Journal of Clinical Monitoring and Computing (2005) 19: 319–328
DOI: 10.1007/s10877-005-0677-y

HIGH-PERFORMANCE EXACT ALGORITHMS FOR MOTIF SEARCH

Sanguthevar Rajasekaran,[1] *Sudha Balla,*[1] *Chun-Hsi Huang,*[1] *Vishal Thapar,*[1] *Michael Gryk,*[2] *Mark Maciejewski,*[2] *and Martin Schiller*[2]

Rajasekaran S, Balla S, Huang C-H, Thapar V, Gryk M, Maciejewski M, Schiller M. High-performance exact algorithms for motif search
J Clin Monit Comput 2005; 19: 319–328

ABSTRACT. Objective. The human genome project has resulted in the generation of voluminous biological data. Novel computational techniques are called for to extract useful information from this data. One such technique is that of finding patterns that are repeated over many sequences (and possibly over many species). In this paper we study the problem of identifying meaningful patterns (i.e., motifs) from biological data, the motif search problem. **Methods**. The general version of the motif search problem is NP-hard. Numerous algorithms have been proposed in the literature to solve this problem. Many of these algorithms fall under the category of heuristics. We concentrate on exact algorithms in this paper. In particular, we concentrate on two different versions of the motif search problem and offer exact algorithms for them. **Results**. In this paper we present algorithms for two versions of the motif search problem. All of our algorithms are elegant and use only such simple data structures as arrays. For the first version of the problem described as Problem 1 in the paper, we present a simple sorting based algorithm, *SMS* (Simple Motif Search). This algorithm has been coded and experimental results have been obtained. For the second version of the problem (described in the paper as Problem 2), we present two different algorithms – a deterministic algorithm (called *DMS*) and a randomized algorithm (Monte Carlo algorithm). We also show how these algorithms can be parallelized. **Conclusions**. All the algorithms proposed in this paper are improvements over existing algorithms for these versions of motif search in biological sequence data. The algorithms presented have the potential of performing well in practice.

KEY WORDS. motif search, algorithm, short sequence motifs.

1. PROBLEM

The human genome project has resulted in the generation of voluminous biological data. Novel computational techniques are called for to extract useful information from this data. One such technique is that of finding patterns that are repeated over many sequences (and possibly over many species). Several variants of this motif search problem have been identified in the literature (see e.g., [3, 5, 12, 13]). In this section we define two versions of motif search and list some related publications.

1.1. Problem 1

A pattern is a string of symbols (also called residues) and ?'s. A "?" refers to a wild card character. A pattern cannot begin or end with ?. AB?D, EB??DS?R, etc. are examples of patterns. The length of a pattern is the number of characters in it (including the wildcard characters). This problem takes

From the [1]Department of CSE, University of Connecticut, Storrs, CT 06269, USA; [2]Department of Neuroscience, University of Connecticut, Farmington, CT 06030, USA.

Received and accepted for publication June 30, 2005.

Address correspondence to Sanguthevar Rajasekaran, Professor of Computer Science and Engineering, University of Connecticut, 371, Fairfield Road, #2155, Storrs, CT – 06269-2155, U.S.A. E-mail: rajasek@engr.uconn.edu

as input a database *DB* of sequences. The goal is to identify all the patterns of length at most P (with anywhere from 0 to $\lfloor P/2 \rfloor$ wild card characters). In particular, the output should be all the patterns together with a count of how many times each pattern occurs. Optionally a threshold value for the number of occurrences could be supplied.

The above motif model has been derived as follows. We have generated a list of 312 minimotifs (i.e., motifs of short length) that have defined biological functions. We have used this list to select parameters for a de novo analysis of novel minimotifs in the human proteome. In this paper we choose to analyze novel motifs with a length (P) of 10 amino acids because 92% of the previously characterized minimotifs in our list are less than 10 amino acids in length. Another reason for choosing a length of 10 amino acids is based on the function of minimotifs. Most minimotifs are in binding domains or substrates of enzymes. The peptide ligand binding surfaces on proteins in the Protein Data Bank is usually no longer than 35 angstroms. A 10 amino acid peptide would achieve a maximum of 35 angstroms in length if it were in a random coil or a beta-sheet structure. Thus, the selection of a length of 10 amino acids is consistent with the length of peptides that interact with binding surfaces on protein domains.

The average minimotif in our list has 2.1 wildcard positions for any amino acid. Wildcards signify any of the 20 amino acids. Since only 13% of minimotifs in our list have more than 50% wild card positions, we chose $\lfloor P/2 \rfloor$ or 5 wild cards as the maximal number in the algorithm.

The second version of motif search considered in this paper is defined next as Problem 2. We provide sequential and parallel algorithms, analyses, and experimental results for Problem 1. For Problem 2 we provide sequential and parallel algorithms and analyses. Our algorithms for Problem 1 as well as Problem 2 employ radix sorting as the basic technique. We believe that similar techniques can be used for solving other variations of the motif search problem as well. For example, we have solved a variant called the planted motif search problem using similar techniques [12].

In the rest of the paper the word "occurrence" is used to denote occurrence within an edit distance of D, and the word "presence" is used to denote exact occurrence (i.e., occurrence with an edit distance of zero).

1.2. Problem 2

The input is a database *DB* of sequences $S_1, S_2, \ldots, S_n$. Input also are integers P, D, and q. Output should be all the patterns in *DB* such that each pattern is of length P and it occurs in at least q of the n sequences. If a pattern V is output, it will mean that V is a string of length P present in one of the input sequences and also that V occurs in at least q of the input sequences. (A string U is considered an occurrence of a pattern V as long as the edit distance between U and V is at most D. For a definition of edit distance see e.g., [7].)

The TEIRESIAS algorithm of Floratos and Rigoutsos [5] addresses a problem similar to Problem 1. The run time of this algorithm is $\Omega(P^{P/2}M \log M)$, where M is the size of the database (i.e., the number of characters (or residues) in the database). An algorithm for Problem 2 has been given by Sagot [13] that has a run time of $O(n^2 m\, P^D |\Sigma|^D)$ where m is the average length of the sequences in *DB*. The space requirement of this algorithm is $O(n^2 m)$. An algorithm with an expected run time of $O(nm + D(nm)^{1+\text{pow}(\varepsilon)} \log nm)$, where $\varepsilon = D/P$ and $\text{pow}(\varepsilon)$ is an increasing concave function, has been proposed by Adebiyi and Kaufmann [2]. The value of $\text{pow}(\varepsilon)$ is roughly 0.9 for protein and DNA sequences.

In this paper we present algorithms for Problems 1 and 2. All of our algorithms are elegant and use only such simple data structures as arrays. The goal of Problem 1 is to identify all the patterns of length at most P (with anywhere from 0 to $\lfloor P/2 \rfloor$ wild card characters) in a given database. In particular, the output should be all the patterns together with a count of how many times each pattern occurs. Optionally a threshold value for the number of occurrences could be supplied. Determining this threshold is a challenging task. One way of determining this threshold is to rank the motifs in the order of the number of their occurrences and choosing certain number of them (either because they are over-represented or because they are under-represented). Another way of determining the threshold is by analyzing the table of occurrences of all the patterns in the database together with a model for the biological sequences under concern. This differs from the first way in that here the number of occurrences of any motif will be weighted as dictated by the model. A typical value for P is 10. We present a simple sorting based algorithm for Problem 1 called the SMS (Simple Motif Search) algorithm. This algorithm has been coded and experimental results have been obtained. The run time of this algorithm is $O(P^{P/2}M)$ for a given pattern length P, the number of wild cards being at most $\lfloor P/2 \rfloor$. The number of residues in the database is M. We present several strategies to parallelize this algorithm. We present speedup results from our parallel implementation.

Also, we present two different algorithms for solving Problem 2. The first algorithm (called DMS) is deterministic that runs in time $O(n^2 m\, P^D |\Sigma|^D)$ using $O(nm\, D + P^D |\Sigma|^D)$ space. In contrast, the algorithm of [13] takes $O(n^2 m\, P^D |\Sigma|^D)$ time and uses $O(n^2 m/w)$ space where w is the word length of the computer. The algorithm of [13] uses suffix trees whereas DMS employs arrays. We expect

DMS to perform better in practice than the algorithm in [13]. DMS is amenable to parallelization. The second algorithm we present is a randomized algorithm that has the potential of performing better than the algorithms of [13] and [2]. This is a Monte Carlo algorithm with a run time of $O(((n^2m^2 \log n)/q)D + gmnD)$, where g is the number of P-mers in the database DB that occurring around q or more sequences in DB. When q is large, the above run time could be $o(mn + D(mn)^{1+\text{pow}(\varepsilon)} \log mn)$. We also calculate the expected value of g.

2. SOLUTION

2.1. Simple motif search (SMS) (Problem 1)

As has been pointed out before, we are interested in identifying all the patterns of length at most P (with anywhere from 0 to $\lfloor P/2 \rfloor$ wild card characters). For every pattern, the number of occurrences should be output. How does a biologist identify biologically important patterns? This is a challenging task for biologists and will not be addressed in this paper.

Define a (u, v)-class as a class of patterns where each pattern has length u and has exactly v wild card characters. For example, GA??C?T belongs to (7, 3)-class. Note that there are $\binom{u-2}{v}|\Sigma|^{(u-v)}$ patterns in a (u, v)-class. Similar notations have been used before. To identify the patterns in a (u, v)-class, we perform $\binom{u-2}{v}$ sorts.

More specifically, for each possible placement of v wild card characters (excluding at the end positions) in a sequence of length u, we perform a sorting. As an example, consider a case where $u = 5$ and $v = 2$. There are three possible placements: C??CC, CC??C, and C?C?C, where C corresponds to any residue. Call every placement as a (u, v)-pattern type. For every (u, v)-pattern type, we perform the following steps.

2.1.1. Algorithm SMS

For every (u, v)-pattern type do

1. If R is a pattern type in (u, v)-class, we generate all possible u-mers in all the sequences of DB. If the sequences in DB have lengths $l_1, l_2, \ldots, l_n$, respectively, then the number of u-mers from S_i is $l_i - u + 1$, for $1 \leq i \leq n$.
2. Sort all the u-mers generated in step 1 only with respect to the non-wild card positions of R. For example, if the pattern type under concern is CC??C?C, we generate all possible 7-mers in DB and sort the 7-mers with respect to positions 1, 2, 5, and 7. Employ radix sort (see e.g., [7]).
3. Scan through the sorted list and count the number of occurrences of each pattern.

The run time of the above algorithm is $O(\binom{u-2}{v}M\frac{u}{w})$ for a (u, v)-class, where M is the total number of residues in DB and w is the word length of the computer.

Now we consider the problem of identifying all of the following patterns: The maximum length is 10. Pattern lengths of interest are: 3, 4, 5, 6, 7, 8, 9 and 10. The maximum number of wild cards are 1, 2, 2, 3, 3, 4, 4 and 5, respectively. In other words we are interested in: (10, 5)-class, (10, 4)-class, ..., (10, 1)-class, (9, 4)-class, (9, 3)-class, ..., (9, 1)-class, ..., (4, 2)-class, (4, 1)-class, and (3,1)-class. Thus the total number of sorts done is

$$\sum_{i=0}^{5}\binom{8}{i} + \sum_{i=0}^{4}\binom{7}{i} + \sum_{i=0}^{4}\binom{6}{i} + \sum_{i=0}^{3}\binom{5}{i} + \sum_{i=0}^{3}\binom{4}{i} + \sum_{i=0}^{2}\binom{3}{i} + \sum_{i=0}^{2}\binom{2}{i} + \sum_{i=0}^{1}\binom{1}{i} = 429.$$

Theorem. *SMS algorithm runs in time* $O(P^{P/2}M)$.

2.2. Random sampling

Random sampling can be employed to speedup computations. In this section we describe a simple form of sampling as it applies to Problem 1.

Let the input database DB consist of the sequences $S_1, S_2, \ldots, S_n$. Assume that the problem is to determine how frequent are certain patterns. In particular, for each pattern we want to determine if the number of sequences in which it occurs is at least q, where q is a given threshold. We can use the following strategy to solve this problem. Randomly choose a sample S of εn sequences (for some appropriate value ε), identify the patterns that are frequent in this sample (with an appropriately modified threshold value), and output these patterns.

Analysis. Let U be a pattern that occurs q times in DB. Then the number of occurrences q' of U in S is Binomially distributed with a mean of εq. Using Chernoff bounds, $\Pr[q' \leq (1-\alpha)\varepsilon q] \leq \exp(-\alpha^2\varepsilon q/2)$. if $q \geq \frac{2\beta \ln n}{\alpha^2\varepsilon}$, then this probability is $n^{-\beta}$. Refer to a probability of $\leq n^{-\beta}$ as *low probability* as long as β is any constant ≥ 1. i.e., if a pattern U occurs q times in DB then its occurrence in S will be at least $(1-\alpha)\varepsilon q$ with high probability. Let $M =$

$\sum_{i=1}^{n} |S_i|$. i.e., M is the number of residues in DB. The total number of patterns that pass the threshold is clearly $<M$. If $q \geq \frac{2}{\alpha^2\varepsilon}(\beta \ln n + \ln M)$, then each pattern that occurs at least q times in DB will occur at least $(1-\alpha)\varepsilon q$ times in the sample.

Also, if a pattern U occurs at most $\frac{1-\alpha}{1+\alpha}q$ times in DB, then the expected number of occurrences of U in the sample is $\frac{1-\alpha}{1+\alpha}\varepsilon q$.

Using Chernoff bounds, this number will be $\leq(1-\alpha)\varepsilon q$ with probability $\geq 1 - \exp(-\alpha^2 \frac{1-\alpha}{1+\alpha}\frac{\varepsilon}{3}q)$. Thus if $q \geq \frac{3}{\alpha^2\varepsilon}\frac{1+\alpha}{1-\alpha}(\beta \ln n + \ln M)$, then every pattern that occurs at most $\frac{1-\alpha}{1+\alpha}q$ times in DB will occur at most $(1-\alpha)\varepsilon q$ times in S. We arrive at the following:

Lemma. *Consider the problem of identifying patterns in a database of n sequences. Each pattern of interest should occur in at least q of the input sequences. To solve this problem it suffices to use a random sample of size εn and a sample threshold of* $(1-\alpha)\varepsilon q$ *In this case, with high probability, no pattern that has an occurrence of less than* $\frac{1-\alpha}{1+\alpha}q$ *in DB will pass the sample threshold, provided* $q \geq \frac{3}{\alpha^2\varepsilon}\frac{1+\alpha}{1-\alpha}(\beta \ln n + \ln M)$.

Example. We present two examples to illustrate the usefulness of sampling. In particular, we want to see how small could ε be. For a given n, M, q, we fix suitable values for β and α and use the above constraint on q to evaluate the value of ε. The value of α that minimizes $(1/\alpha^2)((1-\alpha)/(1+\alpha))$ is approximately 0.6. Thus we use this value for α. We fix the value of β to be 1.

Consider the case of $n = 1000$; $m = 200$; $M = 200000$; $\alpha = 0.6$; $\beta = 1$. Here m refers to the length of each sequence. The condition on q becomes: $q \geq (666.6/\varepsilon)$. If $q = 800$, then it means that ε could be as small as 0.833.

As another example, consider the case where: $n = 10000$; $m = 200$; $M = 2000000$; $\alpha = 0.6$; $\beta = 1$. The condition on q becomes: $q \geq (833.25/\varepsilon)$. If $q = 5000$, the value of ε could be as small as 0.167.

2.3. Motif search with edit distance (Problem 2)

In this section we consider Problem 2. Here the input is a database DB of sequences $S_1, S_2, \ldots, S_n$. Input also are integers P, D, and q. The output should be all the patterns present in the DB such that each pattern is of length P and it occurs in at least q of the n sequences. A string U is considered an occurrence of a pattern V as long as the *edit distance* between U and V is at most D.

An algorithm for the above problem has been given by Sagot [13] that has a run time of $O(n^2 m P^D |\Sigma|^D)$ time where m is the average length of the sequences in DB. This algorithm uses $O(n^2 m/w)$ space where w is the word length of the computer. An algorithm with an expected run time of $O(nm + D(nm)^{1+\text{pow}(\varepsilon)} \log nm)$ where $\varepsilon = D/P$ and $\text{pow}(\varepsilon)$ is an increasing concave function has been given in [2]. The value of $\text{pow}(\varepsilon)$ is roughly 0.9 for protein and DNA sequences.

2.3.1. A deterministic algorithm (DMS)

In this section we describe a sorting based algorithm called DMS that has the same run time as that of [13] but has the potential of performing better in practice. The basic idea behind the algorithm is: We generate all possible P-mers in the database. There are at most mn such P-mers and these are the patterns of interest. For each such P-mer we want to determine if it occurs in at least q of the input sequences. Let U be one of the above P-mers. If V is a string such that the edit distance between U and V is at most D, then we say V is a neighbor of U. We generate all the neighbors of U. For each neighbor V of U we determine a list of input sequences in which V is present. These lists (over all possible neighbors of U) are then merged to obtain a list of input sequences in which U occurs (within an edit distance of D).

Note that if U is a P-mer, then its neighbors will have a length in the interval $[P - D, P + D]$. In other words, there are $(2D + 1)$ possible values for the lengths of the neighbors of U. Also note that more than one neighbor of U could have the same length. Corresponding to each r-mer X (where r is an integer in the interval $[P-D, P+D]$) present in the input, we keep a 4-tuple: (X, *SeqNum*, *Pos*, 0). Here *SeqNum* is an index of the input sequence I that X belongs to. There are $O(nm\,D)$ such 4-tuples. The indexing of the input sequences can be done arbitrarily (e.g., in the order in which they appear in the input). *Pos* is the starting position of X in I. The fourth entry (0) indicates that X is an r-mer present in one of the input sequences. For every neighbor V of U (U being a P-mer present in the input), we keep a 4-tuple as well: (V, *SeqNum*, *Pos*, 1). Here *SeqNum* is the index of the sequence I that U belongs to and *Pos* is the starting position of U in I. The fourth entry (1) indicates that this 4-tuple corresponds to a neighbor. We provide details of the algorithm next.

Note that each P-mer present in the input could be represented as a pair (*SeqNum*, *Pos*) where *SeqNum* is the index of the sequence in which the P-mer is present and *Pos* is the starting position of the P-mer in this sequence. We make use of an array $A[1:n, 1:m, 1:n]$. At the end of the algorithm this array will have the following property: $A[SeqNum, Pos, j] = 1$ if and only if the P-mer (*SeqNum*, *Pos*) occurs in the input sequence with index j $(1 \leq j \leq n)$.

1. Generate 4-tuples for all r-mers present in DB where $r \in [P - D, P + D]$. Each of these 4-tuples has a 0 as its fourth entry. Call this collection C. Sort C in lexicographic order and eliminate duplicates among these 4-tuples for which the first two entries are the same to get L_1. Note that the first entries of the 4-tuples in C could be of different lengths. A simple way of sorting these 4-tuples is to group them into $(2D + 1)$ groups one corresponding to each possible length of the first entry and handle the groups separately. L_1 has $O(nmD)$ entries. Now sort the 4-tuples of L_1 with respect to their first and fourth entries (in lexicographic order) to get L_2.
2. For every distinct P-mer U present in DB generate all the strings V such that U and V are at an edit distance of at most D. The number of such neighbors for a given U is $O(P^D|\Sigma|^D)$ (a proof follows). This generation is done using the algorithm of Myers [11]. Form the 4-tuples corresponding to all possible neighbors of all the distinct P-mers in DB. Each of these 4-tuples has 1 as its fourth entry.
3. Sort all the 4-tuples generated in step 2 with respect to their first and fourth entries. The total number of 4-tuples is $O(nm\,P^D|\Sigma|^D)$. Let L_3 be the sorted sequence.
4. Merge L_3 with L_2 to get L_4. We can think of L_4 as consisting of groups where a group has 4-tuples with the same first entry. A group itself can be thought of as consisting of two subgroups. The 4-tuples of the first subgroup have 0 as their fourth entry. The 4-tuples of the second subgroup have 1 as their fourth entry for each group G of L_4 do

   Let G_1 and G_2 be the two subgroups of G. Identify the distinct sequence indices (i.e., second entries) in the 4-tuples of G_1. Let these indices be $i_1, i_2, \ldots, i_k$. Note that $k \leq n$.

   for each 4-tuple (V, *SeqNum, Pos,* 1) in G_2 do

   $A[SeqNum, Pos, a_j] := 1$, for $1 \leq j \leq k$. (I.e., the P-mer (*SeqNum*, *Pos*) occurs in the input sequences S_{a_j}, for $1 \leq j \leq k$.

Scan through the array A to output the right P-mers. In particular, output (*SeqNum*, *Pos*) if $A[SeqNum, Pos, j]$ is 1 for $1 \leq j \leq n$.

Theorem. *The above algorithm runs in time* $O(n^2 m\,P^D|\Sigma|^D)$. *The space used is* $O(nm\,P^D|\Sigma|^D)$. *The space used can be reduced to* $O(nm\,D + P^D|\Sigma|^D)$.

Proof: The run time of step 1 is $O(nm\,D(P/w))$ using radix sort algorithm.

Let U be any P-mer. Then the number of strings V such that the edit distance between U and V is at most D is $O(P^D|\Sigma|^D)$ as argued next. The same fact has been proven by Crochemore and Sagot as well [13]. Let $N(t)$ be the number of strings obtainable from U by performing t operations (inserts, deletes, and substitutions) on U. The number of strings of interest is then $\sum_{t=0}^{D} N(t)$. Of the t operations let the number of inserts, deletes, and substitutions be i, del, s, respectively with $i + \text{del} + s = t$. For a given choice of i, del, s, it is easy to see that the number of patterns obtainable is

$$\binom{P+i}{i}\binom{P}{\text{del}}\binom{P}{s}|\Sigma|^{s+i}.$$

As a result,

$$N(t) \leq \frac{(t+1)(t+2)}{2}\left(\frac{(P+t)e}{t}\right)^t$$

using the fact that

$$\binom{a}{b}(t) \leq \left(\frac{a\,e}{b}\right)^b.$$

Finally, summing $N(t)$ over all ts, we see the result (as long as $D \geq 6$).

Thus the generation of all the patterns in step 2 takes $O(nm\,P^D|\Sigma|^D)$ time (using the algorithm in [11]).

In step 3 sorting takes time $O(P^D|\Sigma|^D(P/w))$.

Merging in step 4 also takes time $O(P^D|\Sigma|^D(P/w))$. For each 4-tuple of a given G_2 the time spent is $O(k)$ where k is the number of distinct sequence indices in the corresponding G_1. Since $k \leq n$, the total time in processing all the G_2's is $O(n^2 m\,P^D|\Sigma|^D)$. These observations prove the theorem (assuming that $P/w = O(n)$). □

The above algorithm is simpler than the ones in [2, 13]. The algorithms in [2, 13] employ suffix trees. In comparison the above algorithm uses only arrays. The above algorithm can be expected to perform better than that of [2, 13] in practice.

In practice the above algorithm is expected to run much faster. It is easy to see that the run time of the above algorithm is $O(nm\,P^D|\Sigma|^D z)$ where z is the maximum number of distinct sequence indices in the 4-tuples of any G_1. The expected value of z can be calculated as follows.

Let X be any r-mer present in DB. The expected number of sequences that X occurs in is the same as the expected value of z. If I is any input sequence and j is a fixed position in I, probability that X is present in I starting from j is $(1/4)^r$. Thus, probability that X is present somewhere in I is $\leq m(1/4)^r$. As a result, the expected number of sequences in which X is present is $\leq nm(1/4)^r$. Thus the expected value of z is $\leq nm(1/4)^{(P-D)}$.

As an example, if $n = 20$; $m = 600$; $P - D = 10$, the expected value of z is less than 1.

2.3.2. A randomized algorithm

In this section we present a simple randomized algorithm that has the potential of performing better than the algorithms of [2, 13]. The algorithms in [2, 13] employ suffix trees and our algorithm uses arrays. Before presenting the randomized algorithm we present a very simple algorithm. The randomized algorithm is based on this simple algorithm. This algorithm works as follows.

1. Generate all possible P-mers in DB. Let the collection of these P-mers be C. There are at most mn elements in C. Duplicates in C could be eliminated by a simple radix sort.
2. For every P-mer U in C, compute the number of occurrences of U in DB. This can be done in time $O(nmD)$ using the algorithm of Galil and Park [6]. (See also [1, 8, 10, 11, 14]).

Thus we get the following Theorem.

Theorem. *Problem 2 can be solved in time* $O(n^2m^2D)$.

A Randomized Algorithm. A randomized algorithm can be developed based on the above algorithm.

1. Generate all possible P-mers in DB. Let C the collection of these P-mers. C has at most nm elements.
2. For each element U in C, pick a random sample SU from DB of $\frac{16\alpha n \ln n}{q}$ sequences where α is the probability parameter (assumed to be a constant). Count the number of occurrences N_U of U in the sample. This will take time $|S_U|mD$ (using the algorithm of Galil and Park [6]) for a single U.
3. For each U in C such that $N_U > 10.34\alpha \ln n$, compute the occurrences of U in the entire input DB. If the number of occurrences of U in DB is q or more, then output U.

Theorem. *The above algorithm runs in time* $O(\frac{n^2m^2 \log n}{q} D + gmnD)$ *where* g *is the number of* P*-mers that pass the test in step 3. Also, the probability of an incorrect answer is no more than* $n^{-\alpha}nm$. *The space used is linear in the input size.*

Proof: The run time is easy to see. Note that if a P-mer occurs in less than q input sequences, it will never be output. If a P-mer U occurs in at least q sequences of DB, then the number of occurrences of U in S_U (i.e., the value of N_U) is lower bounded by a binomial random variable with mean $16\alpha \ln n$. An application of the Chernoff bounds (second equation) with $\varepsilon = 1/(2\sqrt{n})$ shows that the probability that N_U is $<10.34\alpha \ln n$ is no more than $n^{-\alpha}$. On the same token, let U' be a P-mer that occurs in at most $(3/8)q$ of the input sequences. The number of occurrences $N_{U'}$ of U' in the sample is a binomial with mean $6\alpha \ln n$. Using Chernoff bounds equation 3 with $\varepsilon = 1/\sqrt{2}$, probability that $N_{U'}$ exceeds $10.25\alpha \ln n$ is at most $n^{-\alpha}$.

In summary, if a pattern occurs in q or more input sequences, it will pass the test of step 3 with high probability. Moreover, not many spurious patterns will pass the test of step 3. If a pattern has to pass the test of step 3, then it has to occur in at least $(3/8)q$ of the input sequences (in a high probabilistic sense). Therefore a high probability upper bound on g is the number of patterns that occur in $(3/8)q$ or more of the input sequences. Also note that there are at most nm patterns of interest. □

Note that this algorithm has the potential of performing better than those of [2, 13], especially for large values of q. When q is large (εn for some constant fraction ε, for instance), g can be expected to be small and hence the entire run time could be $O(D(nm)^{1+\mathrm{pow}(\varepsilon)} \log nm)$. Next we show that the expected value of g is very small.

Assume that every residue in each input sequence is picked randomly (from an alphabet of size 4). Let the input consist of n sequences of length m each. Let V be an l-mer and S be any input sequence. Let i be any position in S. Probability that V is present in S starting from position i is $(1/4)^l$. Thus, the probability that V is present in S starting from some position is $\leq (m - l + 1)(1/4)^l$. For every P-mer U there are $\leq c\, P^D|\Sigma|^D$ (for some constant c) strings X such that the edit distance between U and X is at most D [13]. Call any such string X as a *neighbor* of U. The length of X is in the interval $[P - D, P + D]$.

In Problem 2, we are supposed to do the following: For every P-mer U in the input, check if it occurs in at least q of the input sequences.

Therefore, probability that either U or any of its neighbors is present in S is at most $p_1 = (m - P + D + 1)cP^D|\Sigma|^D(1/4)^{P-D}$.

As a result, probability that U occurs in q or more of the input sequences is at most $P_2 = \sum_{i=q}^{n} \binom{n}{i} p_1^i(1 - p_1)^{n-1} \leq \sum_{i=q}^{n} \binom{n}{i} p_1^i$ Using Stirling's approximation, we see that

$$p_2 \leq \frac{2^n\sqrt{2}}{\sqrt{\pi n}} \sum_{i=q}^{n} p_1^i.$$

By simple arithmetic we see that $p_2 \leq 1/(mn)$ when

$$q \geq \frac{(n+1) - \log(mn) - (1/2)\log(\pi n)}{2(P-D) - \log(m - P + D + 1) - \log c - D\log|\Sigma| - D\log P}.$$

If $p_2 \leq 1/(mn)$, then the expected number of patterns that occur in q or more of the input sequences is ≤ 1. Thus indeed the value of g will be small even if q is not this high!

As a numerical example, consider the case: $P = 20$, $D = 2$, $m = 256$, $n = 500$. In this case, the condition on q becomes: $q \geq 36$. Also, if α is 4, the probability of an incorrect answer is 2.048×10^{-6}. This is also an upper bound on the expected number of patterns that will be missed by the algorithm! (A pattern is missed by the algorithm if it occurs in at least q of the input sequences but the algorithm fails to detect this).

The above analysis could be tightened further.

3. EVALUATION OF PERFORMANCE

We have implemented the algorithm SMS (for Problem 1) both sequentially and in parallel. In this section we describe the experimental results.

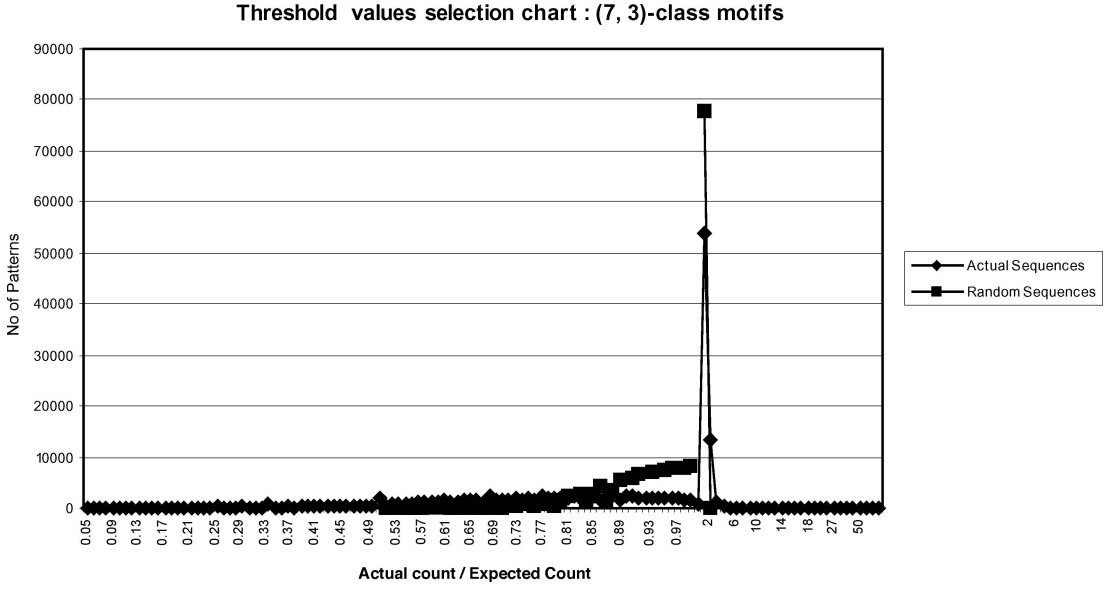
3.1. Sequential implementation

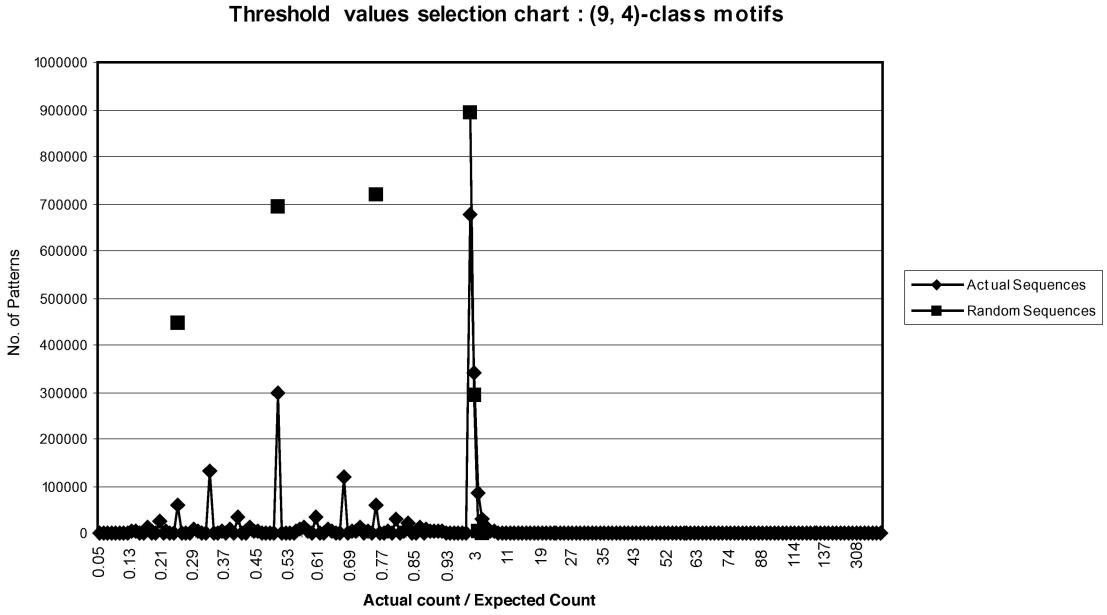
We employ the following form of radix sort: Let $k_1, k_2, \ldots, k_N$ be a sequence of keys to be sorted where each key is a string of residues. Sorting is done with respect to d residues at a time. The optimal value for d can be decided empirically. For the proteome database, a value of $d = 3$ proved to be optimal.

When $d = 3$, the algorithm runs as follows. The only data structures used are arrays. For the proteome database, $|\Sigma| = 20$. We can think of each key to be sorted as an integer in the range [1,8000]. An array c[1:8000] whose entries are initialized to zeros is employed. There are two phases in the algorithm each phase involving a scan through the input sequence. In the first phase input keys are processed one at a time starting from k_1. When key k_i is processed, $c[k_i]$ is incremented by one. Thus at the end of the first phase, $c[j]$ has the number of input keys whose value is j (for $1 \leq j \leq 8000$). In the second phase prefix sums of the array c are computed. A second scan through the input sequence is done and each key is output in an array in an appropriate (stable-sorted) place. The prefix sums are useful in deciding an appropriate output index for each key.

We have employed our algorithm SMS on various proteome sequences. As an example, we report the results pertaining to the Human proteome sequences (RefSeq database). This database has 10,046,356 residues and 19,244 sequences. The average length of the sequences in the database is 522.

On a Pentium 4, 2.4 Ghz machine with 1 GB RAM, SMS takes around 7.25 hours. The graphs above show the distribution of patterns in RefSeq and a database of random sequences (of a comparable size). The patterns shown are for (7, 3)-class and (9, 4)-class. In the x-axis we show the ratio of the actual number of occurrences of a pattern to the expected number of occurrences. In the y-axis we show the number of patterns that have a specific ratio.





3.2. A speedup technique

The run time of SMS can be improved using the fact that

$$\binom{n}{i} = \binom{n-1}{i} + \binom{n-1}{i-1}.$$

Consider the processing of a (u, v)-class. $\binom{u-2}{v}$ sorts are performed to process this class. Processing of this class can be done in conjunction with the processing of the $(u-1, v)$ and $(u-1, v-1)$ classes and in the process a lot can be gained in the run time.

As an example consider the (10, 5)-class. For every (10, 5) pattern type, a sort has to be done. These sorts can be accomplished while processing (9, 5) and (9, 4) classes as follows.

While processing a (9, 5) pattern, all 9-mers are sorted with respect to 4 residues. From the sorted list generate all 10-mers by adding one residue to each 9-mer (for example this could be the residue to the left of the 9-mer in the corresponding sequence). Now sort these 10-mers. Note that this sorting involves sorting with respect to only one more residue. These patterns cover a part of all the (10, 5) patterns.

While processing a (9, 4) pattern, all 9-mers are sorted with respect to 5 residues. Let the sorted list of these 9-mers after sorting them with respect to 4 residues be L'. Sort L' with respect to the next residue to complete the processing of the (9, 4) pattern. Add one more residue to every 9-mer in L' and sort L' with respect to the added residue. This sort is a sort corresponding to a (10, 5) pattern. These (10, 5) patterns are the remaining patterns in the (10, 5) class.

The above technique can be employed to speedup SMS by a constant factor and hence will be of interest in practice.

3.3. More improvements

We can speedup SMS further and also reduce the memory requirements as follows. Let $S_1, S_2, \ldots, S_n$ be the input sequences where $S_i = s(i, 1), s(i, 2), \ldots, s(i, l_i)$, l_i being the length of the sequence S_i. Each residue in the database can be uniquely identified by a combination of the sequence index and the position of the residue in that sequence.

Definition. A position pair is defined as a pair of indices (i, j) and this pair represents the jth residue in S_i.

The basic idea is to save memory by not explicitly generating all possible u-mers (as mentioned in step 1 of Algorithm SMS). We work with the position pairs to perform the sorts (mentioned in step 2 of Algorithm SMS). In particular, each position pair corresponds to a u-mer. Whenever any portion of a u-mer is needed, this portion is obtained from the list of position pairs.

Modify Algorithm SMS as follows. The problem is to find all of the following patterns: (u, v)-class for $2 \leq u \leq P$ and $0 \leq v \leq \lfloor u/2 \rfloor$.

1. Generate X, an array of position pairs such that, $X[u]$ holds all the position pairs for patterns of length u. We

know that, $X[P]$ will have position pairs (i, j) for $1 \leq i \leq n$ and $1 \leq j \leq l_i - P + 1$. Note that $X[P-1]$ can be obtained from $X[P]$ by adding only an additional n position pairs. Likewise, $X[P-2]$ can be obtained from $X[P-1]$ adding an additional n position pairs, and so on. Also note that the number of position pairs in any $X[u]$ is $O(M)$ where M is the total number of residues in the input.

2. Consider the position of wild cards in a pattern of length P. All these positions would generate valid patterns of length $2 \leq u < P$, where $u = 2$ when there are no wild cards in the pattern and $u =$ one plus the position of the last wild card in P. Thus, if we sort the position pairs $X[u]$ for $2 \leq u \leq P$, such that the above condition is satisfied, the time to sort will be reduced to the time required to sort one array of $O(M)$ elements plus the time to sort f arrays of n elements each, where f is the number of arrays in X that satisfy the condition (i.e., instead of $(f+1)$ sorts that require time $O(M)$, the new time will be one sort requiring $O(M)$ time and f sorts requiring $O(n)$ time each).
3. After sorting the respective arrays in X, a scan of the array $X[P]$ will result in the output of the count of u-mers of length P. During this scan, let the position pair (i, j) and the value of the u-mer at that position, say, val_{ij} be stored in an array, say *OutArray*. To report patterns of length $P > u$, 2, merge *OutArray* and $X[u]$, each time updating the val_{ij} in the elements of *OutArray* to the length of the u-mer considered for the scan as follows:

$$\text{val}_{ij} = (\text{val}_{ij} - \text{value of the least significant symbol})/|\Sigma|.$$

This reduces the time for calculating the value of patterns in the subsequent scans to report the number of occurrences of the u-mers considered.

The modified algorithm SMS is as follows:

3.3.1. Algorithm modified-SMS

1. Generate the position pairs array X;
2. For every wild card option for the pattern of length P, we perform the following steps.
   i. Find u, $P \geq u \geq 2$ for which the wild card option would result in a valid pattern.
   ii. Sort $X[u]$ for $P \geq u \geq 2$ with respect to the u-mers at the positions given by the position pairs of $X[u]$ generated in step 1 only with respect to the non-wild card positions. For example, if the pattern type under concern is CC??C?C, we sort $X[7]$ with respect to the 7-mers at the positions given by the position pairs of $X[7]$ only with respect to positions 1, 2, 5, and 7. Employ radix sort (see e.g., [7]).
   iii. Scan through the sorted array $X[P]$ and count the number of occurrences of each pattern, also producing *OutArray*.
   iv. To report patterns of length $P > u$, 2, merge *OutArray* and $X[u]$, each time updating the val_{ij} in the elements of *OutArray* to the length of the u-mer considered.

3.4. Experimental data

We have employed our algorithm SMS on various proteome sequences. As an example, to report novel motifs in the Human proteome sequences of RefSeq database, that has 10,046,356 residues and 19,244 sequences, with the average length of the sequences as 522 amino acids, on a Pentium 4, 2.4 GHz machine with 1 GB RAM, SMS takes around seven hours.

3.5. Parallelism

SMS is amenable to parallel implementations. One possibility is to partition the number of sorts equally among the processors. For example, if there are two processors, then a reasonable partition is for the first processor to work on a maximum pattern length of 10 and the second processor to work on the remaining maximum pattern lengths. Processor 1 will perform 219 sorts and the second processor will do 210 sorts.

A second possibility is to partition the sequences as equally among the processors as possible and finally merge the occurrence numbers of patterns.

A third possibility is to treat each sort as a job. To begin with all the jobs are available. To begin with, each processor takes up an available job. As soon as a job is taken up (by some processor) it is marked unavailable. When a processor completes its job, it takes up another available job. Computation stops when there are no more available jobs.

We have employed the third approach to parallelize SMS. The speedups obtained have been very close to linear and are shown in the Table 1. Extrapolating on our experience

Table 1. Performance of the SMS algorithm on real biological sequence data (refer section 3 for details)

No. of Processors	Run time (Hours)	Speedup
1	7.25	1.00
2	3.67	1.98
4	1.89	3.84
6	1.27	5.73

we predict that on a 64-node cluster, SMS will run in less than an hour on the RefSeq database of human proteome sequences.

We could also parallelize the DMS algorithm. From the description of DMS we see that all the steps in the algorithm can be completed employing sorting algorithms. Optimal parallel sorting algorithms can be found in the literature (see e.g., [7]). If one employs any of these algorithms, it is easy to see that DMS can be parallelized to run in $O(\log mn)$ time (for example on the Concurrent Read Exclusive Write Parallel Random Access Machine model), the work done being nearly the same as the sequential run time.

4. DISCUSSION

In this paper we have considered two versions of the motif search problem and offered sequential and parallel exact solutions for these versions. The algorithms presented have the potential of performing well in practice. An interesting open problem is to implement all the algorithms that have been proposed for Problem 2 in the literature and determine under what conditions which algorithms will perform better.

This research has been supported in part by the NSF Grants CCR-9912395 and ITR-0326155. A preliminary version of this paper was presented in the *Third Asia-Pacific Bioinformatics Conference* (*APBC*), 2005.

APPENDIX A: CHERNOFF BOUNDS

A *Bernoulli trial* is an experiment with two possible outcomes viz. *success* and *failure*. The probability of success is p. A binomial random variable X with parameters (n, p) is the number of successes in n independent Bernoulli trials, the probability of success in each trial being p. We can get good estimates on the tail ends of binomial distributions (see e.g., [4]). In particular, it can be shown that,

Lemma A.1. *If X is binomial with parameters (n, p), and $m > np$ is an integer, then,*

$$\Pr[X > m] \leq (np/m)^m \exp(m - np)$$

Also,

$$\Pr[X \leq (1 - \varepsilon)np] \leq \exp(-\varepsilon^2 np/2)]$$

and

$$\Pr[X \geq (1 + \varepsilon)np] \leq \exp(-\varepsilon^2 np/3)]$$

for all $0 < \varepsilon < 1$.

REFERENCES

1. Adebiyi EF, Jiang T, Kaufmann M. An efficient algorithm for finding short approximate non-tandem repeats. Bioinformatics 2001; 17(1): S5–S12.
2. Adebiyi EF, Kaufmann M. Extracting common motifs under the Levenshtein measure: Theory and experimentation, Proc. Workshop on Algorithms for Bioinformatics (WABI). Springer-Verlag LNCS 2002; 2452: 140–156.
3. Buhler J, Tompa M. Finding motifs using random projections, Proc. Fifth Annual International Conference on Computational Molecular Biology (RECOMB) 2001.
4. Chernoff H. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. Annals of Math Statistics 1952; 23: 493–507.
5. Floratos A, Rigoutsos I. On the Time Complexity of the TEIRESIAS Algorithm, Research Report RC 21161 (94582), IBM TJ, Watson Research Center 1998.
6. Galil Z, Park K. An improved algorithm for approximate string matching. SIAM Journal of Computing 1990; 19(6): 989–999.
7. Horowitz E, Sahni S, Rajasekaran S. Computer Algorithms. W. H. Freeman Press, 1998.
8. Landau GM, Vishkin U. Introducing efficient parallelism into approximate string matching and a new serial algorithm, Proc. ACM Symposium on Theory of Computing 1986: 220–230.
9. Martinez HM. An efficient method for finding repeats in molecular sequences. Nucleic Acids Research 1983; 11(13): 4629–4634.
10. Myers EW. Incremental Alignment Algorithms and Their Applications, Technical Report 86-22, Department of Computer Science, University of Arizona, Tucson, AZ 85721, 1986.
11. Myers EW. A sublinear algorithm for approximate keyword searching. Algorithmica 1994; 12: 345–374.
12. Rajasekaran S, Balla S, Huang CH. Exact Algorithms for Planted Motif Challenge Problems, Proc. Asia-Pacific Bioinformatics Conference (APBC), 2005: 249–260.
13. Sagot MF. Spelling approximate repeated or common motifs using a suffix tree. Springer-Verlag LNCS 1998; 1380: 111–127.
14. Ukkonen E. Finding approximate patterns in strings. Journal of Algorithms 1985; 6: 132–137.