

IDENTIFICATION OF SPACED REGULATORY SITES VIA SUBMOTIF MODELING

E.WIJAYA AND R. KANAGASABAI

*Institute for Infocomm Research,
21 Heng Mui Keng Terrace,
Singapore 119613
E-mail: kanagasa@i2r.a-star.edu.sg*

In this paper we propose a novel approach for identification of generic motifs in an integrated manner by introducing the notion of submotifs. We formulate the motif finding problem as a constrained submotif pattern mining and present an algorithm called SPACE for identifying motifs that may contain spacers. When spacers are present, we show that the algorithm can identify motifs where 1) the spacers may be of varying lengths, 2) the number of motif segments may be unknown, and 3) the lengths of motif segments may be unknown. We perform rigorous experiments with the Motif Assessment Benchmarks by Tompa et al., and observe that our algorithm overall is able to outperform all popular algorithms tested so far, with significant improvements on sensitivity and specificity.

1. Introduction

The availability of vast genomic sequences from several organisms provides us with a rich opportunity for advancing our knowledge about biological systems. However, regulatory networks in most of these sequenced genomes is largely unexplored. One of the major challenges facing biologists today is to understand the regulatory mechanism of genes. This challenge includes detection of transcription factor binding sites involved in regulation and discovery of regulatory networks.

The problem of *de novo* identification of transcription factor binding site motifs has been widely studied and a number of motif-finding algorithms have been proposed. Most of these algorithms can be classified as follows:

- profile-based methods that addresses motif finding by learning a matrix (profile) model that describes the binding sites, e.g. Gibbs sampler [13], MotifSampler [20], SeSiMCMC [6], MEME [2], NestedMICA [11], Improbizer [1].
- consensus-based methods that enumerate oligos of (or upto) a given length to find strings that appear in many sequences and use statistical measures to rank them, e.g. MULTIPROFILER [12], Weeder [17], MITRA [5], TEREISIAS [18], Gemoda [10], CMF [24].
- hybrid methods that use a combination of the above two methods e.g. [8].

In spite of the availability of these methods, motif-finding (in general) continues to be a difficult problem because of challenges like complex motif structure, presence of weak signals and scalability to the whole genome. For example, recently Tompa [21] conducted

an assessment of 13 popular motif discovery algorithms over 56 datasets drawn from *H. Sapiens*, *M. Musculus*, *D. melanogaster*, and *S. cerevisiae* genomes, and found that all the algorithms performed badly overall. In fact, barring yeast datasets, the performance on all organisms was significantly worse. This motivates developing more sophisticated motif-finding algorithms.

In the recent survey Eisen [4] brought up the issue that many of these contemporary methods does not incorporate the structural properties of the binding sites. In the real biological context, the regulatory motifs may be more intricate. For example, they may appear as motifs with spacers with sparse conservation in the binding sites. This characteristics is significant in the regulatory mechanism, because two or more sites are often recognized by the same protein (as is frequently the case, for instances, of RNA polymerase) [14]. Moreover the binding sites are often recognized by different macromolecular complexes that make contact with one another [16, 25]. Our focus in this paper is to find such spaced motifs.

2. Related Work

There have been some works on extraction of motifs with spacers. OligoDyad [22] finds spaced motifs by counting the number of occurrences of each defined spaced pair of trinucleotides, before finally assessing its statistical significance. MITRA [5] is another algorithm for finding dyads. SesiMCMC [6] first optimizes the weight matrix with Gibbs sampling for a given motif and spacer length, and finally looks for the best motif and spacers length based on the information content of motif and distributions of motif occurrence position. The methods reported in [3, 15] use suffix tree to store the regularly spaced motif before finally identifying the motif pairs. YMF [19] handles spacers by enumerating all the possible gap lengths between two composite motifs.

The approaches used by the existing methods can be characterized in three ways. The first, and most common, approach is to address the problem by assuming fixed length spacers. The second approach is to address the finding of spaced motifs as finding single motif. The main characteristic of this approach is the use of efficient data structures to traverse the space of IUPAC patterns. The third approach handles spacers by enumerating all the possible gap lengths between two composite motifs. Though this approach can find motifs with varying length spacers, it is computationally expensive and often limited to finding short motifs.

In this paper we propose a new approach for finding spaced motifs, and develop a novel motif-finding algorithm that offers flexibility in handling variations in spacer length, the number of motif parts (henceforth called *motif segments*) and their lengths.

3. Our Approach

Central to our approach is the notion called the *submotifs*. Submotif is defined as a conserved sub-region of a given motif (spaced or not). Our idea is to use a divide-and-conquer approach whereby we find the target motif by first finding its submotifs and then strategically compositing them to deduce the target motif. The number of submotifs is assumed to

be unknown and they may be possibly overlapping. This setting has the following advantages:

- (1) The length of the target motif need not be known *even if* we pre-fix the length of the submotif. This follows because any l -mer can be represented as a union of its substrings (possibly overlapping) of length $l_s, l_s < l$. Thus this method can result in higher sensitivity.
- (2) The conservation of the instances formed by union of the submotif instances will be stronger in conservation than the instances obtained directly without submotifs. This can be seen by observing that the former instances will always be a subset of the latter. This implies that the method can yield higher specificity.
- (3) It provides a natural extension for finding motifs with spacers, in which neither the spacer length nor the segment length need to be known. This is because the method attempts to find the longest motif and the presence of spacers does not preclude submotif finding (provided the compositing of submotifs is done appropriately).

However, the challenge is that there could be potentially too many submotifs (many of them spurious) and how effectively the submotif-compositing can be done to return "good" motifs. To tackle this problem, we formulate this task as a constrained frequent itemset mining problem and propose a new algorithm for solving it. This algorithm can output spaced patterns optimized based on user considerations such as degree of conservation, range of spacer widths, etc. The algorithm and our overall method are described in the following sections.

The rest of the paper is organized as follows. Section 4 explains the definition of our novel motif model and terminologies. It is followed by section 5 where we describe our proposed method in detail. In section 6 we present our experimental results on benchmark dataset in comparison with existing tools and results on real biological data. Finally in section 7 we will conclude our paper by discussing the strengths and limitations of our approach and directions for our future research.

4. Problem Definition

Let $\mathbf{S} = \{s_1, s_2, \dots, s_t\}$ be a set of t sequences. Our aim is to identify the target motif(s) and its instances from this set of sequences. We view the target motif as a cascade of segments separated by spacers, where both the segments and spacers could be of varying lengths (that is unknown). Thus we define the motif finding problem as one of finding all segments and spacers of a motif. As stated in the previous section, our key concept is *submotif* which will be used to model the segments and in turn the motif.

Let $hd(x, x')$ denote the Hamming distance between strings x and x' . A string x is called a $(\mathbf{l}_s, \mathbf{d})$ - **submotif** of a set of strings $\mathbf{X}' = \{x'_1, x'_2, \dots, x'_k\}$, each of length l_s if $hd(x, x'_i) \leq d; \forall i = 1, 2, \dots, k$. In this case x'_i is called an instance of the $(\mathbf{l}_s, \mathbf{d})$ - **submotif** x . Next we define the motif segments.

A string m is called a $(\mathbf{l}_s, \mathbf{d})$ - **segment** of length $|m|$ if it contain at least 2 overlapping $(\mathbf{l}_s, \mathbf{d})$ - **submotifs**. The total length of the $(\mathbf{l}_s, \mathbf{d})$ -submotif is equal to $|m|$. For

illustration, consider a 3-segments spaced motif shown in Figure 1. In the example, x_{11} and x_{12} are (l_s, d) -submotifs with x'_{11} and x'_{12} as their instances. Furthermore m_1 is a (l_s, d) -segment with m'_1 as one of its instances.

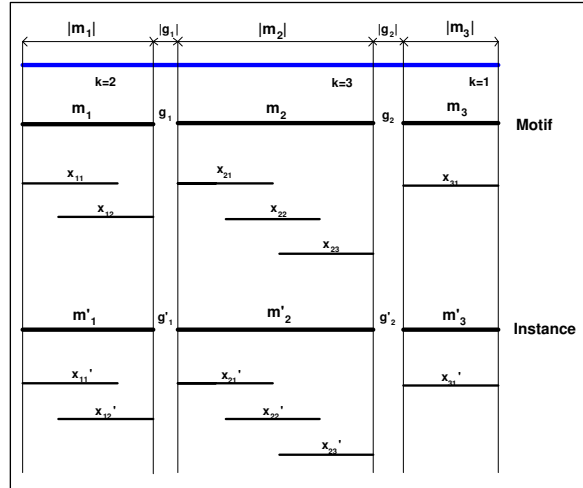


Figure 1. A 3-parts motif with submotifs and an instance.

A string $M = m_1 \cdot g_1 \cdot m_2 \dots m_{N-1} \cdot g_{N-1} \cdot m_N$ is called a (l_s, d, q, e) – motif of S , where m_i is a string for $i = 1, 2, \dots, N$ and g_i is a spacer of length $|g_i|$ for $i = 1, 2, \dots, N - 1$. And it has the following properties:

- (1) there exist at least q of its instances $M' = m'_1 \cdot g'_1 \cdot m'_2 \dots m'_{N-1} \cdot g'_{N-1} \cdot m'_N$ in S
- (2) m'_i is a (l_s, d) -segment instance of m_i for $i = 1, 2, \dots, N$.
- (3) $|g_i - g'_i| \leq e$ for $i = 1, 2, \dots, N$.

For illustration, in Figure 1, $M = m_1 \cdot g_1 \cdot m_2 \cdot g_2 \cdot m_3$ is a (l_s, d, q, e) – motif. Given a set of sequence S , our motif finding problem is to find (l_s, d, q, e) – motif. In the next section, we describe our algorithm, called SPACE to solve this problem.

5. Algorithm SPACE

Algorithm SPACE consists of three major components namely, the *generation of candidate patterns*, *constrained frequent pattern mining*, and finally *significance testing*. Generation of candidate patterns is the step where we list the candidate motifs and their respective (l_s, d) -segments and (l_s, d) -submotifs. In the second step, we process every candidate motif and its segments and submotifs to mine frequent submotif patterns. Eventually, in the last part we score the motifs based on its input sequences and background model significance testing. Figure 3 depicts the overall strategy of our algorithm.

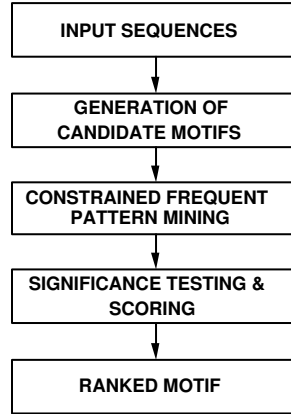


Figure 2. Outline of SPACE motif discovery algorithm

5.1. Generation of Candidate Motifs

In this step, a set of motif candidates $\mathbf{M} = \{M_1, M_2, \dots, M_k\}$ for all $k = 1, 2, \dots, N \leq t(n - l + 1)$ is identified from the input sequences $\mathbf{S} = \{s_1, s_2, \dots, s_t\}$. This is done by enumerating all l length substrings from \mathbf{S} and then scanning the sequences to obtain instances for each candidate motif generated. Motif candidates M_i will be formed by (l_s, d) -submotif. Similarly its instances will be formed by string of length l_s where $l_s < l$. And Hamming Distance between (l_s, d) -submotif with its instances is less or equal to d . Furthermore we require that the coverage of the submotif instances should be greater or equal than l_c . The coverage value l_c determines in what way does the conserved segments are placed to each other. This measurement allows us to determine segments that comes with spacers of varying length. In principle the l_c can be seen as the total length of the N number of (l_s, d) -segment strings of the given instances. It is computed as follows: $\sum_{i=1}^N |m'_i|$.

At the end of the candidate motif generation step, we will have a hash table \mathbf{B} in which each entry is a motif candidate and its corresponding instances. Note that each instance in turn consists of submotif instances, some of which could be spurious. To minimize the spurious instances and hence maximize the chances of identifying the target motif, we employ a frequent motif pattern mining method.

5.2. Constrained Frequent Pattern Mining

We present a novel method that aims to identify genuine submotifs (and hence the segments) borrowing ideas from Association Rule Mining [9]. However, it should be noted that our intention is not to mine rules but rather submotif patterns that are more *structured*. The mined patterns are structured because 1) the order of the submotifs is important, 2) the spacings between them have to be uniform. These two conditions imply a constrained association mining algorithm, described below.

Before presenting the algorithm, we introduce a key concept called the *generalized gap*.

This is used to denote the spacings between the submotifs which could overlap^a.

5.2.1. GENERALIZED GAP

Gap between two strings is conventionally described as any maximal, consecutive run of spaces in a single string [7]. This description presupposes that they are non-overlapping. We hereby introduce a new notion of a generalized definition of gap that includes the overlapping situation.

Generalized gap 5.1. Let p be the position of x_1 and q be the position of x_2 in a string M . We say that (p, q, α) is a pair of a string M if $\alpha = M[p..p + |\alpha| - 1] = M[q..q + |\alpha| - 1]$, and $p < q$. The gap of pair x_1 and x_2 , denoted as $g(x_1, x_2)$ is the number of characters $q - p - |\alpha|$ between the two occurrences of the substring α . If the gap is non-negative then the pair is non-overlapping.

The following lemma describes the additive property of the generalized gap, which will be useful in our analysis later in this section.

Lemma 5.1. *Given an ordered substrings set $\mathbf{x} = \{x_1, x_2, \dots, x_K\}$ of string w , each of length l_s . The generalized gap between x_1 and x_K is: $g(x_1, x_K) = g(x_1, x_2) + \dots + g(x_{K-1}, x_K) + (K - 2) * l_s$.*

For the illustration of generalized gap and proof of the lemma above please refer the full paper version.

5.2.2. MINING OF CONSTRAINED FREQUENT PATTERNS

Recalling from section 5.1, the generation of candidate motifs processes the input sequences to output a table **B**. Note that each entry of this table is a motif candidate and its instances, and each motif candidate instance in turn contains (l_s, d) -submotif instances. The constrained frequent pattern mining step processes each entry of the table **B** and generates frequent motif patterns. We describe this process by considering a single entry of this table, i.e. a motif candidate M and its instances. This set of instances can be viewed as a *transactional* database, henceforth called **D**.

Let $M' = \{M'_1, M'_2, \dots, M'_i\}$ be a set of instances of M , for some i . Let $\mathbf{X}'_i = \{x'_{i1}, x'_{i2}, \dots, x'_{ij}\}$ be the set of (l_s, d) -submotif instances contained in M'_i , for $j = 1, \dots, l - l_s + 1$. Then, **D** can be thought of as a table with M'_i as its rows, as shown in Table 1 below.

We first map all the submotif instances \mathbf{X}'_i in **D** to a distinct set of items, called I . Given $M' = \{M'_1, M'_2, \dots, M'_i\}$ and $\mathbf{X}'_i = \{x'_{i1}, x'_{i2}, \dots, x'_{ij}\}$, let the union of \mathbf{X}'_i , namely $\mathbf{X}' = \{\mathbf{X}'_1 \cup \mathbf{X}'_2 \dots \cup \mathbf{X}'_i\}$. Then we cluster the elements of \mathbf{X}' if the pairwise Hamming distance between any two strings is less or equal to $2d$. Elements of \mathbf{X}' that do not satisfy this criteria are assumed to form isolated clusters. Let there be k clusters formed using this procedure. We reduce each cluster to a string that is a consensus of the strings in the cluster.

^aWe use 'gap' and 'spacer' interchangeably to denote the same concept.

M'_i	$\mathbf{X}'_i, \mathbf{G}'_i$
M'_1	$x'_{11}, g(x'_{11}, x'_{13}), x'_{13}$
M'_2	$x'_{23}, g(x'_{23}, x'_{25}), x'_{25}, g(x'_{25}, x'_{26}), x'_{26}$
M'_3	$x'_{32}, g(x'_{32}, x'_{33}), x'_{33}$
...	...
M'_i	$x'_{ia}, g(x'_{ia}, \dots), \dots, g(\dots, x'_{ib}), x'_{ib}$
	s.t. $(1 \leq (a, b) \leq i)$

Denoting the consensus by x_i , we obtain the set $\mathbf{I} = \{x_1, x_2, \dots, x_k\}$. We will call this as the submotif set.

Using the submotif set \mathbf{I} , we will derive frequent submotif pattern(s) in \mathbf{D} . We will deduce the motif(s) from these frequent pattern(s). We provide some definitions before describing our method.

Definition 5.2. (*n*-itemset pattern) A tuple $\mathbf{P}_n = (x_1, g_1, x_2, g_2, \dots, g_{n-1}, x_n)$, $n \geq 2$, is called a *n*-itemset pattern, where $x_i \in I$ and $g_i = g(x_i, x_{i+1})$, is the generalized gap between x_i and x_{i+1} as defined in Definition 1. For $n = 1$, $\mathbf{P}_1 = (x_1)$.

Unlike the definition of itemset in Association Rule mining, the itemset in our algorithm also includes the integer value of a gap.

Since x_i is a substring of a motif candidate (which is of length l), observe that the biggest gap occurs when there are only two strings x_1 and x_2 and they are located at the two ends of the motif candidate. Then the gap between them is $l - 2l_s$. We denote this maximum gap as g_{max} . The smallest gap occurs when two adjacent substrings x_i and x_{i+1} overlap such that the gap is $-l_s + 1$. (Note that they cannot overlap entirely, because then x_i and x_{i+1} will be interpreted as a single submotif instance. See step 8 in Algorithm 1.) We denote this minimum gap as g_{min} .

Definition 5.3. (Length of *n*-itemset pattern) Given a *n*-itemset pattern \mathbf{P}_n , the length of \mathbf{P}_n is defined as $L(\mathbf{P}_n) = g(x_1, x_n) + 2l_s$.

We next describe a tree growing procedure that will be useful in the description of our algorithm.

Consider a tree \mathbf{T} . We assume the tree has an empty root node at Level 0. We will grow the tree such that the nodes in the *n*-th level of the tree are *n*-itemset patterns, for $n \geq 1$. Level 1 nodes also can be thought of as 1-itemset patterns one each for every x_i . i.e. if the *k*-th node in Level 1 is denoted $\mathbf{P}_1^{k_1}$, then $\mathbf{P}_1^{k_1} = (x_{k_1})$, $k_1 = 1, \dots, k$. Each Level 1 node is grown as follows. Let $\mathbf{G} = \{g_{min}, \dots, g_{max}\}$. Let $\mathbf{L}_2^{k_1}$ stand for the children of $\mathbf{P}_1^{k_1}$. Then $\mathbf{L}_2^{k_1} = \mathbf{P}_1^{k_1} \times \mathbf{G} \times \mathbf{I}$, where: $\mathbf{P}_1^{k_1} \times \mathbf{G} \times \mathbf{I} = \{(\mathbf{P}, g, x) | \mathbf{P} \in \mathbf{P}_1^{k_1}, g \in \mathbf{G}, \text{ and } x \in \mathbf{I}\}$.

It can be noted that each element of $\mathbf{L}_2^{k_1}$ is actually a 2-itemset pattern denoted by $\mathbf{P}_2^{k_1, k_2} = (x_{k_1}, g_1, x_{k_2})$, where $g_1 \in \mathbf{G}$, $x_{k_1} \in \mathbf{I}$. Generalizing this process to the *n*-th level, we obtain: $\mathbf{L}_n^{k_1, \dots, k_{n-1}} = \mathbf{P}_{n-1}^{k_1, \dots, k_{n-2}} \times \mathbf{G} \times \mathbf{I}$.

Each element of $\mathbf{L}_n^{k_1, \dots, k_{n-1}}$ is a *n*-itemset pattern, given by $\mathbf{P}_n^{k_1, \dots, k_n} = (x_{k_1}, g_1, \dots, g_{n-1}, x_{k_n})$, $x_{k_i} \in \mathbf{I}$ and $g_i \in \mathbf{G}$. For convenience we denote $\mathbf{P}_n^{k_1, \dots, k_n}$ by

$\mathbf{P}_n^{k_n}$. The tree growth is stopped at node $\mathbf{P}_n^{k_n}$ if $L(\mathbf{P}_n^{k_n}) = l$.

Before describing our algorithm, we provide some definitions:

Definition 5.4. (Instance Satisfaction)

Given a n -itemset pattern $\mathbf{P}'_n = \{x'_1, g'_1, x'_2, g'_2, \dots, g'_{n-1}, x'_n\}$, where \mathbf{P}'_n is a subset of \mathbf{SM}'_i . We define $\mathbf{X}'_n = \{x'_1, x'_2, \dots, x'_n\}$ and $\mathbf{G}'_n = \{g'_1, g'_2, \dots, g'_{n-1}\}$ such that \mathbf{X}'_n and \mathbf{G}'_n is a subset of \mathbf{P}'_n . And given a gap tolerance value e . We say that \mathbf{P}'_n is the valid occurrence of \mathbf{P}_n if all the following conditions are satisfied:

- 1) $|\mathbf{P}_n| = |\mathbf{P}'_n|$,
- 2) $hd(x_i, x'_i) \leq d$, for $i = 1, \dots, n$; $x_n \in \mathbf{X}_n$ and $x'_n \in \mathbf{X}'_n$,
- 3) $|g_i - g'_i| \leq e$, for $i = 1, \dots, n - 1$; $g_i \in \mathbf{G}_n$ and $g'_i \in \mathbf{G}'_n$.

Note that checking the valid occurrence of \mathbf{X}'_n may require us to determine all possible gaps between any ordered combination of elements in \mathbf{SM}'_i . We use Lemma 1 to expedite this step.

Definition 5.5. (Support) The support of an n -itemset patterns, denoted as $\text{sup}(\mathbf{P}_n)$, is the number of times in which an n -itemset patterns satisfy Definition 9 in \mathbf{D} .

Definition 5.6. (Frequent Patterns) A n -itemset patterns \mathbf{P}_n is *frequent* if its support is more than or equal to some threshold minimum support (q).

Definition 5.7. (Closed Frequent Pattern) A n -itemset patterns \mathbf{P}_n is a *closed frequent pattern* if there exist no n -itemset pattern \mathbf{P}'_n such that 1) $\mathbf{P}'_n \subset \mathbf{P}_n$ 2)for all transaction \mathbf{D} , $\mathbf{P}_n \in \mathbf{D} \rightarrow \mathbf{P}'_n \in \mathbf{D}$.

We can use the naive tree growing procedure to propose a brute-force algorithm for mining frequent patterns. It is easy to see that the brute-force algorithm will return all closed frequent itemsets, as all valid frequent itemsets are enumerated. However it is too inefficient to be executed in practice. It runs in $O(|\mathbf{I}|l^n)$. Below we propose an efficient algorithm to solve this problem.

The idea of the algorithm is to determine all the closed frequent pattern by merging nodes of the tree with all the valid nodes at second level of the tree.

The algorithm is presented in Algorithm 2. In step 1, we initialize the frequent pattern set. A submotif set is obtained in step 2. From this submotif set we initialize level 1 of the tree in step 3. The subsequent tree growing procedure is divided into two parts. First part generates level 2 as shown in step 7-24. We find gaps of all ordered pairs in \mathbf{D} in step 9-13. Then we obtained all the nodes of level 2 in step 14. Frequent pattern at level 2 is verified in steps 17-24. We use closed frequent pattern at level 2 to generate nodes level 3 onwards as shown in step 26. And closed frequent pattern verification is done in step 29-33.

Definition 5.8. (Generalized A-priori Condition) A n -itemset pattern is frequent only if its parents k -itemset patterns, for all $k = 1, \dots, n - 1$, are frequent.

From this definition we derive the following result:

Lemma 5.2. *Algorithm 2 satisfies the Generalized Apriori Condition (stated in Definition 13).*

Theorem 5.1. *(Correctness of Algorithm 1) Algorithm 1 returns all closed frequent patterns.*

Algorithm 1 Constrained Frequent Pattern Mining

ConstrainedFP(D, gap_threshold(e), q)

```

1: FP =  $\phi$ 
2: I = Cluster(D)
3:  $L_1^0 = \{P_1^1, \dots, P_1^{|I|}\}$ , where  $P_1^{k_1} = \{x_{k_1}\}$ 
4:  $\{G', I', \bar{G}\} = \phi$ 
5: n = 2
6: for all  $P_{n-1}^{k_{n-1}} \in L_{n-1}^{k_{n-1}}$  do
7:   if (n = 2) then
8:     for all  $i \in M_i$  do
9:       for all pairs  $(x_{ij}, x_{ik})$  where  $x_{ij}, x_{ik} \in X'_i$  do
10:        compute  $g(x_{ij}, x_{ik})$  using Lemma 1
11:        append  $g(x_{ij}, x_{ik}), g(x_{ij}, x_{ik}) + e,$ 
           and  $g(x_{ij}, x_{ik}) - e$  into  $\bar{G}$ 
12:      end for
13:    end for
14:     $L_2^{k_1} = P_1^{k_1} \times \{\bar{G} \times I\}$ 
15:    for  $P_2^{k_2} \in L_2^{k_1}$  do
16:       $min\_sup(P_2^{k_2}) \leftarrow CountSupport(D, P_2^{k_2}, e)$ 
17:      if  $sup(P_2^{k_2}) \geq q$  then
18:        append  $G_2^{k_2}$  to  $\hat{G}_{x_{k_2}}$ ,
           where  $x_{k_2}$  is last element of  $P_2^{k_2}$ 
19:        append  $X_2^{k_2}$  to  $\hat{I}_{x_{k_2}}$ 
20:        append  $P_2^{k_2}$  to FP
21:      else
22:        stop expanding  $P_2^{k_2}$ 
23:      end if
24:    end for
25:  else
26:     $L_n^{k_{n-1}} = P_{n-1}^{k_{n-1}} \times \{\hat{G}_{x_{k_{n-1}}} \times \hat{I}_{x_{k_{n-1}}}\}$ ,
           where  $x_{k_{n-1}}$  is last element of  $P_{n-1}^{k_{n-1}}$ 
27:    for all  $P_n^{k_n} \in L_n^{k_{n-1}}$  do
28:       $sup(P_n^{k_n}) \leftarrow CountSupport(D, P_n^{k_n}, e)$ 
29:      if  $sup(P_n^{k_n}) \geq q$  then
30:        append  $P_n^{k_n}$  to FP
31:      else
32:        stop expanding  $P_n^{k_n}$ 
33:      end if
34:    end for
35:  end if
36:  n  $\leftarrow$  n + 1
37: end for
38: RETURN FP

```

Proof for Lemma 5.2 and correctness analysis of Theorem 5.1 can be found in the full paper version.

5.3. Significance Testing and Scoring

Obtaining frequent motif patterns allows us to minimize spurious instances. But it does not say anything about its biological significance. Weeder [17] introduced an effective method to examine the biological significance by considering input sequences and a background model. This evaluation is tied up with the motif scoring mechanism. We adapt Weeder scoring mechanism and extend it to include motifs with *spacers*.

First of all, we have the table of expected frequency values $\varepsilon(p)$ for oligonucleotides ranging from 6 to 8 bp. They are pre-computed from background sequences from RSAT database [23]. The formula used for computing this frequency is: $\varepsilon(p) = o(p)/T$. Where $o(p)$ is the total number of times p was found in the background sequences, and T is the total number of length m oligos in the background sequences.

For motif longer than 8 bp, that comes with or without spacers the expected frequency is modelled using seventh order Markov chain. Let $p = p_1p_2 \dots p_n$ be an n -mer with n greater than 8. It is computed as follows:

$$\varepsilon(p) = \varepsilon(p_1p_2 \dots p_8) \prod_{i=9}^n P(p_i|p_{i-7} \dots p_{i-1}) \quad (1)$$

Let n be a spacer and $\mathbf{n} = \{n_{i1}, \dots, n_{ij}\}$ be set of j number of spacers, at position i_1, \dots, i_j where $j \leq i$. The conditional probability $P(p_i|p_{i-7} \dots p_{i-1})$ of having nucleotide p_i preceded by nucleotides $p_{i-7} \dots p_{i-1}$, is computed by using the expected frequency of 8-mers:

$$P(p_i|p_{i-7}, \dots, p_{i-1}) = \frac{\sum_{n_{i1}} \dots \sum_{n_{ij}} \varepsilon(p_{i-7}p_{i-6} \dots p_{i-2}n_{i1}p_{i-2}n_{ij}p_i)}{\sum_{n_{i1}} \dots \sum_{n_{ij}} \sum_{\bar{n}_i} \varepsilon(p_{i-7}p_{i-6} \dots p_{i-3}n_{i1}p_{i-1}n_{ij}\bar{n}_i)}$$

Given the motif p by allowing d substitution, with or without spacers, we use formula (2) to obtain motif score. It consists of *sequence specific score* - $\sigma(p)$ and *background score* - $\beta(p)$. The sequence specific, measures the quality of the motif with the regard to input data set. And background score measures the quality of the motif with regard to background model.

Let l_1, \dots, l_t be the length of input sequence in \mathbf{S} . Let d_i be the minimum number of mutations p appears in the i -th sequence. The sequence specific score is given by:

$$\sigma(p) = \sum_i \log \frac{1}{\varepsilon(p, d_i).l_i}$$

Furthermore, the background score is given by:

$$\beta(p) = \log \frac{o(p, d)}{\varepsilon(p, d) \cdot \sum l_i}$$

The final motif score is obtained by adding up the two values: $MotifScore(p) = \sigma(p) + \beta(p)$.

The overall algorithm SPACE can be seen in Algorithm 3. It follows the structure showed in Figure 3.

Algorithm 2 SPACE motif finding algorithm

FindMotif($l, d, l_s, l_c, e, min_sup, \mathbf{S}$)

- 1: Initialize **SEL_MOTIFS** as set of selected motif
 - 2: Initialize **FP_ALL** as set of all frequent patterns
 - 3: append *GenCandMotifs*($l, d, l_s, l_c, \mathbf{S}$) to **B**
 - 4: **for** each $\mathbf{D} \in \mathbf{B}$ **do**
 - 5: append *ConstrainedFP*(\mathbf{D}, e, min_sup) to **FP_ALL**
 - 6: **end for**
 - 7: Initialize **FP_INST** as set of instances of the given FP
 - 8: **for** each $FP \in \mathbf{FP_ALL}$ **do**
 - 9: append *FindInstances*(FP) to **FP_INST**
 - 10: $M \leftarrow \textit{FindConsensus}(\mathbf{FP_INST})$
 - 11: $Score \leftarrow \textit{FinalScore}(M)$
 - 12: append to **M,Score** to **SEL_MOTIFS**
 - 13: **end for**
 - 14: RETURN Ranked **SEL_MOTIFS**
-

In the next section we will described the preliminary experimental result.

6. Experimental Results

We perform experiments of Algorithm SPACE on two classes of datasets and report our preliminary results. The first class comprises the 56 datasets that Tompa [21] used for the assessment of motif discovery algorithms. This provides a direct way for comparing the performance of our algorithm with 13 other well known algorithms. However, to further analyze its capabilities we consider 3 synthetic datasets and report performance results. For this case we compare our algorithm with Weeder which was the best performing algorithm on Tompa's assessment experiments.

For performance measurement, we use the same four measures proposed by Tompa [21] namely, *sensitivity* (nSN), *positive predictive value* ($nPPV$), *performance coefficient* (nPC), and *correlation coefficient* (nCC). Index n is used to denote that the assessment is done at the nucleotide level instead of site level.

6.1. Results on Tompa's Benchmark Data set

Tompa's benchmark dataset has been constructed based on real transcription factor binding sites drawn from four different organisms [21]. It consists of 56 datasets in total. The number of sequences ranges from 1-35 and the sequence lengths may be up to 3000 bp.

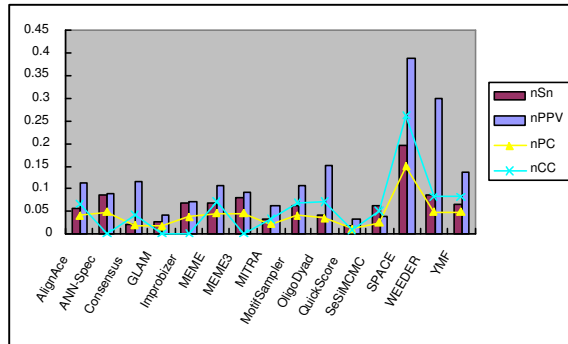


Figure 3. Performance comparison between SPACE algorithm over 13 other motif discovery tools. It represents the total performance value from 4 types of organism in the datasets.

The performance of Algorithm SPACE averaged over all datasets is shown in Figure 3. For this run, we did not use any expert input or per dataset adjustment. It can be observed that our algorithm performs significantly better compared to all other algorithms over all four measures. Compared to the second-best algorithm Weeder, our algorithm achieved more than double sensitivity improvement. We believe this is due to the fact that Algorithm SPACE cannot be constrained by the motif length and indeed can find longer motifs. The algorithm also achieved better positive predicted value. As noted in Section 3, this advantage results because of submotif modeling. To illustrate this further, we show the binding sites identified by our algorithm and Weeder on two specific datasets: *hm22m* and *hm17g*.

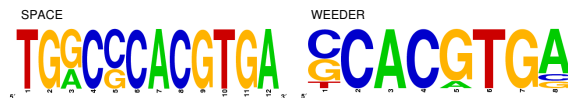


Figure 4. Binding sites reported in *hm22m* (human). SPACE with $nSn = 0.26$, $nPPV = 0.62$, $nPC = 0.23$, $nCC = 0.39$ and submotifs: TGGCC, CCACG, CGTGA. Weeder with $nSn = 0.28$, $nPPV = 0.47$, $nPC = 0.21$, $nCC = 0.35$.

Figure 4 shows the motifs output by SPACE and WEEDER on *hm22m*. It can be observed that SPACE found a 12 length motif and WEEDER only an 8 length motif. Figure 5 shows the actual binding sites identified by each algorithm (blue shows the correct binding site and green color shows the binding site found by the algorithm). It can be noted that SPACE does not identify any spurious binding sites.

We also analyze the performance of SPACE across the four organisms. Figure 6 shows the average scores of SPACE for each organism, compared with the best performing algorithm for the respective organism.

On *yeast*, Weeder is the algorithm that did exceptionally well. SPACE achieved only marginal improvements. Improbizer and YMF performed the best on *mouse* dataset.

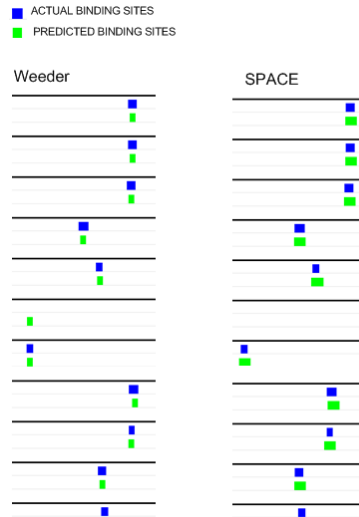


Figure 5. Binding sites reported by SPACE in hm17g (*human*), with $nSn = 0.90$, $nPPV = 0.72$, $nPC = 0.67$ and $nCC = 0.80$. Weeder with $nSn = 0.61$, $nPPV = 0.89$, $nPC = 0.57$ and $nCC = 0.73$.

Though SPACE performed as good as the best in terms of sensitivity, it was significantly better on PPV score. On *fly* datasets, SeSiMCMC did best in terms of both PPV and sensitivity. SPACE was significantly worse on sensitivity but did well on PPV. However, on *human* dataset where ANN-Spec was the best-performing algorithm, SPACE achieved significantly better on both sensitivity and PPV. The latter results show evidence that our algorithm can indeed find complex regulatory sites.

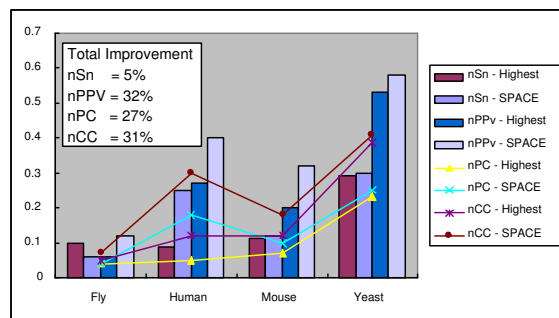


Figure 6. Performance comparison of SPACE and best performing algorithms on 4 types of organisms. Followed by percentage of improvement given by SPACE.

6.2. Results on Synthetic Data set

To further analyze the capabilities of SPACE, we we conduct experiments on 3 synthetic datasets created by planting motifs (with and without spacers) on randomly generated sequences. All the 3 datasets contain 10 sequences with the length of each sequence fixed at 300 bp. The motifs implanted in the datasets as as follows: 1) An (8,1)-motif *TGGGTACC* implanted in 5 out of 10 sequences of 300bp each. 2) A (15,1)-motif *CCTGTNNNAGTTGTC* containing 2 segments of length 5 and 7 with a spacer of length 3. 3) Same as the motif in (2) but the instances have spacers of varying length 2-4bp.

We compare our algorithm with Weeder which is the best performing algorithm on Tompa's benchmark datasets. We run Weeder in the exhaustive mode (i.e. *large* mode). The performance results are shown in Figure 7.

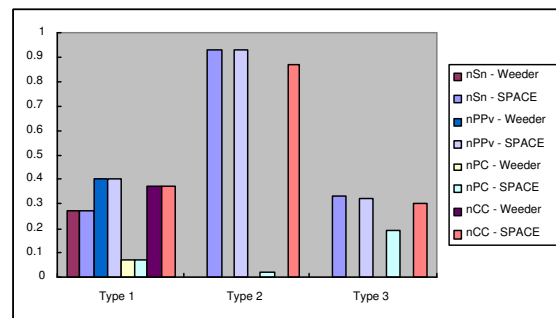


Figure 7. Performance comparison of SPACE and Weeder on 3 synthetic datasets.

For Dataset 1 (simple motif with no spacer), SPACE and Weeder return the same motif *GTACC* which is a substring of the target motif. On Dataset 2, SPACE returns a spaced motif *CCTGTNNNAGTTG* whereas Weeder gives a spurious motif *TTAGTATA*. On Dataset 3, SPACE again returns a spaced motif *CTGTANNNTTGTC* and Weeder gives *TAATGTTTGT*. On further investigation, we found that Weeder identifies part of the correct motif *GTAGGTG* albeit at a lower rank.

The results show that SPACE can find spaced motifs with high sensitivity and PPV even when the binding sites contain varying length spacers.

7. Discussion and Conclusions

In this paper we have proposed a new approach for motif-finding through the notion called submotifs. We developed a novel motif-finding algorithm SPACE that offers flexibility in handling variations in spacer length, the number of motif segments and their lengths. SPACE uses a divide-and-conquer approach whereby the target motif is detected by first finding its submotifs and then strategically compositing them using a novel frequent pattern mining compositing them to deduce the target motif. We showed that the algorithm can find multiform TFBS containing spacers.

The existing spaced motif methods generally make several assumptions on spacer length, motif length or number of segments. OligoDyad [22] finds spaced motifs by finding motif pairs with fixed length spacer. Suffix tree approach by Sagot [15] can find multi-segment motifs but the spacer length is required to be fixed. YMF [19] is the algorithm that allows varying length of spacers. However, this is done by enumerating all the possible gap lengths which is computationally expensive and hence is usually limited to finding short motifs. In comparison, SPACE does not assume fixed length spacers and can find longer motifs. This is possible because of the novel constrained frequent pattern mining method employed by SPACE. This method enables structured submotif patterns can be mined from all the submotifs found, without strong assumptions on the spacer/motif length or the number of segments. The fact that the algorithm allows overlapping submotifs also enables the algorithm to find varying length motifs.

This paper reports only preliminary results of our ongoing work. Currently we are optimizing some key components of the algorithm for efficiency, and also testing it on more real biological datasets.

References

1. Ao, W. Gaudet, J., Kent, W.J., et.al (2001) Environmentally induced foregut remodeling by PHA-4/FoxA and DAF-12/NHR, *Science*, **305**, 1743-1746.
2. Bailey, T. Elkan, C. (1995) Unsupervised learning of multiple motifs in biopolymers using expectation maximization *Machine Learning*, **21**, 51-80.
3. Carvalho, A.M., Freitas, A.T., Olivera, A.L. (2003) A Highly Scalable Algorithm for the Extraction of Cis-Regulatory Regions, In *Proceedings of the Third Asia-Pacific Bioinformatics Conference (APBC)*, 273-282.
4. Eisen, M.B., (2005) All motifs are not created equal: structural properties of transcription factor - DNA interaction and the inference of sequence specificity, *Genome Biology*, **6**, P7.
5. Eskin, E. and Pevzner, P. (2002) Finding composite regulatory patterns in DNA sequences, *Bioinformatics*, **18**, S354-S363.
6. Favorov, A.V., Gelfand, M.S., Gerasimova, A.V., Ravcheev, D.A., Mironov, A.A., and Makeev, V.J. (2005) A Gibbs sampler for identification of symmetrically structured, spaced DNA motifs with improved estimation of the signal length., *Bioinformatics*, **21**, 2240-5.
7. Gusfield, D. (1997) *Algorithm on Strings, Trees, and Sequences: Computer Science, and Computational Biology*, 235-236.
8. Hertz, G.Z. and Stormo, G.D. (1999) Identifying DNA and protein patterns with statistically significant alignments of multiple sequences, *Bioinformatics*, **15**, 563-577.
9. Jiawei, H. and Kamber, M. (2000) *Data Mining: Concepts and Techniques*.
10. Jensen, K.L., Styczynski, M.P., Rigoutsos, I. and Stephanopoulos, G. (2006) A generic motif discovery algorithm for sequential data, *Bioinformatics*, **22**, 21-28.
11. Down, T.A. and Hubbard, T.J.P., (2005) NestedMICA: sensitive inference of overrepresented motifs in nucleic acid sequence, *Nucleic Acids Research*, **33**:5, 1445-1453.
12. Keich, U. and Pevzner, P. (2002) Finding motifs in the twilight zone. In *Proceedings of the Sixth Annual International Conference on Research in Computational Molecular Biology (RECOMB)*, 195-204.
13. Lawrence, C. and Altschul, S. et.al (1993) Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment, *Science*, **193**, 133-154.
14. Lewin, B. (1997) *Genes VI*, Oxford University Press.
15. Marsan, L. and Sagot, M-F. (2000) Algorithms for Extracting Structured Motifs Using a Suffix

- Tree With an Application to Promoter and Regulatory Site Consensus Identification, *Journal of Comp. Biol.*, **7**, 345-360.
16. Owen,G., and Zelent,A. (2000) Origins and evolutionary diversification of nuclear receptor superfamily, *Cell Mol. Life. Sci.* , **57**, 809-827.
 17. Pavesi,G., Mauri,G. and Pesole,G.,(2001) An algorithm for finding signals of unknown length in DNA sequences, *Bioinformatics*, **17**, S207-S214.
 18. Rigoutsos,I. and Floratos, A.(1998) Combinatorial pattern discovery in biological sequences, *Bioinformatics*, **14**, 55-67.
 19. Sinha,S. and Tompa,M. (2000) A statistical method for finding transcription factor binding sites. In *Proceedings of the 8th International Conference on Intelligent Systems for Molecular ISMB-00* , 344-354.
 20. Thijs,G., et.al. (2002) A Gibbs sampling method to detect over-represented motifs in the upstream regions of co-expressed genes, *Journal of Computational Biology*, **9**, 447-464.
 21. Tompa,M., Li,N., Bailey,T, et.al (2005) Assessing computational tools for the discovery of transcription factor binding sites, *Nature Biology*, **23**, 137-144.
 22. van Helden,J., Rioas,A.F., and Collado-Vides,J. (2000) Discovering regulatory elements in non-coding sequences by analysis of spaced dyads, *Nucleic Acids Research*, **28**, 1808-1818.
 23. van Helden,J.(2003) Regulatory sequence analysis tools, *Nucleic Acids Research*, **31**, 3539-3596.
 24. Wijaya,E., Kanagasabai R., Bramachary,M., Bajic,V.B and Sung,S.Y. (2005) A Hybrid Algorithm for Motif Discovery from DNA Sequences, *APBC 2006*.
 25. Werner,T. (1999) Models for prediction and recognition of eukaryotic promoters, *Mammalian Genome*, **10**, 168-175.
 26. Wingender, E., Dietze, P., Karas, H., and Knüppel, R. (1996) TRANSFAC: a database on transcription factors and their DNA binding sites *Nucleic Acid Rresearch*, **24**, 238-241.
 27. Zaki,M. (2001) Spade: An efficient algorithm for mining frequent sequences, *Proceedings of 9th Conf. of Information and knowledge management*, 422-429.

Detection of regulatory elements with constrained submotif patterns mining

Supplementary Material

By. Edward WIJAYA, Rajaraman KANAGASABAI and SUNG Wing Kin

Appendix A Generalized Gap

With regards to the Figure 1 below, we characterize x_1 as vector \overrightarrow{AB} meaning string that spans from A to B . Similarly $x_2 = \overrightarrow{CD}$ and $x_3 = \overrightarrow{EF}$. Then we define the directional length of string x_1 as: $|x_1| = |\overrightarrow{AB}|$ and $|x_1| = -|\overrightarrow{BA}|$. The gap value between the two strings is $g(x_1, x_2) = |\overrightarrow{BC}|$.

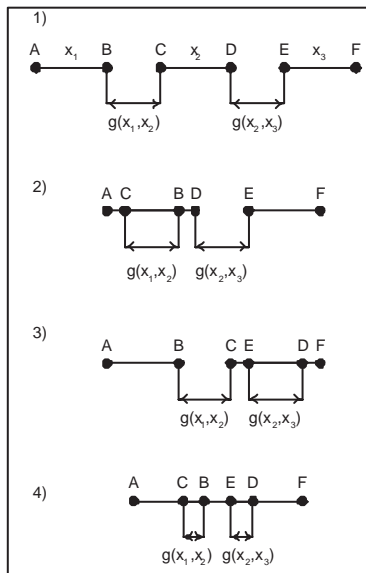


Figure 1: Four possibilities of arrangement between 3 strings: x_1 , x_2 , and x_3 at any linear string. Example 1 shows that $|\overrightarrow{BC}|$ is a *positive* gap when x_1 and x_2 do not overlap. In example 2, $|\overrightarrow{CB}|$ is a *negative* gap, when x_1 and x_2 overlap. Same conception also hold for gap between x_2 and x_3 .

Appendix B Additive Property of Generalized Gap

The following proposition describes the additive property of the generalized gap. This is used throughout our Constrained Submotif Patterns Mining step.

Proposition 1 Given an ordered substrings set $\mathbf{x} = \{x_1, x_2, \dots, x_K\}$ of string w , each of length l_s . The generalized gap between x_1 and x_K is: $g(x_1, x_K) = g(x_1, x_2) + \dots + g(x_{K-1}, x_K) + (K - 2) * l_s$.

Proof: Firstly, we will give proof for $K = 3$. Given three consecutive strings $x_1 = \overrightarrow{AB}$, $x_2 = \overrightarrow{CD}$, and $x_3 = \overrightarrow{EF}$ (see Figure 1 above). Suppose we intend to find the gap between x_1 and x_3 , which is $g(x_1, x_3) = |\overrightarrow{BE}|$. There are 4 different cases in which they can relate to one another.

Case 1 is where all the strings do not overlap. Because the length of the string x_1, x_2 and x_3 are uniform (l_s), we can obtain that:

$$\begin{aligned} g(x_1, x_3) \simeq |\overrightarrow{BE}| &= |\overrightarrow{BC}| + |\overrightarrow{CD}| + |\overrightarrow{DE}| \\ &= g(x_1, x_2) + l_s + g(x_2, x_3) \end{aligned} \tag{1}$$

Case 2 is when x_1 overlaps with x_2 , but x_2 does not overlap with x_3 . The proof is as follows:

$$\begin{aligned} g(x_1, x_3) \simeq |\overrightarrow{BE}| &= |\overrightarrow{BD}| + |\overrightarrow{DE}| \\ &= (|\overrightarrow{CD}| - |\overrightarrow{CB}|) + |\overrightarrow{DE}| \\ &= |\overrightarrow{CD}| + |\overrightarrow{BC}| + |\overrightarrow{DE}| \\ &= l_s + g(x_1, x_2) + g(x_2, x_3) \end{aligned} \tag{2}$$

Case 3 is when x_1 does not overlap with x_2 , but x_2 overlap with x_3 . The proof is similar to Case 2.

Case 4, when all of the three strings overlap with each other. That is x_1 overlaps with x_2 and x_2 overlaps with x_3 . The proof is as follows:

$$\begin{aligned} g(x_1, x_3) \simeq |\overrightarrow{BE}| &= |\overrightarrow{CD}| - (|\overrightarrow{CB}| + |\overrightarrow{ED}|) \\ &= |\overrightarrow{CD}| + |\overrightarrow{BC}| + |\overrightarrow{DE}| \\ &= l_s + g(x_1, x_2) + g(x_2, x_3) \end{aligned} \tag{3}$$

These prove the lemma for $K = 3$. When $K = 4$, we have 4 strings: $x_1, x_2, x_3,$ and x_4 . We can compute $g(x_1, x_4)$ by considering the 3 consecutive strings $x_1, x_3,$ and x_4 . By the above proof we can derive:

$$\begin{aligned} g(x_1, x_4) &= g(x_1, x_3) + l_s + g(x_3, x_4) \\ &= (l_s + g(x_1, x_2) + g(x_2, x_3)) + \\ &\quad g(x_3, x_4) + l_s \\ &= g(x_1, x_2) + g(x_2, x_3) + \\ &\quad g(x_3, x_4) + (2 * l_s) \end{aligned} \tag{4}$$

Thus the lemma can be generalized for any K . ■

Appendix C Clustering Submotif Instances

Consider these submotif instances sets, with $l_s = 5$ and $d = 1$:

$$\begin{aligned}\mathbf{X}'_1 &= \{CCTAG, CTAGG, GGATT\} \\ \mathbf{X}'_2 &= \{GCTGG, CTGGG, GGTGT\} \\ \mathbf{X}'_3 &= \{GCTAG, CTAGG, GGGGT\}\end{aligned}$$

Forming union of these sets will give us:

$$\mathbf{X}' = \{CCTAG, CTAGG, GGATT, GCTGG, CTGGG, GGTGT, GCTAG, CTAGG, GGGGT\}$$

Then we cluster the elements of \mathbf{X}' if the pairwise Hamming distance between any two strings is less or equal to $2d$. Elements of \mathbf{X}' that do not satisfy this criteria are assumed to form isolated clusters. Initially we begin by taking the string CCTAG. The first cluster is formed with CCTAG and GCTGG as members. \mathbf{X}' is pruned as and when its members are assigned to a cluster. Using the same process, we obtain 3 clusters:

$$\begin{aligned}\{CCTAG, GCTGG, GCTAG\} \\ \{CTAGG, CTGGG, CTAGG\} \\ \{GGATT, CTGGG, GGGGT\}\end{aligned}$$

We reduce each cluster to a string that is a consensus of the strings in the cluster. Hence, we derive the submotif set:

$$\mathbf{I} = \{GCTAG, CTAGG, GGGGT\}$$

Appendix D Proof of Lemma 1 - Satisfaction of Generalized A-priori Condition

It is enough to prove the lemma for $n \geq 2$. Let $n = 2$. At step 5 of ConstrainedFP subroutine we have $\mathbf{P}_2^{k_2}$ from $\mathbf{P}_1^{k_1}$. Given $\mathbf{P}_1^{k_1} = \{x_{k_1}\}$, we obtain $\mathbf{P}_2^{k_2} = \{x_{k_1}, g_1, x_{k_2}\}$, where $g_1 \in \bar{\mathbf{G}}$ and $x_{k_1} \in \mathbf{I}$. Now by Step 6, it can be observed that $\text{sup}(\mathbf{P}_2^{k_2}) \leq \text{sup}(\mathbf{P}_1^{k_1})$. Hence, $\mathbf{P}_2^{k_2}$ is frequent only if its parent $\mathbf{P}_1^{k_1}$ is frequent.

Similarly for $n > 2$, at step 10 we generate $\mathbf{P}_n^{k_n}$ from $\mathbf{P}_{n-1}^{k_{n-1}}$. Given $\mathbf{P}_{n-1}^{k_{n-1}} = \{x_{k_1}, g_1, \dots, g_{n-2}, x_{k_{n-1}}\}$, we obtain $\mathbf{P}_n^{k_n} = \{x_{k_1}, g_1, \dots, g_{n-1}, x_{k_n}\}$, where $g_i \in \hat{\mathbf{G}}_{\mathbf{x}_{k_{n-1}}}$ and $x_{k_i} \in \hat{\mathbf{I}}_{\mathbf{x}_{k_{n-1}}}$. By step 12, it follows that $\text{sup}(\mathbf{P}_n^{k_n}) \leq \text{sup}(\mathbf{P}_{n-1}^{k_{n-1}})$. Thus, $\mathbf{P}_n^{k_n}$ is frequent only if $\mathbf{P}_{n-1}^{k_{n-1}}$ is frequent. Now the lemma follows by induction. \blacksquare

Appendix E Proof of Theorem 1 - Correctness of ConstrainedFP Procedure

It is enough to show that any n -itemset pattern that is not explored by ConstrainedFP subroutine is not frequent. In other words, $\mathbf{P}_n^{k_n} \notin \mathbf{L}_n^{k_{n-1}}$, for all n , is not frequent.

Let $n = 2$. Let $\bar{\mathbf{G}}^c$ be a complement of $\bar{\mathbf{G}}$ and let $\mathbf{P}_2^c = \{x_1, g_1^c, x_2\}$, be a pattern in $\mathbf{P}_1^{k_1} \times \{\bar{\mathbf{G}}^c \times \mathbf{I}\}$. Let $x_1, x_2 \in I$, be such that both x_1 and x_2 can be found in M'_j , for some $1 \leq j \leq i$. Let $g_1' = g(x_1, x_2)$, the generalized gap between x_1 and x_2 . Consider the 2-itemset pattern $\mathbf{P}'_2 = \{x_1, g_1', x_2\}$. We will prove that $|g_1^c - g_1'| > e$. This will imply that Condition (3) of Instance satisfaction will not be satisfied and hence the pattern will be infrequent. Case 1: Suppose $g_1^c > g_1'$. From the Step 5, $\{g_1', g_1' + e, g_1' - e\}$ subset of $\bar{\mathbf{G}}$. This means that $g_1^c > g_1' + e$. Thus we have:

$$\begin{aligned} |g_1^c - g_1'| &= g_1^c - g_1' \\ &> e \end{aligned}$$

Case 2: Suppose $g_1^c < g_1'$. Following similar arguments as in Case 1, we have $g_1^c < g_1' - e$. Thus,

$$\begin{aligned} |g_1^c - g_1'| &= g_1' - g_1^c \\ &> e \end{aligned}$$

Hence the theorem follows for $n = 2$.

Let $n > 2$. We consider two cases.

Case 1: Every pattern in $\bar{\mathbf{L}}_n^{k_{n-1}} = \mathbf{P}_{n-1}^{k_{n-1}} \times \{\hat{\mathbf{I}}_{\mathbf{x}_{k_{n-1}}}^c \times \mathbf{G}\}$ is not frequent. Where $\hat{\mathbf{I}}_{\mathbf{x}_{k_{n-1}}}^c$ is the complement of $\hat{\mathbf{I}}_{\mathbf{x}_{k_{n-1}}}$. Note that the proof for this case is enough for $\bar{\mathbf{G}}$, since it is a set with all possible gaps within the range of tolerance (e) in \mathbf{D} (see step 5). Let $\bar{\mathbf{P}}_n^{k_n}$ be a pattern of $\bar{\mathbf{L}}_n^{k_{n-1}}$. We denote $\bar{\mathbf{P}}_n^{k_n} = \{x_{k_1}, g_1, \dots, x_{k_{n-1}}, g_{n-1}, x_{k_n}^c\}$, such that $x_{k_n}^c \in \hat{\mathbf{I}}_{\mathbf{x}_{k_{n-1}}}^c$ and $g_{n-1} \in \bar{\mathbf{G}}$. We know that $\bar{\mathbf{P}}_2^{k_2} = \{x_{k_{n-1}}, g_{n-1}, x_{k_n}^c\}$ is not frequent because as verified by step 18, only pattern that contain string in $\hat{\mathbf{I}}_{\mathbf{x}_{k_{n-1}}}$ can satisfy the minimum support condition. Now, by Lemma 1, it follows that $\bar{\mathbf{P}}_n^{k_n}$ is also not frequent.

Case 2: Every pattern in $\bar{\mathbf{L}}_n^{k_{n-1}} = \mathbf{P}_{n-1}^{k_{n-1}} \times \{\hat{\mathbf{G}}_{\mathbf{x}_{k_{n-1}}}^c \times \mathbf{I}\}$ is not frequent. Where $\hat{\mathbf{G}}_{\mathbf{x}_{k_{n-1}}}^c$ is the complement of $\hat{\mathbf{G}}_{\mathbf{x}_{k_{n-1}}}$. Let $\bar{\mathbf{P}}_n^{k_n}$ be a pattern of $\bar{\mathbf{L}}_n^{k_{n-1}}$. We denote $\bar{\mathbf{P}}_n^{k_n} = \{x_{k_1}, g_1, \dots, x_{k_{n-1}}, g_{n-1}^c, x_{k_n}\}$,

such that $x_{k_i} \in \mathbf{I}$ and $g_{n-1}^c \in \widehat{\mathbf{G}}_{\mathbf{x}_{k_{n-1}}}^c$. We know that $\bar{\mathbf{P}}_2^{\mathbf{k}_2} = \{x_{k_{n-1}}, g_{n-1}^c, x_{k_n}\}$ is not frequent because by step 18 we can see that only pattern that contain gaps in $\widehat{\mathbf{G}}_{\mathbf{x}_{k_{n-1}}}$ can satisfy the minimum support condition. Hence, as in Case 1, Lemma 1 implies that $\bar{\mathbf{P}}_n^{\mathbf{k}_n}$ not frequent. ■

Appendix F Post-processing

On performing the post-processing procedure, our contention is that real motif should have positive significance over background sequences and it should be overrepresented over different runs. Our application has the following parameterization mode:

	small	medium	large
maximal motif length: (l)	8, 15, 20	8, 15, 20	8, 15, 20, 25
coverage: (l_c)	0.8l	0.5l, 0.8l	0.5l, 0.8l
min support: (q)	0.5t	0.5t	t, 0.5t
submotif length: (l_s)	5	5	5
max mismatch: (d)	1	1	1
gap tolerance: (e)	1	1	1

Table 1: SPACE parameterization scheme (t = total number of input sequences). Every mode runs on combination of respective parameters. For example, "small" mode runs on 3 parameter sets.

We would then determine the final motifs with the following method. First we would determine the score of the motif, using the scoring method explained in Section 4. This score also represent the significance of the motif given the background model. Then overrepresentation of the motif is found by intersecting the high ranking motifs of all the parameter sets.

Two motifs are said to have a valid intersection if there is has at least 60% intersecting (common) nucleotides. We use the intersection count as the basis of finding the redundancy of a motif. A motif is called *horizontally* redundant if it occurs in more than one parameter sets. It is called *vertical* redundant motif if it occurs in more than once in a single parameter sets. Thus the total overrepresentation count of a motif is the combined count of horizontal and vertical redundancy. Only motifs that has horizontal redundancy greater than 1 and vertical redundancy lesser than 2 will be considered as potential motifs.

Appendix G Evaluation Measure

For performance measurement, we use the same four measures proposed in (Tompa et.al, 2005). The following abbreviations are used to specify how the scores are calculated: TP (true positive) is the number of the overlapped nucleotides both in real and predicted sites, FP (false positives) the number of the overlapped nucleotides not in known sites but in predicted sites, TN (true negatives) the number of the overlapped nucleotides neither in known sites and in predicted sites, and FN (false negatives) the number of the overlapped nucleotides neither in known sites and but not in predicted sites.

The performance of the algorithm is measured according the following statistics: *sensitivity*, *positive predictive value (PPV)*, *specificity*, *performance coefficient (nPC)*, *average site performance (ASP)* and *correlation coefficient (CC)*. They are formulated as follows:

$$\text{Sensitivity} \underset{\text{(Recall)}}{=} TP/(TP + FN)$$

$$\text{(Precision)} \overset{PPV}{=} TP/(TP + FP)$$

$$\text{Specificity} = TN/(TN + FP)$$

$$PC = TP/(TP + FN + FP)$$

$$ASP = (SN + PPV)/2$$

$$CC = \frac{TP.TN - FN.FP}{\sqrt{(TP + FN)(TN + FP)(TP + FP)(TN + FN)}}$$

Appendix H Result of Tompa’s Benchmark Dataset

Data set	nSn	nPPV	nSp	nPC	nCC	sSn	sPPV	sASP
dm01g	0.24	0.6	0.9965957	0.2068966	0.3717255	0.5714286	0.6666667	0.6190476
dm02r	0	0	0.9846233	0	-0.0195568	0	0	0
dm03m	0.0769231	0.2	0.9945726	0.0588235	0.1146627	0.2222222	0.5	0.3611111
dm04g	0	0	0.9927527	0	-0.0110985	0	0	0
dm05g	0	0	0.9938692	0	-0.0114708	0	0	0
dm06r	0	0	0.9896623	0	-0.0184691	0	0	0
dm07m	NaN	0	0.9933333	0	NaN	NaN	0	NaN
dm08m	NaN	0	0.995	0	NaN	NaN	0	NaN
hm01g	0	NaN	1	0	NaN	0	NaN	NaN
hm02r	0.077821	0.2	0.9908498	0.0593472	0.1091124	0.0909091	0.2	0.1454545
hm03r	0	0	0.9938322	0	-0.0129914	0	0	0
hm04m	0	NaN	1	0	NaN	0	NaN	NaN
hm05r	0.0673575	0.2888889	0.9885999	0.0577778	0.1129465	0.0909091	0.3333333	0.2121212
hm06g	0.72	0.36	0.9783051	0.3157895	0.4980148	0.6666667	0.6	0.6333333
hm07m	0	0	0.9897394	0	-0.0162251	0	0	0
hm08m	0.6210526	0.6051282	0.9894665	0.4419476	0.6028473	0.7692308	0.7692308	0.7692308
hm09g	0	0	0.9966307	0	-0.0060049	0	0	0
hm10m	0.505618	0.4368932	0.9800756	0.3061224	0.4525698	0.5454545	0.4615385	0.5034965
hm11g	0	0	0.992239	0	-0.0162153	0	0	0
hm12r	0.1428571	0.3333333	0.9784946	0.1111111	0.1815056	0.2	0.3333333	0.2666667
hm13r	0	0	0.9922892	0	-0.0145724	0	0	0
hm14r	0.1829268	0.3333333	0.9843587	0.1339286	0.2236712	0.25	0.3333333	0.2916667
hm15r	0	0	0.9949431	0	-0.0075615	0	0	0
hm16g	0	0	0.9956806	0	-0.0058205	0	0	0
hm17g	0.9034483	0.7197802	0.9904762	0.6683673	0.8006721	0.9	1	0.95
hm18m	0	0	0.996647	0	-0.0044426	0	0	0
hm19g	0.137931	0.16	0.9738914	0.08	0.1201381	0.25	0.2	0.225
hm20r	0	NaN	1	0	NaN	0	NaN	NaN
hm21g	0	0	0.9898105	0	-0.0138362	0	0	0
hm22m	0.2641509	0.6222222	0.9941258	0.2276423	0.3922857	0.4	0.6666667	0.5333333
hm23r	0.1048951	0.25	0.9757674	0.0797872	0.1218341	0.2	0.25	0.225
hm24m	0.4891304	0.375	0.9808086	0.2694611	0.4129583	0.625	0.4166667	0.5208333
hm25g	0	0	0.9623656	0	-0.052249	0	0	0
hm26m	0	NaN	1	0	NaN	0	NaN	NaN
mus01r	0	0	0.9681979	0	-0.0431029	0	0	0
mus02r	0.0431034	0.2	0.995438	0.0367647	0.0821723	0.0833333	0.2	0.1416667
mus03g	0.1338028	0.38	0.9868533	0.1098266	0.199479	0.2222222	0.4	0.3111111
mus04m	0.0378788	0.1428571	0.9910926	0.0308642	0.0554699	0.0714286	0.1428571	0.1071429
mus05r	0.125	0.275	0.9848326	0.0940171	0.160901	0.1666667	0.25	0.2083333
mus06g	0	0	0.9790649	0	-0.0308899	0	0	0
mus08m	0	0	0.9899081	0	-0.0096373	0	0	0
mus09r	0	0	0.9687174	0	-0.0363628	0	0	0
mus10g	0.2421525	0.6428571	0.997652	0.2134387	0.3886159	0.2	0.6	0.4
mus11m	0.3175355	0.7444444	0.9960269	0.2863248	0.4751734	0.4	0.8571429	0.6285714
mus12m	0.1241379	0.3	0.9690037	0.0962567	0.1404561	0.2857143	0.5	0.3928571
yst01g	0.0983607	0.2	0.9945934	0.0705882	0.1320874	0.1428571	0.2	0.1714286
yst02g	0.4259259	0.575	0.9820296	0.3239437	0.4705325	1	0.625	0.8125
yst03m	0.1632653	0.3428571	0.9880613	0.1243523	0.2171341	0.1666667	0.4285714	0.297619
yst05r	0.5	1	1	0.5	0.6983587	0.75	1	0.875
yst06g	0.4375	0.875	0.997006	0.4117647	0.6072548	1	0.875	0.9375
yst08r	0.2795699	0.65	0.9960825	0.2429907	0.4172308	0.5714286	0.6666667	0.6190476
yst09g	0.2930233	0.525	0.996389	0.2316176	0.3862269	0.6153846	0.5333333	0.574359
Fly	0.0566319	0.1217949	0.9935269	0.0402116	0.0732482	0.1176471	0.2068966	0.1622718
Human	0.0988474	0.2959064	0.9957287	0.0800253	0.1626543	0.147651	0.3333333	0.2404922
Mouse	0.122807	0.3181818	0.9915556	0.0972222	0.1822944	0.1702128	0.3265306	0.2483717
Yeast	0.2982774	0.5812721	0.9948363	0.2455224	0.4068082	0.5147059	0.6034483	0.5590771
Total	0.1259488	0.3337524	0.9949291	0.1006444	0.1955473	0.1976517	0.3768657	0.2872587

Table 2: Assessment results on Tompa’s Benchmark Dataset. Prefix n refers to the evaluation on the nucleotide level and s refer to the site level.