**DIMACS Technical Report 2009-15**
**June 2009**

# Efficient discovery of common patterns in sequences over large alphabets

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 data modeled as strings or sequences. In particular, we present a new deterministic algorithm for finding patterns that are embedded as exact or inexact instances in all or most of the input strings. The proposed algorithm (1) improves search efficiency compared to existing algorithms, and (2) scales well with the size of alphabet. Our algorithm is several orders of magnitude faster than existing deterministic algorithms for common pattern identification. We evaluate our algorithm on benchmark motif finding problems and real applications in biological sequence analysis and show that our algorithm maintains predictive performance with significant running time improvements.

# 1 Introduction

Finding motifs or repeated patterns in data is of wide scientific interest [1, 2, 3, 4]. 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. Discovery of motifs in music sequences, text, or time series data is a fundamental, general means of summarizing, mining and understanding large volumes of data. The motif search problem abstracts many important problems in analysis of data modeled as strings or sequences, where motifs are, for example, biologically important patterns.

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 $\Sigma$. The solution for the $(k, m, \Sigma, N)$-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 $a \in \mathcal{M}$, $|a| = k$, is at Hamming distance at most $m$ from all (or almost all) strings $s \in \mathcal{S}$.

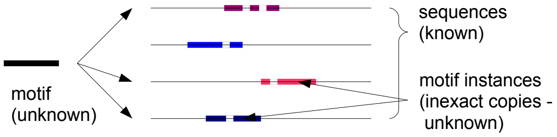


Figure 1: The motif search problem.

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 by the exponential complexity of such search. As a consequence, the problem quickly becomes intractable for even moderately long motifs and small alphabets. We present a new deterministic algorithm for finding common patterns with the search complexity that scales well with the size of the alphabet.

Compared to existing algorithms in this class (e.g. [5, 6]) that have strong dependency on the alphabet size and work with small-alphabet input, our algorithms significantly improve search efficiency in the important case of large-alphabet inputs (e.g. protein alphabet, extended DNA alphabet, word alphabets in text, etc.)

As we show in the experiments, using both synthetic and real data, our algorithms are orders-of-magnitude faster than existing state-of-the-art motif deterministic search algorithms, especially on large-alphabet inputs. This result extends applicability of the motif search algorithms to more complex problems requiring analysis of data modeled as strings over large alphabets.

## 2 Related work

The problem of motif discovery has been tackled extensively over the past two decades [7]. Within the class of exhaustive methods, a number of approaches have been proposed, including graph methods (WINNOWER) [2], explicit trie traversal (MITRA) [5], explicit mapping (Voting algorithms) [8], 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 *neighborhood* of a $k$-mer, i.e. the number of $k$-mer sequences at Hamming distance of at most $m$ from it. This size, $V(k,m) = \sum_{i=0}^{m} \binom{k}{i}(|\Sigma|-1)^i$, depends on the alphabet size, and 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 [8] | $O(nNV(k,m))$ | $O(nV(m,k))$ |
| RISOTTO [6] | $O(nN^2V(k,m))$ | $O(nN^2)$ |
| PMS [10] | $O(n^2NV(k,m))$ | $O(n^2N)$ |

Explicit mapping (voting) algorithms proposed in [8] 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 and requires at least $O(k^{m+1}|\Sigma|^m nN)$ space. The large space requirement of the algorithm restricts its usage to small values of $k$ and $m$, as well as to small alphabet size $|\Sigma|$.

One of the most efficient exact algorithms for motif search, the mismatch tree (MITRA) algorithm [5], uses efficient trie traversal to find a set of motifs in the input strings. Under a trie-based computation 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 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 how the algorithm complexity depends on the alphabet size since during a trie traversal possible substitutions are drawn from the alphabet set $\Sigma$ explicitly; consequently, to control the complexity of the algorithm we need to restrict the number of allowed mismatches $m$ and length $k$, as well as the alphabet size.

Another class of efficient 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 solution 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 algorithm [8]. 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. We note that *explicit enumeration* is employed by all above-mentioned algorithms.

In contrast to existing exact exhaustive algorithms, we approach the problem of motif finding by performing an efficient search over patterns with wildcards. As a consequence, the proposed method's complexity becomes independent of the alphabet size.

# 3 Combinatorial algorithm for motif search

In this section, we develop an efficient combinatorial algorithm for motif finding with the search complexity independent of the size of the alphabet $|\Sigma|$. The algorithm begins by finding a set of candidate motifs, followed by the construction of the intersections of those candidates' neighborhoods, the sequences that are at most $m$ symbols apart from each candidate pair. In a crucial departure from other approaches, this set is efficiently represented using *stems*, or patterns with wildcards. The number of the stems does not depend on the alphabet size and is a function of the motif length ($k$), the number of mismatches ($m$) and the Hamming distance between $k$-mers. Patterns common to all (or almost all) input strings are then found by pruning the stems that do not satisfy the motif property (i.e., do not occur in all input strings).

The main idea of our approach is to construct a candidate set $\mathcal{C}$ which includes all motifs $\mathcal{M}$ plus some non-motifs, i.e. $\mathcal{M} \subseteq \mathcal{C}$, and then efficiently select true motifs from the candidate set. Given $\mathcal{C}$, the complexity of motif finding is then proportional to its size: the motifs can be extracted from $\mathcal{C}$ by checking each candidate against the motif property, a task we accomplish using $\binom{k}{m}$ rounds of counting sort in Algorithm 2. To generate $\mathcal{C}$, we collect the sets of stems which characterize the common neighbors of the pairs of $k$-mers $(a, b)$ in the input. We call these sets the *stem sets*, $\mathcal{H}(a, b)$. Finding each $\mathcal{H}(a, b)$ is independent of the alphabet size and is accomplished in Algorithm 3. To further reduce the complexity, we construct the stem sets only for potential motif instances $\mathcal{I}$, those $k$-mers that are at Hamming distance of at most $2m$ from every input string. We find $\mathcal{I}$ using $\binom{k}{2m}$ rounds of counting sort (Algorithm 2). We outline our motif search algorithm below:

This algorithm uses as its main sub-algorithm (in step 2) a procedure that finds the intersection of $k$-mer neighborhoods for any pair of the $k$-mers $a$, $b$. This intersection finding algorithm is described in Section 3.2. We describe selection and pruning steps (steps 1 and 3) in Section 3.1.

The overall complexity of the algorithm is $O(\binom{k}{2m}nN + \binom{k}{m}HI^2)$, where $H$ is the maximum size of $\mathcal{H}(a, b)$, and $I$ is the size of $\mathcal{I}$, the number of $k$-mers used to construct the candidate set $\mathcal{C}$. The important fact that makes our algorithm efficient in practice is that typically

**Algorithm 1** Motif search algorithm

```
1. Use multiple rounds of counting sort to iterate over input strings and construct a set
   of potential motif instances $\mathcal{I}$, $k$-mers that are at Hamming distance of at most $2m$ from
   each string (Algorithm 2).
2. Construct candidate set $\mathcal{C}$ by building stem sets $\mathcal{H}(a,b)$ for $k$-mer pairs in $\mathcal{I}$ (Algo-
   rithm 3)
3. Prune all stems from $\mathcal{C}$ that do not satisfy motif property using $\binom{k}{m}$ rounds of counting
   sort (Algorithm 2, Section 3.1.1)
4. Output remaining stems as motifs.
```

$I \ll \min(nN, |\Sigma|^k)$ and $H \ll V(k,m)$, particularly for large alphabets. We demonstrate this in our experimental results and provide an expected-size analysis in Section 3.1.

## 3.1 Selection algorithm

A necessary condition for a group of $k$-mers to have a shared, common neighbor (motif) is that the Hamming distance between any pair of patterns cannot exceed $2m$.

We will use this condition to select $k$-mers from input that are potential motif instances and place them in set $\mathcal{I}$. 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. To select the valid $k$-mers, we use multiple rounds of count sort by removing iteratively $2m$ out of $k$ positions and sorting the resulting set 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 same $k$-mers together. Using a simple linear scan over the sorted list of all input $k$-mers, we can find the set of potential motifs and construct $\mathcal{I}$. This algorithm is outlined below:

**Algorithm 2** Selection algorithm

```
Require: set of $k$-mers with associated sequence index, distance parameter $d$
Ensure: set 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.
4. Output the $k$-mers that appear in every input string.
```

As we will see in the experiments (Section 5), the selection step significantly reduces the number of $k$-mer instances considered by the algorithm and improves search efficiency. The number of selected $k$-mers, i.e. the size of $\mathcal{I}$, is small, especially for large-alphabet inputs. 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

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

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 $\mathcal{I}$, is

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

Since $t$ and $p$ are typically small, for small $pn$, $E[\mathcal{I}] \ll nN$, the number of $k$-mers in the input. In the protein example above the expected size of $\mathcal{I}$ is about $1 + 3 + 8 = 12$ for $t = 1$, which is orders of magnitude smaller than $nN = 12000$, signifying the importance of creating $\mathcal{I}$ first. This is empirically demonstrated in Section 5.

### 3.1.1 Pruning using selection

The sorting approach of Algorithm 2 is also used to select patterns satisfying the motif property from the candidates $\mathcal{C}$ (Step 3 in main Algorithm 1). The pruning step is based on verifying the motif property (i.e. whether given patterns match all input sequences with up to $m$ mismatches) and can be accomplished using $\binom{k}{m}$ rounds of counting sort.

## 3.2 Motif generation

In what follows, we describe an efficient algorithm that finds the set of *stems* that represent the set of $k$-mers shared by a pair of $k$-mers $a, b$. This process is used to create set $\mathcal{C}$ from potential instances $\mathcal{I}$, which is subsequently pruned to yield the true motif instances.

The number of $k$-mers in the common neighborhood of any two particular $k$-mers $a, b$ assumes a fixed set values depending on the Hamming distance $d(a, b)$ between $k$-mers [15], for given values of $|\Sigma|$, $k$, and $m$. We want to represent the shared $k$-mers in this intersection using a set of *stems*, patterns with wildcards. However, the number of stems will not depend on the alphabet size $|\Sigma|$.

To find all stems shared by $k$-mers $a$,$b$, consider two sets of positions: *mismatch region* in which $a$ and $b$ disagree and *match region* in which $a$ and $b$ agree. We consider two cases depending on the number of mismatch positions (i.e. Hamming distance between $a, b$). In the first case, the distance $d(a, b)$ is at most $m$, the maximum number of mismatches allowed. In the second case, the distance $d(a, b)$ exceeds $m$. When $d(a, b) \leq m$, wildcard characters can appear both inside and outside of the mismatch region. When $d(a, b) > m$, wildcard characters can appear only inside the mismatch regions. Consider for example, the case of $d(a, b) = 0$ and $m = 1$. In this case, the set of stems is the set of patterns with 1 wildcard at each of the possible $k$ positions (with the remaining positions as in $a$) plus one stem with 0 wildcards. When $m = 2$, and $d(a, b) = 1$, the set of stems will include patterns with 0 or 1 wildcard in $k - d$ positions and 0 or 1 wildcards in the remaining $d = 1$ positions. For example, for the pair $(tgt, tgc)$ the corresponding patterns with wildcards are $tg?$, $t??$, $?g?$, $t?c$, and $?gc$, where ? denotes a wildcard.

We outline algorithm for finding set of stems for the intersection of the $k$-mer neighborhoods in Algorithm 3 below.

**Algorithm 3** Stem generation (independent of the alphabet size $|\Sigma|$)

```
Require: pair of $k$-mers $a$, $b$
Ensure: set of stems (patterns with wildcards)
  shared by $a$ and $b$
  if if $d(a, b) \leq m$ then
    Set stem = $a$
    Set $i = 0 \dots d$ positions in the mismatch region of the stem
    as in $b$
    Place $j_1 = 0 \dots d - i$ wildcards inside the mismatch region
    Place $j_2 = 0 \dots m - \max(d - i, j_1 + i)$ wildcards outside the mismatch region
  end if
  if $d(a, b) > m$ then
    Set stem = $a$
    Fix $i = d - m \dots m$ positions in the mismatch region of the current stem as in $b$
    Place $j = 0 \dots m - i$ wild-cards in the remaining $d - i$ positions in the mismatch region
  end if
  Output resulting stems (patterns with wildcards)
```

The number of stems generated by the algorithm is

$$0 \leq d \leq m : \sum_{i=0}^{d} \sum_{j_1=0}^{d-i} \sum_{j_2=0}^{\min(m-d+i, m-i-j_1)} \binom{d}{i} \binom{d-i}{j_1} \binom{k-d}{j_2}$$

$$2m \geq d > m : \sum_{i=d-m}^{m} \sum_{j=0}^{m-i} \binom{d}{i} \binom{d-i}{j}$$

The number of stems that describe all the explicit $k$-mers shared between $a, b$ does not depend on the alphabet size. The complexity of the stemming algorithm is proportional to the number of stems generated. The maximum number of stems $H$ is $O(\sum_{i=0}^{2m} \binom{k}{i})$ for typical values of

$m < k/2$. We use Algorithm 3 for every pair of $k$-mers in $\mathcal{I}$ (in step 2) to construct $\mathcal{C}$ as outlined in the main algorithm.

### 3.2.1 Algorithm analysis

The complexity of the selection step 1 for constructing $\mathcal{I}$ is $O(\binom{k}{2m})nN$ and does not depend on the alphabet size $|\Sigma|$. Steps 2 and and 3 have the complexity $O(\binom{k}{m}HI^2)$ and again do not depend on $|\Sigma|$. As a consequence, the three-step procedure gives us an efficient,

alphabet-independent motif search algorithm that outputs all motifs embedded in the input $\mathcal{S}$. Our experiments will next demonstrate that this allows efficient exploitation of sparsity of typical solutions—we explore only a small portion of the motif space by focusing (using Algorithm 2) only on the support samples that are potential instances of the motifs. This results in significant reductions in running times, especially for large-alphabet inputs, i.e. the cases difficult for the current exact motif finding algorithms.

# 4 Extensions

Our proposed framework can be used to reduce search complexity for a wide class of exact search-based motif finding algorithms.

Existing exhaustive algorithms typically (e.g. [5, 8, 10]) use the entire input (i.e. all the $k$-mers in the input) and find motif by essentially exploring neighborhoods of every $k$-mer in the input. Their search complexity can be drastically improved by using a *reduced* set of $k$-mers instead of all input samples. This reduced set of $k$-mers can be obtained using our linear time selection algorithm (Algorithm 2, Section 3.1). Using reduced set of $k$-mers, the actual search complexity after the selection step becomes sublinear in the input size (since the number of selected $k$-mers $I = |\mathcal{I}|$ is much smaller than input length $O(nN)$). For instance, the search complexity of the trie-based algorithms (eg. [5]) can be reduced to $O(\binom{k}{m}knN + IV(k,m))$ 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 ($O(knN)$) is replaced with a smaller set $\mathcal{I}$ of $k$-mers that match with up to $2m$ mismatches every string in the input.

# 5 Experimental evaluation

We evaluate our algorithms on a synthetic benchmark motif finding task and real 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 two challenging biological sequence analysis problems.

## 5.1 Planted motif problem

A planted motif problem is the task where synthetic motifs are injected in otherwise motif-less strings. 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 a motif at a random location in every string. The task is then to identify motifs hidden in the input. As we can see from the results in Table 2 and Table 3, our algorithms show significant reduction in running times compared to state-of-the-art methods, especially for large-$|\Sigma|$ inputs (Table 3). For large alphabets and large $k$,$m$ trie traversal takes substantial amount of time and results in these cases are not reported. In Figure 2(a), we show the running time ratio (logarithmic scale) between the mismatch trie traversal (MITRA) algorithm and our algorithm 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 $10^4$ times faster than the mismatch trie. The difference in running time increases with the size of the alphabet.

Figure 2(b) shows efficiency of the selection (step 1 in the algorithm) as a ratio between the input size and the number of the selected samples ($k$-mers) $|\mathcal{I}|$. We observe that across different input sizes selection reduces the number of samples by a factor of about $10^3$. The observed number of selected samples $I = |\mathcal{I}|$ agrees with the theoretical estimates (e.g. in the case of $|\Sigma|$=50, n=5000, N=20, we expect about 52 $k$-mers to selected, and the observed size of $\mathcal{I}$ is 103 $k$-mers). The results agree with our quantitative analysis in Section 3.1. For small $np_{k,2m}$ the planted motif terms dominate the expected size of $\mathcal{I}$. For large $np_{k,2m}$ (large $n$ and small $|\Sigma|$) the number of matches by chance increases and can even result in the decrease exhibited in the $|\Sigma| = 20$ case for $n > 8000$ when $E[\mathcal{I}_B]$ increases faster than $nN$.

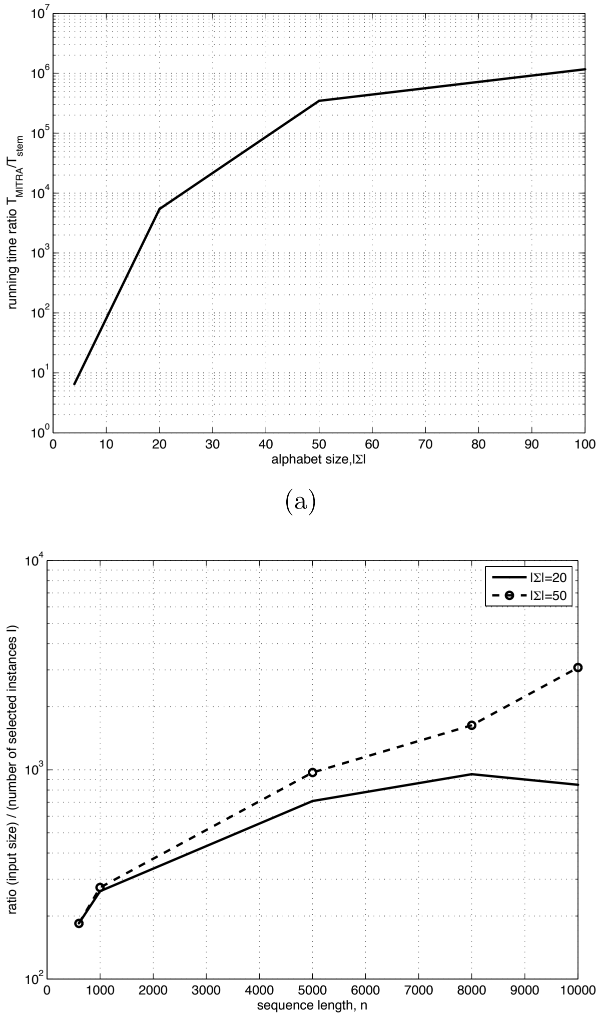
Our algorithm also allows one to solve more challenging motif finding instances for longer motifs and larger number of mismatches (e.g. (19,7) instance in Table 2). This is an important result since motif discovery in real data often requires analysis of potentially very long motifs with many substitutions.

Table 2: Running time comparison on the challenging instances of the planted motif problem (DNA, $|\Sigma| = 4$, $N = 20$ sequences of length $n = 600$). Problem instances are denoted by $(k, m, |\Sigma|)$, where $k$ is the length of the implanted motif, and $m$ is the number of mismatches.

| | Instances | | | | | |
|---|---|---|---|---|---|---|
| Algorithm | (9,2,4) | (11,3,4) | (13,4,4) | (15,5,4) | (17,6,4) | (19,7,4) |
| Stemming | 0.95 | 8.8 | 31 | 187 | 1462 | 8397 |
| MITRA [5] | 0.89 | 17.9 | 203 | 1835 | 4012 | n/a |
| PMSPrune [10] | 0.99 | 10.4 | 103 | 858 | 7743 | 81010 |
| RISOTTO [6] | 1.64 | 24.6 | 291 | 2974 | 29792 | n/a |

## 5.2 Finding motifs in real DNA and protein samples

We next illustrate the algorithm on several challenging applications in a biological domain ranging from identifying DNA binding sites to finding subtle sequence motifs which determine topology of a protein at secondary or super-secondary level.



(a)

(b)

Figure 2: (a) Running time ratio ($T_{MITRA}/T_{stem}$) as a function of the alphabet size (planted motif problem, $k = 13$, $m = 4$). Note logarithmic scale. (b) Ratio between input size ($nN$) and the number of selected samples $k$-mers ($I = |\mathcal{I}|$) as a function of the input length and alphabet size (planted motif problem, $k = 13$, $m = 4$). Note logarithmic scale.

Table 3: Running time, in seconds, on large-$|\Sigma|$ inputs. $(k, m)$ instances denote implanted motifs of length $k$ with up to $m$ substitutions.

| $\|\Sigma\|$ | (9,2) | | (11,3) | | (13,4) | | (15,5) | |
|---|---|---|---|---|---|---|---|---|
| | MITRA | Stemming | MITRA | Stemming | MITRA | Stemming | MITRA | Stemming |
| 20 | 8.39 | 0.637 | 1032.17 | 1.07 | 28905 | 5.247 | n/a | 12.31 |
| 50 | 89.82 | 0.633 | 12295.73 | 0.963 | 685015 | 2.244 | n/a | 11.92 |
| 100 | 265.94 | 0.645 | n/a | 0.967 | > 1 month | 2.227 | n/a | 11.86 |

## 5.3 Identifying CRP binding sites

We use dataset with experimentally confirmed E. coli CRP binding sites [2] from [16]. In this problem, there are 18 DNA sequences of length 105 with CRP binding sites at unknown positions within sequences. The task is to identify those sequence motif corresponding to CRP binding site and the binding site positions within sequences.

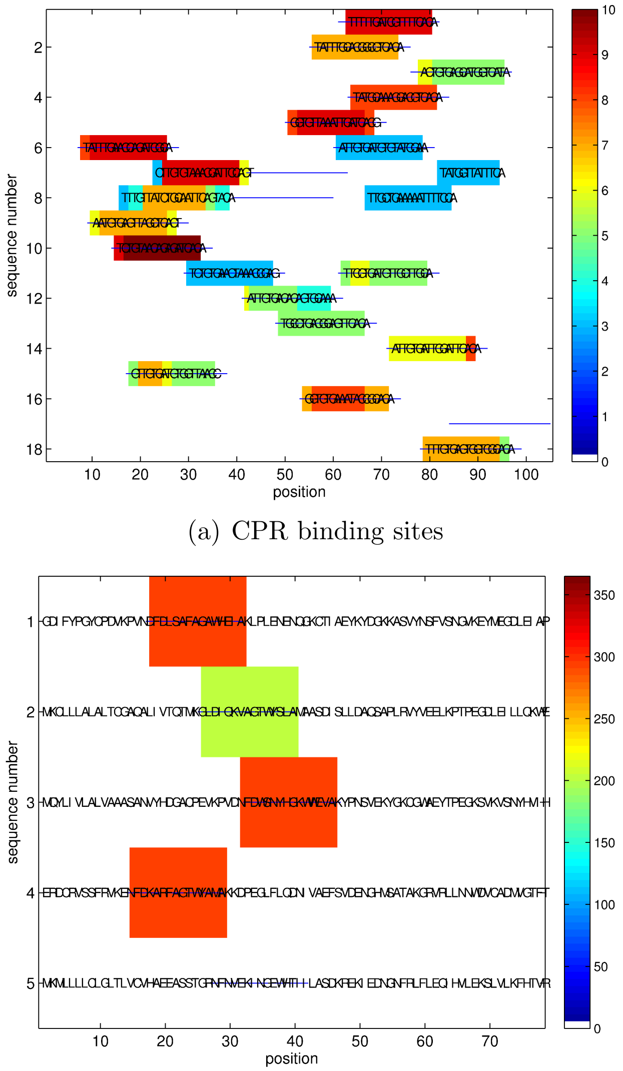
We use relatively long $k$-mers of length $k = 18$, with a large number of mismatches ($m = 7$) from a given set of 18 DNA sequences ($|\Sigma| = 4$).

Figure 3(a) illustrates motifs found by the algorithm. In the figure, blue horizontal lines indicate true (confirmed) locations of the binding sites. Colors in the figure indicate importance of positions as measured by the number of hits between the found motif patterns and the sequences. The set of discovered locations agrees with the set of experimentally confirmed primary positions. The discovered motifs correspond to instances of the reference consensus motif `TGTGAnnnnnnTCACA` [17, 16]. Because of large $k$ and $m$ we observe running time improvements similar to the benchmark planted motif problems: the running time of our algorithm is about 6 minutes, while the mismatch trie traversal requires about 12 times as long (4489 seconds). Allowing a large number of mismatches ($m = 7$) in this case is critical for the motif prediction performance because fewer mismatches do not lead to successful identification of the binding sites.

## 5.4 Protein motif finding

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

We focus on motifs in *lipocalins* which are topologically similar but have very diverse primary sequences. Using $k$-mer of length $k = 15$ with $m = 7$ mismatches, we able to identify motifs containing 15 residues with the instance majority `FD[IKLW]S[AKNR]FAGTWYE[ILMV]AK`, which agrees with the known reference motif [18]. Our algorithm takes about 5 minutes to complete this task, while using the mismatch trie algorithm this task takes more than a day. Figure 3(b) shows identified motif positions within the sequences, with blue lines indicating true motif locations. Most of the true motif locations are recovered with the exception of sequence 5 which contains a very weak motif instance. As in the case of the DNA, a large number of mismatches is critical for finding motifs with small $k, m$ leading to many misses.



(a) CPR binding sites

(b) Lipocalin motifs

Figure 3: (a) Recognition of CRP binding sites ($k = 18, m = 7, |\Sigma| = 4$). (b) Lipocalin motifs ($k = 15, m = 7, |\Sigma| = 20$).

# 6 Conclusions

We presented a new deterministic and exhaustive algorithm for finding motifs, the common patterns in sequences modeled as discrete-valued strings. Our algorithm reduces computational complexity of the current algorithms and demonstrate strong running time improvements in domains with large alphabet sizes. These improvements make combinatorial algorithms for finding motifs a practical alternative for general sequence analysis tasks. The proposed algorithms could be applied to other challenging problems in sequence analysis and mining, potentially characterized by large alphabets, such as finding genre- or artist- specific patterns in music data, or motion patterns in video/image sequences.

# 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.

[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] 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.

[8] Francis Y. L. Chin and Henry C. M. Leung. Voting algorithms for discovering long motifs. In *APBC*, pages 261–271, 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] Pavel Kuksa, Pai-Hsi Huang, and Vladimir Pavlovic. Scalable algorithms for string kernels with inexact matching. In *NIPS*, pages 881–888, 2008.

[16] G D Stormo and G W Hartzell. Identifying protein-binding sites from unaligned DNA fragments. *Proceedings of the National Academy of Sciences of the United States of America*, 86(4):1183–1187, 1989.

[17] CE Lawrence and AA Reilly. An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences. *Proteins*, 7(1):41–51, 1990.

[18] 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.