DIMACS Technical Report 2010-X
March 2010

Sublinear Selection Algorithms for Motif Finding

by

Pavel P. Kuksa
Dept. of Computer Science
Rutgers University
New Brunswick, New Jersey 08903

Vladimir Pavlovic
Dept. of Computer Science
Rutgers University
New Brunswick, New Jersey 08903

ABSTRACT

We consider the problem of identifying motifs, recurring or conserved patterns, in the sets of biological sequences. To solve this task, we present new deterministic and exact algorithms for finding patterns that are embedded as exact or inexact instances in all or most of the input strings. The proposed algorithms (1) improve search efficiency compared to existing exact algorithms by focusing search on a selected set of potential motif instances, and (2) scale well with the input length and the size of alphabet. While a variety of exact and probabilistic methods exist, our algorithms enhance pattern detection ability of these methods by (1) applying as a wrapper *speed-up mechanism* to a variety of common *exact* enumeration-based pattern finders, allowing to search for longer, less conserved motifs, (2) combining with *probabilistic* pattern finders as *candidate selectors* and accelerating search for pattern models. Our algorithms are orders of magnitude faster than existing exact algorithms for common pattern identification. We evaluate our algorithms on benchmark motif finding problems and real applications in biological sequence analysis and show that our algorithms exhibit significant running time improvements compared to the state-of-the-art approaches.

1 Introduction

Finding motifs or repeated patterns in data is of wide scientific interest [1, 2, 3, 4], with many applications in genomic and proteomic analysis. The motif search problem abstracts many important problems in analysis of sequence data, where motifs are, for instance, biologically important patterns. For example, elucidating motifs in DNA sequences is a critical first step in understanding biological processes as basic as the RNA transcription. There, the motifs can be used to identify promoters, the regions in DNA that facilitate the transcription. Finding motifs can be equally crucial for analyzing interactions between viruses and cells or identification of disease-linked patterns.

The task of motif finding for a given a set of sequences is to discover motifs and its instances without prior knowledge of the consensus motif string and the positions of its instances in the sequences (Figure 1).

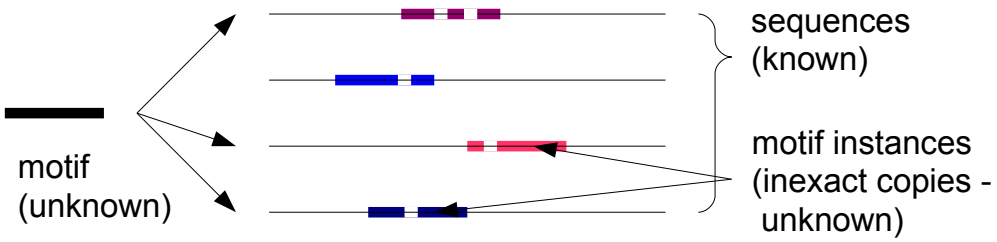

Figure 1: The motif search problem.

For the purpose of this study, motifs are (short) patterns that occur in an exact or *approximate* form in all or most of the strings in a data set. Consider a set of input strings $\mathcal{S}$ of size $N = |\mathcal{S}|$ constructed from an alphabet Σ. The solution for the (k, m, Σ, N)-motif finding problem (Figure 1) is the set $\mathcal{M}$ of k-mers (substrings of length k), $\mathcal{M} \subseteq \Sigma^k$, such that each motif string $a \in \mathcal{M}, |a| = k$, is at Hamming distance at most m from all (or almost all) strings $s \in \mathcal{S}$.

In this work, we focus on a deterministic, exhaustive approach to motif search. Exhaustive motif finding approaches are guaranteed to report all instances of motifs in a set of sequences, but are faced with the very high complexity of such search. We present new deterministic and exact algorithms (Section 3) for finding common patterns with the search complexity that scales well with the input length and size of the alphabet. Proposed algorithms improve motif search efficiency by focusing on the input instances that are more likely to be motif instances as opposed to using the entire input directly. Compared to existing exact algorithms (e.g., [5, 6, 7]) our algorithms improve search efficiency in the important cases of large-alphabet inputs (e.g., protein alphabet, extended DNA alphabet, etc.) and inputs of large length. As we show in the experiments, using both synthetic and real biological data, our algorithms are orders-of-magnitude faster than existing state-of-the-art deterministic search algorithms, especially

on large-alphabet inputs (e.g., protein data). This result extends applicability of the exact motif search algorithms to more complex problems requiring analysis of biological sequence data modeled as strings over large alphabets.

2 Related work

The problem of motif discovery has been studied extensively over the past two decades [8]. Within an important class of exhaustive methods (i.e. methods that are guaranteed to report all instances of the motifs), a number of approaches have been proposed, including graph methods (WINNOWER) [2], explicit trie traversal (MITRA) [5], voting algorithms [7], suffix trees [9, 6], sorting and enumeration [10], etc. Existing exhaustive algorithms use explicit exploration of the motif space and require time proportional to the size of the k-mer *neighborhood* which consists of all k-length substrings at Hamming distance of at most m from a k-mer. The number of (k,m)-neighbors for any particular k-mer is $V(k,m) = \sum_{i=0}^{m} \binom{k}{i}(|\Sigma|-1)^i$, and depends on the alphabet size, as well as the length k and the maximum number of allowed mismatches m. This can lead to high computational complexity and running times, as shown in Table 1.

Table 1: Exact algorithms for motif search

Algorithm	Time Complexity	Space Complexity
SPELLER [9]	$O(nN^2V(k,m))$	$O(nN^2/w)$
MITRA [5]	$O(knNV(k,m))$	$O(nNk)$
CENSUS [11]	$O(knNV(k,m))$	$O(nNk)$
Voting [7]	$O(nNV(k,m))$	$O(nV(m,k))$
RISOTTO [6]	$O(nN^2V(k,m))$	$O(nN^2)$
PMS [10]	$O(n^2NV(k,m))$	$O(n^2N)$

Notation: N-number of sequences, n-sequence length,
k - k-mer length, m-number of mismatches

Voting algorithms proposed in [7] explore local neighborhood sets $V(k,m)$ (e.g., using trees of size $O(k^m|\Sigma|^m)$) of the input k-mers and use an indicator array V of the maximum size $|\Sigma|^k$ to find motifs through *voting*. Each length-k substring observed in the input has at most one vote for each input sequence and gives this vote to all of its $V(k,m)$ neighbors. The substrings that occur in every input string will receive N votes and will be included in the output motif set $\mathcal{M}$. The algorithm takes $O(k^{m+1}|\Sigma|^m nN)$ time as it traverses neighborhood sets $V(k,m)$ of all k-mers in the input.

One of the most efficient exact algorithms for motif search, the mismatch tree (MITRA) algorithm [5], uses efficient trie traversal over all $O(|\Sigma|^k)$ possible motif patterns to find a set of motifs in the input strings. Compared to local neighborhood searches in voting algorithms, this algorithm efficiently traverses a single global tree of all $O(|\Sigma|^k)$ k-mer patterns. Under a trie-based framework [5, 12], the list of k-long contiguous substrings (k-mers) extracted

from given strings is traversed in a *depth-first* search manner with branches corresponding to all possible symbol substitutions from alphabet $|\Sigma|$. Each leaf node at depth k corresponds to a particular k-mer feature (either exact or inexact instance of the observed exact string features) and will contain a list of matching features from each string. The leaf nodes corresponding to motifs will contain k-mer instances from all (or almost all) strings. The complexity of the trie-based traversal algorithm for motif finding is $O(k^{m+1}|\Sigma|^m nN)$. Note that the algorithm essentially explores the neighborhood of all $O(nN)$ k-mers in the input to select motifs.

Another class of efficient exhaustive algorithms is based on sorting and enumeration [10, 13, 14]. The PMS1 algorithm [13] enumerates all possible k-mers for each input string s_i, creates lists L_i of all k-mers that are variants (neighbors) of the k-mers observed in s_i, and finds the motif set $\mathcal{M}$ as an intersection $\cap_i L_i$ of these lists. The PMSP algorithm [14] enumerates all possible neighboring k-mers for the first string s_1 and outputs k-mers that occur in every string with Hamming distance at most m, similar to the Voting algorithms [7]. The PMSprune algorithm [10] employs a more efficient search strategy to traverse the candidate space and is an improvement, in the expected case, over the PMSP.

In this study, we focus on tree-based algorithms for motif search. In particular, we consider global tree search algorithms (as mismatch tree (MITRA) algorithms) and local tree algorithms (as Voting algorithms) and show how the search complexity can be significantly improved for both approaches using fast candidate selection algorithms, which we describe in Section 3. We propose two motif search algorithms with selection that use for search a single global tree (Section 3.1) or use local tree searches (Section 3.2).

3 Motif selection algorithms

Existing exhaustive algorithms typically (e.g., [5, 7, 10]) use the entire input $\mathcal{S}$ (i.e. all the k-mers in the input) and find motif by essentially exploring neighborhood sets of every k-mer in the input. To improve their search complexity, we propose selection-based approach to motif search that uses a *reduced* set $\mathcal{R} \subset \mathcal{S}$ which contains only k-mer samples that are potential motif instances instead of all input samples $\mathcal{S}$. We obtain set $\mathcal{R}$ from $\mathcal{S}$ by removing all k-mer samples that do not satisfy motif constraints, as described below.

A necessary condition for a group of k-mers to have a shared, common (k, m)-neighbor (motif) is that the Hamming distance between any pair of k-mer patterns has to not exceed $2m$. This is true, since if some of the distances exceed $2m$, than corresponding pairs share no k-mers, and if the common neighbor exists, then the pairwise distances have to be at most $2m$, otherwise the common neighbor would not have existed.

We will use this condition to select k-mers from input $\mathcal{S}$ that are potential motif instances and place them in set $\mathcal{R}$. A particular k-mer a in the input is a potential motif instance if it is at minimum Hamming distance at most $2m$ from each of the input strings. All other k-mers that violate the above condition cannot be instances of a motif and can be discarded (i.e. using the reduced set $\mathcal{R} \subset \mathcal{S}$ results in same output as in case of using $\mathcal{S}$). The proposed reduced (feasible) set selection is illustrated in Figure 2.

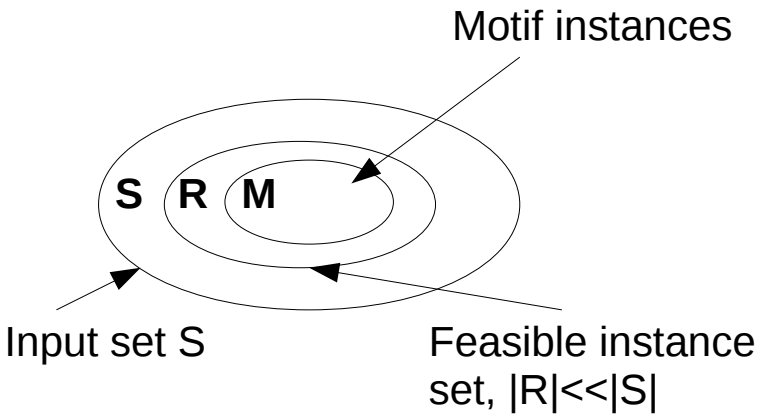

Figure 2: Feasible instance selection.

At first, it seems that in order to obtain a set of k-mers $\mathcal{R}$ at distance of at most $2m$ from every string one needs to compute all $O(n^2N^2)$ pairwise distances between input k-mers and select k-mers at distance of at most $2m$ — which would require *quadratic* running time. We will show next how to obtain a reduced set of k-mers in time *linear* in the input length n.

The reduced set $\mathcal{R}$ of potential motif instances, i.e. k-mers at distance of at most $2m$ from each of the input strings, can be obtained using our linear time selection algorithm (Algorithm 1). To select the valid k-mers (i.e. a set of potential motif instances), we use multiple rounds of count sort by removing iteratively $2m$ out of k positions and sorting the resulting sets of $(k-2m)$-mers. A k-mer is deemed a potential motif instance if it matched at least one k-mer from each of the other strings in at least one of the sorting rounds. The purpose of sorting is to group the same k-mers together; note that $(k-2m)$-mers corresponding to k-mers at distance of at most $2m$ will match exactly. Using a simple linear scan over the sorted list of all input k-mers, we can find the set of potential motif instances and construct $\mathcal{R}$. This algorithm is outlined below (Algorithm 1).

Algorithm 1 Selection algorithm

Input: set $\mathcal{S}$ of k-mers with associated sequence index L, distance parameter d

Output: set $\mathcal{R}$ of k-mers at distance d from each input string

1. Pick d positions and remove from the k-mers symbols at the corresponding positions to obtain a set of $(k-d)$-mers.
2. Use counting sort to order (lexicographically) the resulting set of $(k-d)$-mers.
3. Scan the sorted list to create the list of all sequences in which k-mers appear using sequence index L.
4. Output the k-mers that appear in every input string.

As we will see in the experiments (Section 4), the selection using Algorithm 1 significantly reduces the number of k-mer instances considered by the motif algorithm and improves its search efficiency. The number of selected k-mers, i.e. the size of $\mathcal{R}$, is small, especially

for large-alphabet inputs (e.g., sequences over protein alphabet). This can be seen from the expected case analysis. For this purpose we assume that sequences are generated from a background process with few motifs implanted in the background-generated sequences. Assuming an iid background model with equiprobable symbols, the expected number of k-mers in the input of N strings of length n that match each of the N strings with up to $2m$ mismatches by chance is

$$E[\mathcal{R}_B] = |\Sigma|^k(1-(1-p_{k,2m})^n)^N =$$
$$= |\Sigma|^k \left(1-\left(1-\sum_{i=0}^{2m}\binom{k}{i}\left(\frac{1}{|\Sigma|}\right)^{k-i}\left(\frac{|\Sigma|-1}{|\Sigma|}\right)^i\right)^n\right)^N,$$

where $p_{k,2m}$ is the probability that two randomly selected k-mers are at distance of at most $2m$. For instance, for a set of $N = 20$ protein sequences (sampled from alphabet $|\Sigma| = 20$) of length $n = 600$ the expected number of potential motifs of length $k = 13, m = 4$ by chance is about 8, with $p_{13,8} = 2.9\ 10^{-4}$. Given t implanted motif instances, the average number of k-mers that will be selected from nN input samples, or the expected size of $\mathcal{R}$, is

$$E[\mathcal{R}] = t + nN(1-(1-p_{k,2m})^t) + E[\mathcal{R}_B]$$

Since t and p are typically small, for small pn, $E[\mathcal{R}] \ll nN$, the number of k-mers in the input. In the protein example above the expected size of $\mathcal{R}$ is about $1 + 3 + 8 = 12$ for $t = 1$, which is orders of magnitude smaller than $nN = 12000$, signifying the importance of creating $\mathcal{R}$ first. This is empirically demonstrated in Section 4.

Using reduced set $\mathcal{R}$ of k-mers, the actual search complexity after the selection step (Algorithm 1) becomes *sublinear* in the input size (since the number of selected k-mers $R = |\mathcal{R}|$ is much smaller than input length $O(nN)$). For instance, the search complexity of the trie-based algorithms (e.g., [5]) can be reduced to $O(\binom{k}{2m}knN + RV(k,m))$ instead of $O(knNV(k,m))$, where $V(k,m)$ is $O(k^m|\Sigma|^m)$. This will lead to a more efficient search especially for large-alphabet since a possibly large input $\mathcal{S}$ of size $O(nN)$ is replaced with a smaller set $\mathcal{R} \subset \mathcal{S}$ of potential motif instances, i.e. k-mers that match with up to $2m$ mismatches every string in the input.

We next present two tree-based motif search algorithms that use selection (Algorithm 1) to improve search efficiency. We will evaluate both algorithms in Section 4.

3.1 Mismatch trie algorithms with selection

We outline motif search using selection and a single global tree in Algorithm 2. We use selection (Algorithm 1) to obtain reduced set $\mathcal{R}$ of potential motif instances (step 1). This set is then used in the tree search (step 2). The tree search follows depth first search strategy (as in mismatch trie [5]) with each node corresponding to a substring (prefix) formed by the characters selected from the alphabet set Σ. A search proceeds to the next level only if the set of input k-mers matching the current substring contains instances from the specified number of strings (e.g., all N strings). The overall complexity of this algorithm

is $O(\binom{k}{2m}knN + R \cdot V(k,m))$ and is an improvement over $O(knN \cdot V(k,m))$ complexity of the trie-based algorithms (e.g., [5, 11]).

Algorithm 2 Full tree algorithm with selection

Input: input set S of k-mers, occurrence threshold t
Output: motif set $\mathcal{M}$

```
1: Obtain reduced set R from S using Algorithm 1
2: full-tree-traverse(R,0,[])
3: function full-tree-traverse(S,depth,b)
4: if depth==k then
5:    output current k-length substring b as a motif
6: else
7:    depth = depth + 1
8:    for i = 1 to |Σ| do
9:       b[depth] = i
10:      S' = {a ∈ S: d(a,b) ≤ m}
11:      if #strings in S' ≥ t then
12:         full-tree-traverse(S', depth, b)
13:      end if
14:   end for
15: end if
```

3.2 Local tree algorithms with selection

Algorithm 3 outlines motif search using selection and local tree searches over sample neighborhood sets. The algorithm uses potential motif instances (set $\mathcal{R}$, selected in step 1) as starting points for neighborhood sets traversals (step 2) using depth first strategy, similar to the neighborhood search in voting algorithms [7] and PMS algorithms [10]. Each node in the tree (with the maximum depth m) corresponds to a particular k-length substring from the (k,m)-neighborhood. The node is expanded further only if the set of matching k-mers contains instances from the specified number of input strings (e.g., N strings). The overall complexity of this algorithm is $O(\binom{k}{2m}knN + R \cdot V(k,m)R)$. We note that this improves search complexity as we will show in the experimental section.

4 Experimental evaluation

We evaluate our algorithms on a synthetic benchmark motif finding task and real protein sequence data. We first test our algorithms on the planted motif problem commonly used as a benchmark for evaluating performance of motif finding algorithms [5, 10, 2]. We then illustrate our method on challenging biological sequence analysis problems.

Algorithm 3 Local tree search algorithm with selection

Input: input k-mer samples S, occurrence threshold t
Output: motif set $\mathcal{M}$

use Algorithm 1 to obtain reduced set $\mathcal{R}$ from $\mathcal{S}$
for all $a \in \mathcal{R}$ **do**
  local-tree(a, 0, 1, R)
end for
function local-tree(a, *depth*, *start*, S)
  if $depth == m$
    return
  end if
  depth = depth + 1
  for i=start to k
    $a' = a$ with ith position replaced by a character from Σ
    $S' = \{a \in S\text{: } d(a, a') - (m - depth) \leq m\}$
    if #strings in $S' \geq t$
      output a' as a motif if at distance of at most m from S
      local-tree(a', *depth*, $i + 1$, S')
    end if
  end for
end

4.1 Benchmark motif search problems

A planted motif problem is the task where synthetic motifs are injected in otherwise motif-less strings [2]. For this problem, we follow the standard setting used in previous studies [2, 5, 10] and synthesize $N = 20$ random strings of length $n = 600$ using iid, uniformly distributed symbols from an alphabet of size $|\Sigma|$. We then embed a copy (with up to m substitutions at random positions) of the motif at a random location in every string. The task is to identify motifs hidden in the input.

4.1.1 Full tree algorithms

We use various challenging instances of the planted motif problem and compare in Table 2 running time of the mismatch tree algorithm (MITRA) with the mismatch tree that uses candidate selection (Algorithm 2). All the running time measurements are obtained on a machine with a 3.0GHz CPU. We observe significant improvements in running time using our algorithm with candidate selection (for example, for the protein alphabet ($|\Sigma| = 20$) our algorithm improves running time by a factor of 10^3 on the (13,4) motif problem instance). For large alphabets and large k,m full trie traversal using entire input takes substantial amount of time and results in these cases are not reported. We also note that our algorithm significantly reduces running time compared to the state-of-the-art suffix tree algorithm RISOTTO [6]

which takes about 56, 1860, and 61956 seconds to complete on (9,2), (11,3), and (13,4) instances, respectively, using protein alphabet. We also show in Table 2 the number of k-mer instances $|\mathcal{R}|$ selected by the algorithm (out of $Nn = 20 \cdot 600 = 12000$ input samples). We observe that selection consistently results in much smaller sets compared to the entire set $\mathcal{S}$.

Table 2: Running time (s) on challenging instances of planted motif problems (n=600, N=20) using mismatch tree algorithms

$(k, m, \|\Sigma\|)$	MITRA	S-MITRA	$\|R\|$
(9,2,20)	4.73	0.4338	39
(9,2,50)	89.8	0.4359	39
(9,2,100)	266	0.4423	39
(11,3,20)	307	1.71	42
(11,3,50)	12296	1.76	42
(11,3,100)	-	1.73	42
(13,4,20)	9524	7.44	65
(13,4,50)	685015	5.26	38
(13,4,100)	-	5.27	38
(15,5,20)	-	120	97
(15,5,50)	-	883	90
(15,5,100)	-	318	65
(17,6,20)	-	4549	86
(17,6,50)	-	51088	85
(17,6,100)	-	130265	85

In Figure 3, we show the running time ratio (logarithmic scale) between the mismatch trie traversal (MITRA) algorithm and our algorithm with candidate selection (Algorithm 2) as a function of the alphabet set size. The running time is measured on $(13, 4)$ instances of the planted motif problem. For relatively small alphabet of size 20 our algorithm is about 10^3 times faster than the mismatch trie. The difference in running time increases with the size of the alphabet. We note that analysis of large alphabet sequences such as 3D protein structures may be required in difficult cases when almost no similarity can be observed at the nucleotide or amino acid sequence level.

In Figure 4 we illustrate efficiency of the candidate selection by our algorithm and show the ratio between the total number of the k-mers in the input and the number of k-mers selected from the input as potential motif instances. We observe that across different input sizes, selection reduces the sample size by a factor of $10^2 - 10^3$. Also, as expected from our theoretical analysis, we observe that our algorithm scales linearly with the input sequence length (Table 3).

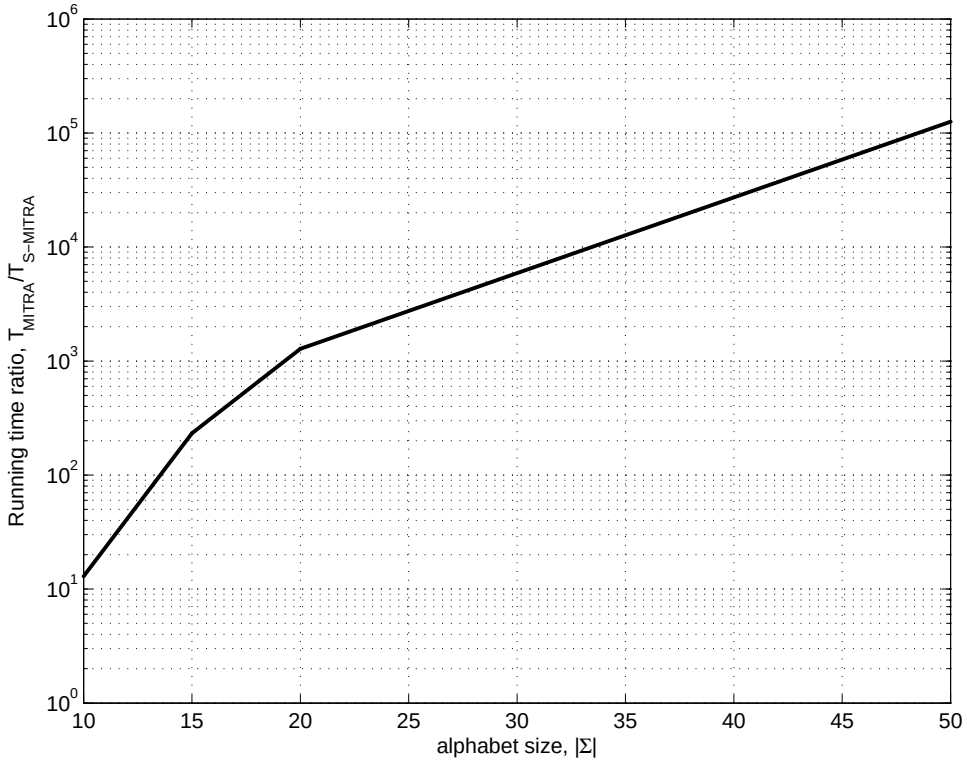

Figure 3: Running time ratio $T_{MITRA}/T_{S-MITRA}$ as a function of the alphabet size $|\Sigma|$ ($k = 13$, $m = 4$, $n = 600$, $N = 20$)

Table 3: Mismatch tree (S-MITRA) running time as a function of the sequence length n (protein alphabet, $|\Sigma| = 20$)

n	(13,4,20)	$\|R\|$	(15,5,20)	$\|R\|$
600	7.4	65	120	97
1000	13.7	76	144	74
2000	17.9	37	217	80
3000	36.4	91	258	94

4.1.2 Voting (local neighborhood tree) algorithms

Similar to the global tree algorithm in the previous section, we observe significant search efficiency improvements in the case of the local neighborhood tree search. As we can see from the results in Table 4, using our algorithm with candidate selection (Algorithm 3) significantly reduces running time compared to the standard voting algorithms that use entire input. For instance, we observe 20-100 times running time improvements on protein data ($|\Sigma| = 20$). Figure 5 displays running time ratio between the standard voting algorithm

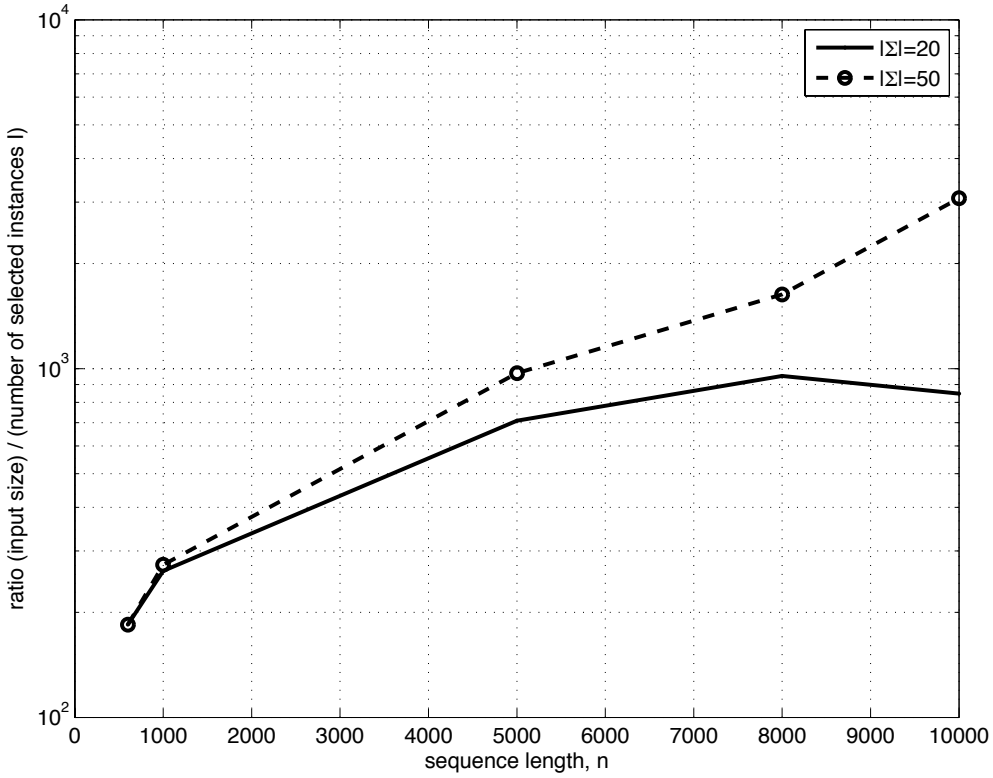

Figure 4: Ratio between the input size (nN) and the number of selected sample k-mers ($R = |\mathcal{R}|$) as a function of the input length and alphabet size (planted motif problem, $k = 13$, $m = 4$).

and our algorithm with selection using (13,4) problem instance. As evident from the figure, we observe 10-100 times improvements depending on the alphabet size.

4.2 Motif finding in real protein data

We also apply our algorithms to finding subtle sequence motifs in protein sequences, a challenging task due to the large alphabet size ($|\Sigma| = 20$) coupled with large k and m.

4.2.1 Lipocalin motifs

We first consider motifs in *lipocalins* which are topologically similar but have very diverse primary sequences. Using k-mer of length $k = 16$ with $m = 6$ mismatches, we are able to identify motifs containing 16 residues with the instance majority `DFD[IKLW]S[AKNR]FAGTWYE[ILMV]AK`, which agrees with the known reference motif [15]. Our algorithms take about 5 minutes to complete this task, while using the mismatch trie algorithm this task takes more than a day. We note that a large number of mismatches is critical for finding sequence motifs, while

Table 4: Running time (s) for challenging planted motif problem instances using voting algorithms

(k, m, Σ)	Voting	S-Voting
(9, 2, 20)	70.5	0.4656
(9, 2, 50)	161.5	0.4721
(9, 2, 100)	311.3	0.5053
(11, 3, 20)	84.1	1.6866
(11, 3, 50)	193.8	1.7282
(11, 3, 100)	375	1.8285
(13, 4, 20)	98.7	5.6239
(13, 4, 50)	227.4	4.7589
(13, 4, 100)	442.7	4.9562
(15, 5, 20)	118.5	13.5
(15, 5, 50)	312.5	17.2
(15, 5, 100)	714.3	35.9
(17, 6, 20)	152.9	46.3
(17, 6, 50)	355.0	84.1
(17, 6, 100)	816.0	257.8

smaller values of k, m do not result in motif identification.

4.2.2 Super-secondary structure sequence motifs

Using Cadherin and Immunoglobin superfamilies as an example, our algorithm finds sequence patterns that correspond to the supersecondary structure (SSS) motifs [16, 17], i.e. arrangements of the secondary structure units (loops, strands) that define structure of the proteins. In particular, in Cadherin superfamily we find long motifs of length 20 (using $m = 4$ mismatches) corresponding to the secondary structure units *strand 1 - loop - strand 2* (`VIPPISCPENE[KR]GPFPKNLV`) and *strand 3 - loop - strand 4* (`YSITGQGAD[KNQT]PPVGVFII`) (3D SSS motif [17]). Figure 6 shows identified motif positions within the sequences. Our algorithm finds 36 potential motif instances (out of 330 samples) and takes about 47 seconds (compared to about 600 seconds using the entire sample). In Immunoglobin superfamily (C1 set domains), we find a sequence motif of length 19 `SSVTLGCLVKGYFPEPVTV` which corresponds to *strand 2-loop-strand 3* secondary structure units (2E SSS motif).

5 Conclusions

We presented new deterministic and exhaustive algorithms for finding motifs, the common patterns in sequences. Our algorithms use fast input candidate selection to improve motif search efficiency by focusing on potential motif instances. Proposed algorithms reduce computational complexity of the current algorithms and demonstrate strong running time

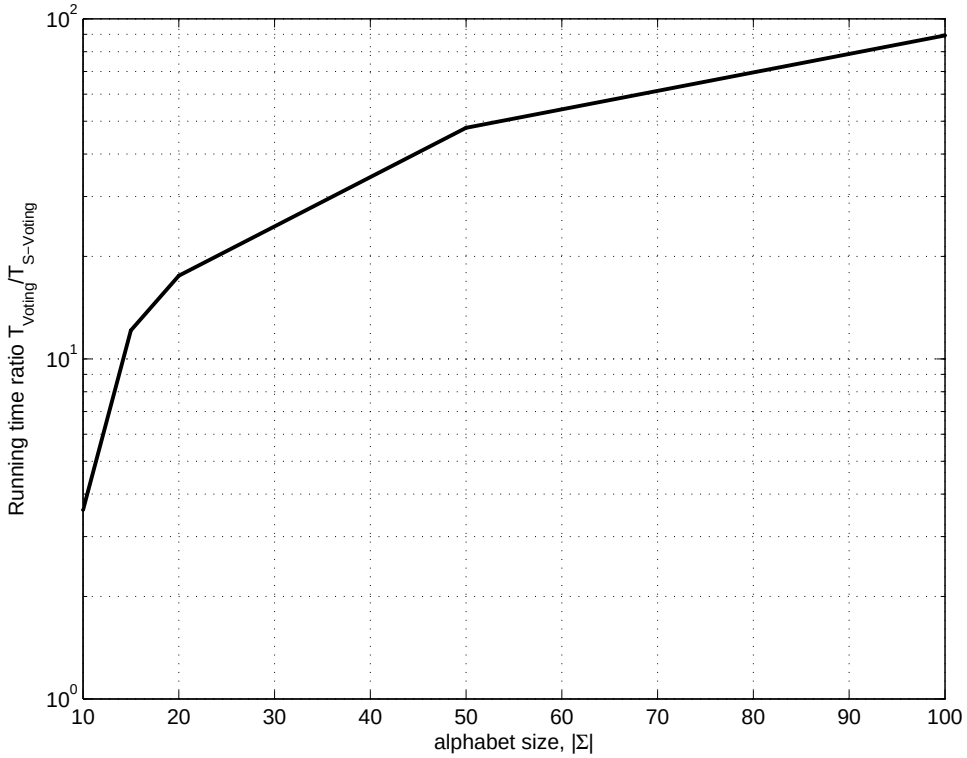

Figure 5: Running time ratio $T_{Voting}/T_{S-Voting}$ as a function of the alphabet size $|\Sigma|$ ($k = 13$, $m = 4$, $n = 600$, $N = 20$)

improvements, especially in an important case of sequences with large alphabet sizes (e.g., protein data). These improvements make exact combinatorial algorithms for finding motifs a practical alternative for general sequence analysis tasks. The proposed algorithms can be readily applied to other challenging problems in sequence analysis and mining.

References

[1] Eric P. Xing, Michael I. Jordan, Richard M. Karp, and Stuart Russell. A hierarchical Bayesian Markovian model for motifs in biopolymer sequences. In *In Proc. of Advances in Neural Information Processing Systems*, pages 200–3. MIT Press, 2003.

[2] Pavel A. Pevzner and Sing-Hoi Sze. Combinatorial approaches to finding subtle signals in dna sequences. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, pages 269–278. AAAI Press, 2000.

[3] Jean-Marc Fellous, Paul H. E. Tiesinga, Peter J. Thomas, and Terrence J. Sejnowski. Discovering Spike Patterns in Neuronal Responses. *J. Neurosci.*, 24(12):2989–3001, 2004.

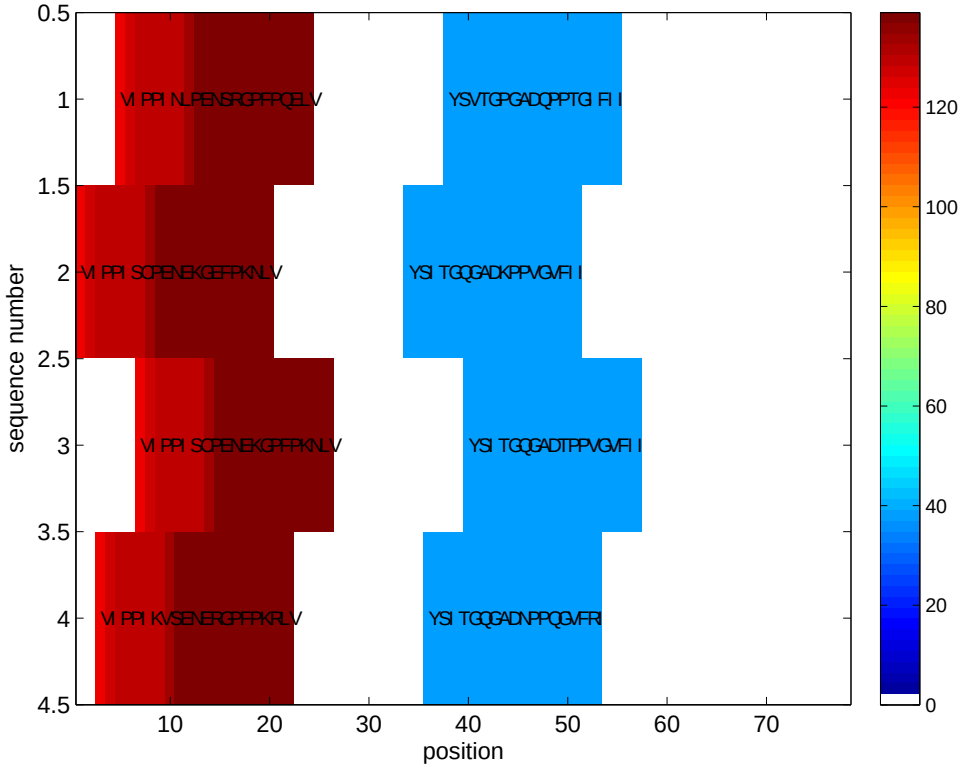

Figure 6: The sequence patterns for supersecondary structure motifs (Cadherin superfamily, $k = 20, m = 4, |\Sigma| = 20$)

[4] Nebojsa Jojic, Vladimir Jojic, Brendan Frey, Christopher Meek, and David Heckerman. Using "epitomes" to model genetic diversity: Rational design of HIV vaccine cocktails. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 587–594. MIT Press, Cambridge, MA, 2006.

[5] Eleazar Eskin and Pavel A. Pevzner. Finding composite regulatory patterns in DNA sequences. *Bioinformatics*, 18(suppl1):S354–363, 2002. http://www.ccls.columbia.edu/compbio/mitra/.

[6] Nadia Pisanti, Alexandra M. Carvalho, Laurent Marsan, and Marie-France Sagot. RISOTTO: Fast extraction of motifs with mismatches. In *LATIN*, pages 757–768, 2006.

[7] Francis Y. L. Chin and Henry C. M. Leung. Voting algorithms for discovering long motifs. In *APBC*, pages 261–271, 2005.

[8] M Tompa, N Li, T Bailey, G Church, and B De Moor. Assessing computational tools for the discovery of transcription factor binding sites. *Nature Biotechnology*, Jan 2005.

[9] Marie-France Sagot. Spelling approximate repeated or common motifs using a suffix tree. In *LATIN '98: Proceedings of the Third Latin American Symposium on Theoretical Informatics*, pages 374–390, London, UK, 1998. Springer-Verlag.

[10] Jaime Davila, Sudha Balla, and Sanguthevar Rajasekaran. Fast and practical algorithms for planted (l, d) motif search. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4(4):544–552, 2007.

[11] Patricia A. Evans and Andrew D. Smith. Toward optimal motif enumeration. In *WADS*, pages 47–58, 2003.

[12] Christina Leslie and Rui Kuang. Fast string kernels using inexact matching for protein sequences. *J. Mach. Learn. Res.*, 5:1435–1455, 2004.

[13] S. Rajasekaran, S. Balla, and C.-H. Huang. Exact algorithms for planted motif problems. *Journal of Computational Biology*, 12(8):1117–1128, 2005.

[14] Jaime Davila, Sudha Balla, and Sanguthevar Rajasekaran. Space and time efficient algorithms for planted motif search. In *International Conference on Computational Science (2)*, pages 822–829, 2006.

[15] CE Lawrence, SF Altschul, MS Boguski, JS Liu, AF Neuwald, and JC Wootton. Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment. *Science*, 262(5131):208–214, 1993.

[16] A. E. Kister, A. S. Fokas, T. S. Papatheodorou, and I. M. Gelfand. Strict rules determine arrangements of strands in sandwich proteins. *Proceedings of the National Academy of Sciences of the United States of America*, 103(11):4107–4110, 2006.

[17] Super-Secondary Structure Database. `http://binfs.umdnj.edu/sssdb/`.