Transformation(DFT)*, *Discrete Wavelet Transformation(DWT)* and *Bit-Filtration Technique(BFT)* as pre-processing filtration techniques for approximate join and range queries. In this work, we propose a new pre-processing filtration technique to the problem of k-Nearest Neighbor(kNN) search in the context of biological databases, and study the imposed trade-offs.

3 Terminology, Formulation

The traditional database search operation is based on the *exact matching* of string tuples, while the approximate search is based on the *approximate matching* of the string tuples. Initially, we build a relational database view on top of the raw DNA contig datasets[13] by chopping its contig sequences into equal-sized subsequences(blocks). This procedure is called *block-based mapping*, which facilitates the initial step of mapping the problem of whole-genome sequence homology search into an approximate string search in the context of relational databases(Fig. 1).

The following definitions introduce the steps in transforming the *original domain(set of strings)* to *frequency domain(set of feature vectors)*:

Definition 1 ***(k-Nearest Neighbor, kNN)*** *Let $S = S_1, \ldots, S_N$ be a string*

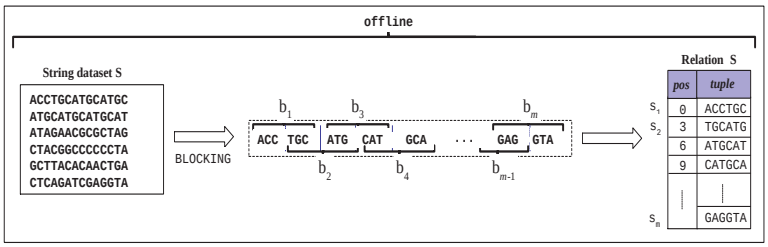

Figure 1: The block-based mapping procedure for *block size* = 6.

database over the alphabet Σ. *Let* $S_{i,j}$ *denote a subsequence of* S_i *starting at index* j, *for* $1 \leq i \leq N$ *and* $0 \leq j < |S_i|$, *where* $|S_{i,j}| = |S_i| - j$. *Given a query pattern* $P \in \Sigma^*$, *an integer* k, *and distance function* d, *the* $kNN_q^d(S, P)$ *is the problem of finding the* k *closest subsequences* $S_{i,j}$ *to the query pattern* P.

Definition 2 ***(frequency vector)*** *Let* s *be a string over the alphabet* $\Sigma_u = \{\alpha_1, \ldots, \alpha_u\}$, *then the q-tuple frequency vector of* s *is a row vector* $f_q(s)$ *and defined as:* $f_q(s) = [f_1, \ldots, f_{u^q}]$, *where each* f_i *is a positive integer corresponding to the occurrence frequency of* i^{th} q*-sized substring from* Σ_u *in* s.

For instance, for Σ={A,C,G,T}, q=2, and string s = $AGGTTGCAATTA$: $f_2(s)$=[1 0 1 1 1 0 0 0 0 1 1 1 1 0 1 2] where the first entry is used to represent the frequency of AA, second entry for AC, third entry for AG, ..., and the last entry to represent the frequency of TT in s.

Definition 3 ***(frequency quantization)*** *Let* $S = \{S_1, \ldots, S_n\}$ *be a string relation from the alphabet* Σ_u. *The q-tuple frequency quantization of relation* S, $S_q^F = [\xi_1, \ldots, \xi_{u^q}]$, *is an* $(n \times u^q)$ *matrix, where each vector* ξ_j *corresponds to the* $(n \times 1)$*-dimensional column vector for* j^{th} q*-tuple of all* S_i *strings, for* $1 \leq j \leq u^q$.

In other words, Definition 3 is equivalent to extracting the frequency vectors for each of the tuples of the given string relation S and placing them as rows in a new matrix S^F. Hence, for

$$S = \begin{bmatrix} A & G & G & T & T & G & C & A & A & T & T & A \\ C & C & G & T & A & A & T & T & A & G & G & C \\ T & G & T & G & C & C & C & A & G & G & A & C \end{bmatrix},$$

applying frequency quantization(Def. 3) results in:

$$S_2^F = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 2 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 2 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 2 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 2 & 0 \end{bmatrix}.$$

Definition 4 ***(Singular Value Decomposition, SVD)*** *SVD decomposes a matrix* $A \in R^{m \times n}$ *into the product of two orthonormal matrices,* $U \in R^{m \times m}$,

$V \in R^{n \times n}$, and a pseudo-diagonal matrix $\Sigma = diag(\sigma_1, \ldots, \sigma_r) \in R^{m \times n}$ where $r \leq min(m,n)$ (all the components except the first r diagonal components are zero), such that $A = U\Sigma V^T$. By convention, the diagonal elements σ_i of Σ, named singular values, are all non-negative and are sorted in decreasing order.

One way to solve the kNN problem is as follows: Given relation S, compare all tuples of S against the query pattern P using *Edit Distance(ED)*[a] as the distance measure, either through direct application of dynamic programming [16,18] or other popular heuristics[3,5,17]. 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 [13] 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 or inefficient, respectively. A solution could involve mapping the approximate string similarity search of the sequence domain into a vector comparison in a vector domain using a well-defined *Frequency Distance (FD)*, 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* $f_q(S_i)$, for a given tuple size q, and use an appropriate frequency distance function to approximate the edit distance of the original string domain[1,2,6,8,9,10,12,14].

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, for the price of generating some *false positives*. Hence, approximate string search is potentially much more efficiently evaluated incorporating the frequency domain properties. These claims are further validated in the simulation study section.

4 Proposed *SVD-Filtration* Technique

Given the string dataset S with a total of B blocks, we first extract its blocks and construct the relational equivalent of it as S= $\{S_1, \ldots, S_B\}$(Fig. 1). The tuple extraction procedure is very fast and needs only a single scan of the given database. In the second step, the corresponding frequency vector(s), $f_q(S_i)$, are extracted into the resulting relation $S_q^F = \{f_q(S_1), \ldots, f_q(S_n)\}$(Fig. 2). The schema of S_q^F has two attributes: *i)* The *index* of the tuple in the original database as the primary key, and *ii)* $|\Sigma|^q$-dimensional *frequency feature vector*.

The transformation-based *SVD* filtration is an automated procedure except for the choice of required precision or in other words the desired amount

[a] Minimum number of Insertion, Deletion, and Replacement operations needed to convert a string into another string.

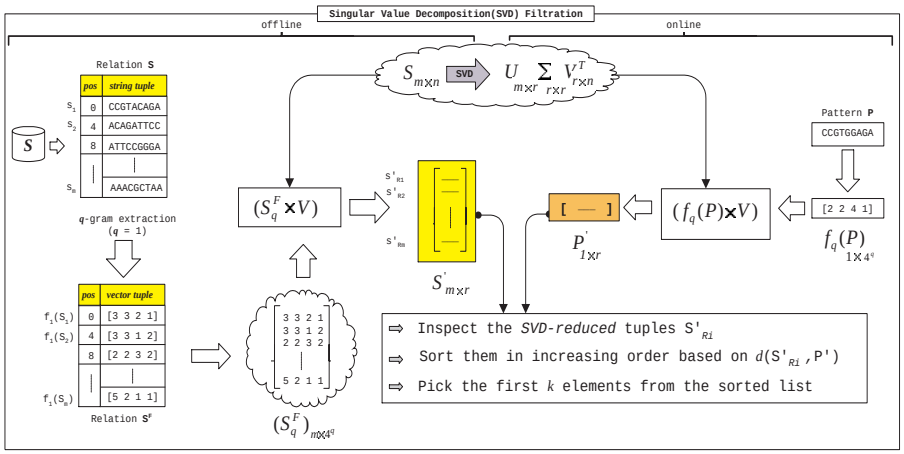

Figure 2: The SVD filtration procedure for *block size* = 9, and $q = 1$.

of reduction. The general process is depicted in Figures 2-3. Given the genome dataset S and query pattern P, the kNN procedure is performed in two different stages: *offline* and *online*. In the offline stage, all the blocks/tuples S_i are extracted, and each tuple is mapped onto its corresponding frequency feature vector(Def. 2). As a result, S is mapped into S_q^F frequency vector matrix, which is a row matrix of frequency feature vectors $f_q(S_i)$. Each $f_q(S_i)$ keeps the quantity of each q-sized substring(q-gram) of the original string S_i. Furthermore, SVD is applied on S_q^F, decomposing it into U, Σ, and V matrices. Finally, the reduced version of the frequency vector matrix S_q^F is calculated as $S' = S_q^F \times V$. The actual kNN operation is performed in the online stage. Initially, the reduced version of P is calculated: P$\rightarrow f_q(P) \rightarrow P' = f_q(P) \times V$. All the reduced vector tuples of the relation S$'$ are compared against P$'$ and the k-closest tuples of S to P are reported. All the remaining tuples are pruned from further investigation.

The q-gram technique[9,11] uses the original S_q^F matrix. However, each of the frequency vectors, $f_q(S_i)$, is of size $|\Sigma|^q$ which grows exponentially with the value of q. For instance, for $|\Sigma| = 4$ and $q = 3$: each of the frequency vectors is 64-dimensional. Note that, SVD reduces the dimension of the vector representation of S from B$\times 4^q$(of S_q^F) to B$\times r$(of S'), where r is the rank of S_q^F matrix. This is the main intuition behind applying SVD, to remove the *"curse of dimensionality"*, while *i*) $r \ll 4^q$, and *ii*) SVD captures the shape of the vector reasonably well.

After applying either SVD or q-gram filtration techniques to extract the candidate set, we perform dynamic programming[18] on the candidate set as

Offline pre-processing phase, Given the string database S$\in \Sigma^*$, and integer q:

1. (*Half-overlapped block-based partitioning*) Slide the blocking window of size b on the original DNA dataset S and extract the corresponding b-sized tuples, partitioning S on positions $0, \frac{b}{2}, \frac{2b}{2}, \ldots$ into a total of $B = \lceil \frac{|S|-b+1}{\lfloor \frac{b}{2} \rfloor} \rceil$ blocks. Let S_j denote the block/tuple extracted from S, at position j, where $0 \leq j < |S| - b$,
2. Represent the dataset S with its corresponding relational representation, its tuples being the extracted blocks, and index j as the primary key of the relation,
3. Perform the *frequency quantization* on S, constructing S_q^F as a $B \times |\Sigma|^q$ matrix,
4. Apply SVD transformation on the matrix S_q^F, and calculate the corresponding U, Σ, and V matrices,
5. Calculate the product $S' = S_q^F \times V$ (S$'$ is the *SVD-reduced* version of S),
6. Build an offline index structure on S$'$[*index#*, *reduced tuple vector*].

Online filtration phase, Given query pattern P, distance function d(*cosine* or L_p), and integers k, q:

1. Calculate the frequency vector of P as $f_q(P)$,
2. Calculate P$' = f_q$(P) $\times V$ (P$'$ is the *SVD-reduced* version of P),
3. Allocate a k'-sized table τ, for $(k' > k)$,
4. Insert first k' vectors of S$'$ into τ in increasing order of their distance to P$'$,
   for i $= k' + 1$ to n: **do**
   if $d(S_i$,P$') < d(S_i, \tau[k'])$ **then**
   - Remove $\tau[k']$,
   - Insert S$_i$ onto its proper location in the sorted table τ.

$\triangleright$ *Refinement step*: Apply dynamic programming on the remained k' tuples in τ, and output the k-closest tuples to the query pattern.

Figure 3: Approximate k-Nearest Neighbor(kNN) process.

the final stage to get the actual answer set, removing the false positives. Additionally, a multidimensional indexing structure[7] could be incorporated on the extracted frequency vectors for more efficient tuple pruning. However, we intend to study the impact of the various indexing schemes in our future work.

The main idea of applying SVD filtration is as follows: Instead of calculating the approximate $k\text{NN}_q^{ED}(S, P)$, we may inspect the frequency vector domain for kNN. However, this may result in *false negatives*. Hence, we would need to look for a potentially, slightly larger value $k'(\geq k)$ in the vector space($k'\text{NN}_q^{FD}(S', P')$) to capture all the actual kNN. All the strings $S_i \notin k'\text{NN}_q^{FD}(S', P')$ may be pruned from the answer set without the need to further calculate the costly dynamic programming operation. This property dramatically reduces the *computational cost* and the *search space* of calculating approximate $k\text{NN}_q^{ED}(S, P)$. The total search space would be reduced to $\rho = \frac{k'}{B}$ (where $0 < \rho \ll 1$). We empirically observed that $k'\text{NN}_q^{FD}(S', P')$ contains most of the elements of $k\text{NN}_q^{ED}(S, P)$, specially when there are reasonably enough number of *close-match* sequences in the database to the query.

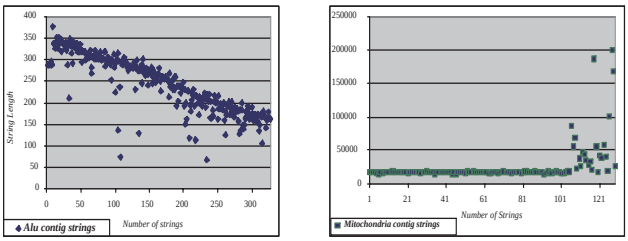

Figure 4: Distribution of string(*contig*) lengths for *Alu*, and *Mitochondria* datasets.

Table 1: The datasets used in our simulations for *block size* = 32.

Dataset	**A**	**C**	**G**	**T**	**Size**	**# of Blcoks**
Alu	24301	18271	22192	15742	80506	4555
Mitochondria	1024379	647278	502392	989164	3163213	197580

5 Performance Analysis

We compared the performance of SVD[b] as a pre-processing filtration technique against q-gram [9,11] filtration and used dynamic programming [18] as the benchmark. Our implementation closely follows the depicted procedures of Figures 1-3. Within the context of vector comparison in multidimensional indexing, *cosine similarity* distance measure is one of the most popular choices for frequency distance(FD) or vector similarity function. We should apply the frequency distance function which demonstrates a better estimate of the ED in the original domain, or in other words "*more precisely*" reflecting the similarity/distance across the sequences. The better the choice of FD, the more effective filtration is expected. For any two *SVD-transformed* vectors U, V: we deploy the *cosine similarity function* among vectors which is defined as $cos\theta(U,V) = \frac{U \cdot V}{||U|| \cdot ||V||}$.

We incorporated various blocking schemes on string datasets for the relational database conversion procedure: i) *Consecutive* partitioning: Each of the consecutive blocks of length b, overlap by $b-1$ residues, ii) *Half-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 $l = \theta(log_{|\Sigma|}|S|)$ [15] partitions of various length. Extracting more blocks during the block partitioning phase, e.g. case (i) versus (ii), results in more precise string search but higher computational cost. However, due to space limitation, we did not include those results in this study.

[b] The frequency vectors were reduced to 5 dimensions using SVD.

5.1 Implementation and Simulation Results

We implemented all the desired algorithms using *Java 1.4.2*, and ran our simulations on an *Intel Xeon 2.4GHz* processor with *2GB* of main memory. The experimental analysis was performed on a Prokaryote and a Eukaryote genome dataset (*Alu*, and *Mitochondria*) [13]. The statistics of the incorporated data are depicted in Fig. 4, and Table 1. In the blocking process, we applied *half-overlapped partitioning* with *block size*= 3 and $q = 3$. Additionally, we could use variable block lengths on our DNA datasets however, we only included the result for the uniform blocking for the sake of simplicity.

Identification of highly conserved sequences, which are likely to correspond to essential sites for the function or the structure of the sequence, is one of the main goals of approximate sequence search in bioinformatics. High similarity among two sequences (very few edit operations) may imply similar functional relationships or interactions, mutual inclusion in the same biological pathway, or may be used to infer evolutionary relationships. Consider the case where the query pattern does not have *close-match* counterparts in the desired database. In this case, the kNN will return a set of sequences, which are closest to the query pattern compared with other sequences of the database. However, if the distance among the query pattern and the returned nearest neighbors is high, then the returned results of the nearest neighbor search would be of no practical use. Moreover, the cosine distance among two SVD-reduced vectors may only reflect the actual similarity of the given sequences, if the sequences are *close-match* counterparts. This problem arises when there are no real *close-match* sequences in the database to the given pattern. Such queries are not very desirable since they will not provide any meaningful implication. A more desirable case is to look for similar patterns to a given query in the database which actually contains some close variations of the query pattern, and study how well the proposed techniques would perform in capturing those occurrences. For this reason, we performed an exhaustive search on both datasets and for each of them we picked a query pattern of length 32, for which the dataset actually contained some *close-match* blocks. More precisely, the query for each of the *Alu* and *Mitochondria* datasets had its actual 20 nearest neighbors within 2 and 4 edit distance, respectively.

Figures 5-6 demonstrate the filtration efficiency of applying SVD filtration compared with q-gram[9,11], and *dynamic programming*[18] techniques on the above datasets for up to 200K total number of block comparisons on each dataset. The results are summarized in the following three sections:

• **True Positive Rate(TPR):** Figure 5 depicts the true positive rate or the actual kNN returned by SVD and q-gram filtration against dynamic

programming method on *Alu* and *Mitochondria* datasets as a function of k. The vertical axis shows the number of returned correct results with k on the horizontal axis. The values returned by *dynamic programming* are the actual correct results and used as a benchmark. A large portion of the correct results were returned by simply trying kNN on the reduced vectors produced by SVD, which might be sufficient enough for approximate search, though with much faster response time. The results of SVD on *Mitochondria* dataset was very promising and close to q-gram method. On the average among the inspected 5 values of k, the percentage of the correct kNN results captured by using only SVD filtration(for $k' = k$) is 74% and 57.6% on the *Mitochondria* and *Alu* datasets, respectively.

• **Filtration Ratio:** In Figure 6, the vertical axis demonstrates the size of the minimum subset produced by SVD and q-gram, that includes the actual k-closest tuples to the given query pattern which is found through dynamic programming. The vertical axis corresponds to the optimal value for k' for the given value of k on the horizontal axis. For instance, looking for 8 nearest neighbors($k = 8$) on the *Mitochondria* dataset, we need to inspect at least 20 nearest neighbors($k' = 20$) in the SVD domain to capture all the actual 8 nearest neighbors. Hence, the application of SVD efficiently reduces the search space from B=197580 blocks to $k' = 20$ blocks, whereafter dynamic programming may be applied to eliminate the false positives. In general, a higher filtration efficiency is achieved when k and k' are closer to each other. Similarly, for low values of $k(< 10)$ on *Alu*, the actual nearest neighbors are captured by just trying the SVD method for k'=k. This means no false positives and no false negatives for $k < 10$ on *Alu*. Meanwhile, the results get worse with increasing values of k, however the filtration ratio is still considerably high. For instance, for $k = 20$, the filtration ratio is equal to $FR = \frac{k'}{B} = \frac{200}{197580} = 0.001$, meaning that more than 99% of the database may effectively be pruned. These graphs may be stored(*offline*) as a profile for each of the given datasets to assist in estimating(*online*) a good value for k'. We also note that in comparison with q-gram, the filtration of SVD is quite good. In the *Alu* dataset the size of the minimum subset is identical up to $k = 8$, and in the *Mitochondria* dataset although the size of the minimum subset is bigger in SVD, it is still comparable to q-gram for up to $k = 14$.

• **Running Time:** Table 2 shows the average running time of approximate kNN for $k = 20$. SVD is slower than q-gram on the computation side, which is because of the floating-point matrix multiplications performed by SVD for query reduction and angle computations. However, when taking the I/O response time into account as well, SVD performs up to 2.3 times faster than q-gram. This property is very desirable in the case of *online* or *interactive*

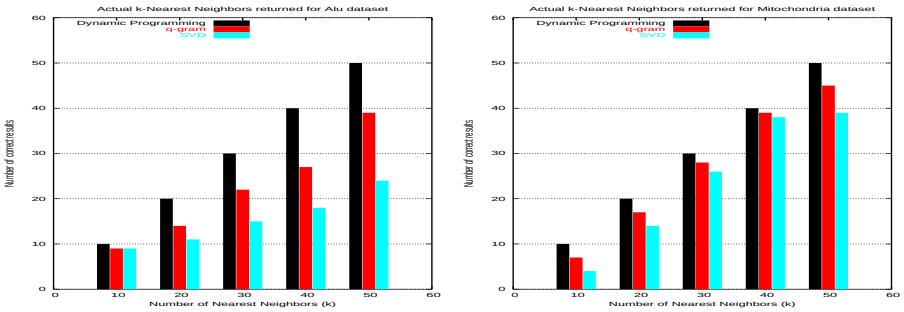

Figure 5: True Positives for kNN on *Alu* and *Mitochondria* datasets.

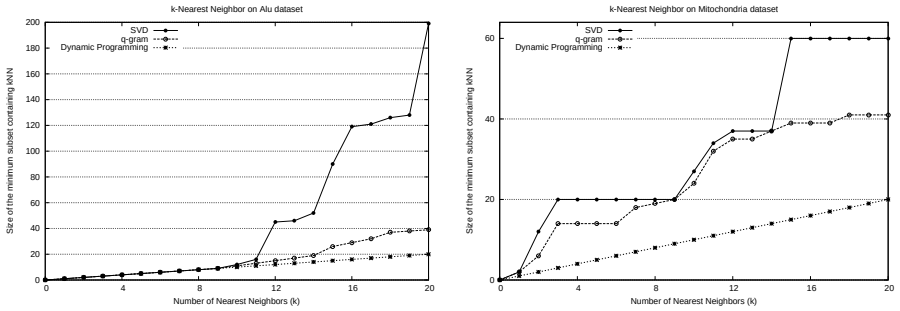

Figure 6: Filtration of approximate kNN on *Alu* and *Mitochondria* datasets.

approximate query search where the reduced vectors can be read much faster from the disk and leads to much less network traffic, resulting in faster overall query response time. q-gram is slower than dynamic programming when taking the I/O into account, because reading the original $|\Sigma|^q$-dimensional frequency vectors used by q-gram takes more time compared with reading string blocks of length 32($block\ size < |\Sigma|^q$), which dominates the computation time. The I/O portion may be further reduced by incorporating a caching technique or loading all the vectors into the main memory.

Table 2: Average running time(in *milliseconds*) of approximate kNN, for k=20.

	Computation		Computation + I/O	
	Alu	*Mitochondria*	*Alu*	*Mitochondria*
Dynamic Programming(DP)	300	11693	1248	30315
q-**gram**(*with DP refinement*)	7	117	1434	36210
SVD(*with DP refinement*)	24	363	984	15239

6 Conclusion

In this paper, we proposed a novel, yet simple, filtration technique for the genome-wide homology search using *Singular Value Decomposition(SVD)* to eliminate undesired tuple comparisons. We studied its integration on biological databases for the problem of approximate k-Nearest Neighbor search. SVD may be applied as a pre-processing filtration step for any of the known heuristic techniques like BLAST[3], QUASAR[5], FastA[17], and even the dynamic programming sequence alignment[16,18]. Our results show that applying the proposed technique, an efficient and fast filtration is achieved when the database contains *close-match* patterns to the given 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 of SVD filtration. In particular, SVD's filtration is comparable to q-gram[9,11] in most interesting cases of comparison, while being much faster in terms of I/O, e.g. we have shown that on different datasets, up to 2.3 times faster running time can be achieved.

Acknowledgments

This research was supported by the NSF grants under EIA02-05675, EIA99-86057, EIA00-80134, and IIS02-09112.

References

1. S.A. Aghili, D. Agrawal and A. El Abbadi, Filtration of String Proximity Search via Transformation. *BIBE* , 149-157 (2003).
2. S.A. Aghili, D. Agrawal and A. El Abbadi, BFT: Bit Filtration Technique for Approximate String Join in Biological Databases. *SPIRE* , (2003).
3. S. Altschul *et al*, *J. Molecular Biology* **215**, 403–410 (1990).
4. A. Apostolico, *Combinatorial Algorithms on Words, NATO ISI Series, Springer-Verlag* , 85–96 (1985).
5. S. Burkhardt *et al*, *RECOMB* , 77–83 (1999).
6. E. Chavez and G. Navarro, *LATIN* , 181–195 (2002).
7. V. Gaede and O. Günther, *ACM Computing Surveys* **30**, 170–231 (1998).
8. E. Giladi *et al*, *Bioinformatics* **18**, 873–877 (2002).
9. L. Gravano *et al*, *VLDB* , 491–500 (2001).
10. L. Jin *et al*, *UCI ICS Technical Report, TR-DB-02-04* (2002).
11. P. Jokinen and E. Ukkonen, Two Algorithms for Approximate String Matching in Static Texts. *MFCS* **16**, 240–248 (1991).
12. T. Kahveci and A.K. Singh, *VLDB* , 351–360 (2001).
13. National Center for Bio. Information(NCBI), *http://www.ncbi.nih.gov/.*
14. G. Navarro *et al*, *J. Discrete Algorithms* **1**, 205–239 (2000).
15. G. Navarro *et al*, *IEEE Data Engineering Bulletin* **24**, 19–27 (2001).
16. S.B. Needleman *et al*, *J. Molecular Biology* **48**, 443–453 (1970).
17. W.R. Pearson, *Methods Molecular Biology* **25**, 365–389 (1994).
18. T.F. Smith *et al*, Identification of Common Molecular Subsequences. *J. Molecular Biology* **147**, 195–197 (1981).