Fast motif selection for biological sequences

Pavel Kuksa and Vladimir Pavlovic
Department of Computer Science
Rutgers University
Piscataway, NJ 08854
{*pkuksa,vladimir*}*@cs.rutgers.edu*

***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. 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 they exhibit significant running time improvements compared to the state-of-the-art approaches.**

***Keywords*-Tree searching, Algorithms, Sequences**

I. 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 (see e.g., [2]) for a given 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).

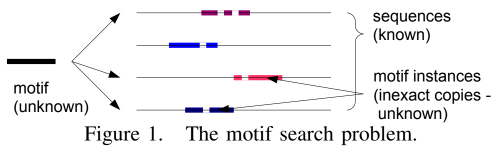

Figure 1. The motif search problem.

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 III) 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.

II. 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 motif instances), 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 $V(k, m) = \sum_{i=0}^{m} \binom{k}{i} (|\Sigma| - 1)^i$ of the k-mer *neighborhood* which consists of all k-length substrings at Hamming distance of at most m from a k-mer, which can lead to high computational complexity as shown in Table I.

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

Table I
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

length-k substring observed in the input gives one 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. 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]. The PMSP algorithm 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 III. We propose two motif search algorithms with selection that use for search a single global tree (Section III-A) or use local tree searches (Section III-B).

III. 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 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 $\mathcal{S}$).

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).

```
Algorithm 1 Selection algorithm
Input: set $\mathcal{S}$ of $k$-mers with associated sequence index $L$,
    distance parameter $d$ ($d = 2m$)
Output: set $\mathcal{R}$ of $k$-mers at distance of at most $d$ from each
    input string
    1. Iteratively 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.
```

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 in Algorithm 1. As we will see in the experiments (Section IV), 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

$$E[\mathcal{R}_B] = |\Sigma|^k (1 - (1 - p_{k,2m})^n)^N$$

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 $p_{13,8} = 2.9 \cdot 10^{-4}$. 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 IV.

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 IV.

A. *Mismatch trie algorithms with selection*

We use selection (Algorithm 1) to obtain reduced set $\mathcal{R}$ of potential motif instances. This set is then used in the tree search. 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]).

B. *Local tree algorithms with selection*

The algorithm uses potential motif instances (set $\mathcal{R}$), as starting points for neighborhood sets traversals 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.

IV. Experimental evaluation

We test our algorithms on the planted motif problem commonly used as a benchmark for evaluating performance of motif finding algorithms [5], [10], [2].

A. *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.

1) Full tree algorithms: We use various challenging instances of the planted motif problem and compare in Table II running time of the mismatch tree algorithm (MITRA) with the mismatch tree (S-MITRA) that uses candidate selection (Algorithm 1). 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 relatively small protein alphabet ($|\Sigma| = 20$) our algorithm improves running time by a factor of 10^3 on the (13,4) motif problem instance compared to the mismatch trie. The difference in running time increases with the size of the alphabet. Large alphabets can, for instance, arise when encoding the 3D protein structure, a necessity in cases when sequences share little similarity at primary level. We also show in Table II 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 $\mathcal{S}$.

In Figure 2 we illustrate efficiency of the candidate selection by Algorithm 1 and show the ratio between the

Table II

RUNNING TIME (S) ON CHALLENGING INSTANCES OF PLANTED MOTIF PROBLEMS (n=600, N=20)

$(k, m, \|\Sigma\|)$	MITRA	S-MITRA	Voting	S-Voting	$\|R\|$
(9,2,20)	4.73	0.4338	70.5	0.4656	39
(9,2,50)	89.8	0.4359	161.5	0.4721	39
(9,2,100)	266	0.4423	311.3	0.5053	39
(11,3,20)	307	1.71	84.1	1.6866	42
(11,3,50)	12296	1.76	193.8	1.7282	42
(11,3,100)	-	1.73	375	1.8285	42
(13,4,20)	9524	7.44	98.7	5.6239	65
(13,4,50)	685015	5.26	227.4	4.7589	38
(13,4,100)	-	5.27	442.7	4.9562	38
(15,5,20)	-	120	118.5	13.5	97
(15,5,50)	-	883	312.5	17.2	90
(15,5,100)	-	318	714.3	35.9	65
(17,6,20)	-	4549	152.9	46.3	86
(17,6,50)	-	51088	355	84.1	85
(17,6,100)	-	130265	816	257.8	85

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 $10^2 - 10^3$. As expected from our theoretical analysis, we also observe that our algorithm scales linearly with the input sequence length (Table III).

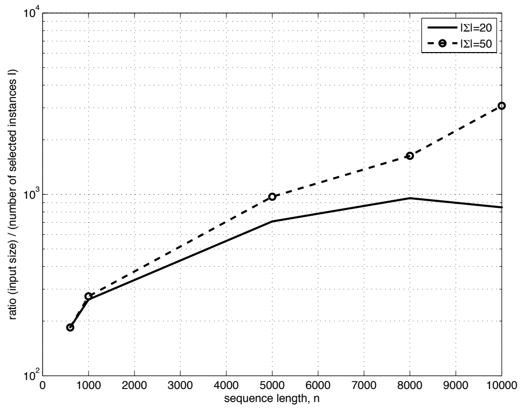

Figure 2. Ratio between the input size (nN) and the number of selected sample k-mers ($R = |\mathcal{R}|$) as a function of the input length and alphabet size (planted motif problem, $k = 13$, $m = 4$).

Table III

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

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. Results in Table II suggest that using our algorithm (S-Voting) with candidate selection (Algorithm 1) significantly reduces running time compared to the standard voting algorithms that use the entire input. For instance, we observe a factor 20-100 running time improvements on protein data ($|\Sigma| = 20$).

V. 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 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] E. P. Xing, M. I. Jordan, R. M. Karp, and S. Russell, "A hierarchical Bayesian Markovian model for motifs in biopolymer sequences," in *Advances in Neural Information Processing Systems*. MIT Press, 2003, pp. 200–3.

[2] P. A. Pevzner and S.-H. Sze, "Combinatorial approaches to finding subtle signals in dna sequences," in *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*. AAAI Press, 2000, pp. 269–278.

[3] J.-M. Fellous, P. H. E. Tiesinga, P. J. Thomas, and T. J. Sejnowski, "Discovering Spike Patterns in Neuronal Responses," *J. Neurosci.*, vol. 24, no. 12, pp. 2989–3001, 2004.

[4] N. Jojic, V. Jojic, B. Frey, C. Meek, and D. Heckerman, "Using "epitomes" to model genetic diversity: Rational design of HIV vaccine cocktails," in *Advances in Neural Information Processing Systems 18*, Y. Weiss, B. Schölkopf, and J. Platt, Eds. Cambridge, MA: MIT Press, 2006, pp. 587–594.

[5] E. Eskin and P. A. Pevzner, "Finding composite regulatory patterns in DNA sequences," *Bioinformatics*, vol. 18, no. suppl1, pp. S354–363, 2002.

[6] N. Pisanti, A. M. Carvalho, L. Marsan, and M.-F. Sagot, "RISOTTO: Fast extraction of motifs with mismatches," in *LATIN*, 2006, pp. 757–768.

[7] F. Y. L. Chin and H. C. M. Leung, "Voting algorithms for discovering long motifs," in *APBC*, 2005, pp. 261–271.

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

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

[10] J. Davila, S. Balla, and S. Rajasekaran, "Fast and practical algorithms for planted (l, d) motif search," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 4, no. 4, pp. 544–552, 2007.

[11] P. A. Evans and A. D. Smith, "Toward optimal motif enumeration," in *WADS*, 2003, pp. 47–58.

[12] C. Lawrence, S. Altschul, M. Boguski, J. Liu, A. Neuwald, and J. Wootton, "Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment," *Science*, vol. 262, no. 5131, pp. 208–214, 1993.

[13] 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*, vol. 103, no. 11, pp. 4107–4110, 2006.

[14] "Super-Secondary Structure Database," http://binfs.umdnj.edu/sssdb/.