IMPROVED ALGORITHMS FOR DISCOVERY OF TRANSCRIPTION FACTOR BINDING SITES IN DNA SEQUENCES

A Dissertation

by

XIAOYAN ZHAO

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2010

Major Subject: Computer Science

Improved Algorithms for Discovery of Transcription Factor Binding Sites in DNA Sequences

IMPROVED ALGORITHMS FOR DISCOVERY OF TRANSCRIPTION FACTOR BINDING SITES IN DNA SEQUENCES

A Dissertation

by

XIAOYAN ZHAO

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	Sing-Hoi Sze
Committee Members,	Jianer Chen
	Vivek Sarin
	Ruzong Fan
Head of Department,	Valerie E. Taylor

December 2010

Major Subject: Computer Science

ABSTRACT

Improved Algorithms for Discovery of Transcription Factor Binding Sites in DNA Sequences. (December 2010)

Xiaoyan Zhao, B.S., Beijing Normal University

Chair of Advisory Committee: Dr. Sing-Hoi Sze

Understanding the mechanisms that regulate gene expression is a major challenge in biology. One of the most important tasks in this challenge is to identify the transcription factors binding sites (TFBS) in DNA sequences. The common representation of these binding sites is called "motif" and the discovery of TFBS problem is also referred as motif finding problem in computer science. Despite extensive efforts in the past decade, none of the existing algorithms perform very well.

This dissertation focuses on this difficult problem and proposes three new methods (MotifEnumerator, PosMotif, and Enrich) with excellent improvements. An improved pattern-driven algorithm, MotifEnumerator, is first proposed to detect the optimal motif with reduced time complexity compared to the traditional exact pattern-driven approaches. This strategy is further extended to allow arbitrary don't care positions within a motif without much decrease in solvable values of motif length. The performance of this algorithm is comparable to the best existing motif finding algorithms on a large benchmark set of samples.

Another algorithm with further post processing, PosMotif, is proposed to use a string representation that allows arbitrary ignored positions within the non-conserved portion of single motifs, and use Markov chains to model the background distributions of motifs of certain length while skipping these positions within each Markov chain. Two post processing steps considering redundancy information are applied in this algorithm. PosMotif demonstrates an improved performance compared to the best five existing motif finding algorithms on several large benchmark sets of samples.

The third method, Enrich, is proposed to improve the performance of general motif finding algorithms by adding more sequences to the samples in the existing benchmark datasets. Five famous motif finding algorithms have been chosen to run on the original datasets and the enriched datasets, and the performance comparisons show a general great improvement on the enriched datasets.

DEDICATION

To my parents, for all their sacrifices, and to my husband, Xiaohua, for all his support

ACKNOWLEDGEMENTS

I would never have been able to finish this dissertation without the guidance from my committee members, help from friends and support from my family.

I would like to thank my committee chair, Dr. Sing-Hoi Sze, for his tremendous guidance, support and patience for my study and research. He has helped me to overcome many difficulties to complete this dissertation. I am also grateful to all the members of my committee, Dr. Jianer Chen, Dr. Vivek Sarin and Dr. Ruzong Fan, for their encouragement during my down times and helpful inputs to my research.

I would like to thank my officemates, Yue Lu and Qingwu Yang, for valuable discussions and inputs. I would also like to thank Songjian Lu, Fenghui Zhang, and Jiahao Fan, for their encouragement and help when I needed it the most. Thanks also go to all my friends and the department faculty and staff for making my time at Texas A&M University a great experience.

Finally, I would like to thank my husband, my parents and parents-in-law for their continuous support and encouragement throughout all these years. My gratitude also go to my children, Leyou and Leyao, who have brought unparalleled and numerous joy to me while I am working on this dissertation.

TABLE OF CONTENTS

CHAPTER Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

CHAPTER I

INTRODUCTION

A. Background Overview

1. Introduction

In molecular biology, transcription is the synthesis of a single-stranded RNA molecule using the DNA template (one strand of DNA is transcribed). The regulatory sequences are stretches of DNA sequences which are binding sites for RNA polymerase and its accessory molecules, and a wide variety of transcription factors. Together, the regulatory sequences with their bound proteins act as molecular switches that determine the activity state of the gene e.g., OFF or ON. These binding sites are located in the regulatory region of the gene and a single transcription factor can be bound to different binding sites that have different underlying DNA sequences. Motif is the common representation of these binding sites. The discovery of motifs will allow the biologist to understand the complex mechanism that regulates gene expression. However, it is very difficult due to the characteristics of real input samples. These are:

1. The length of binding sites is unknown. It is usually 5-12, but can be up to 30.
2. The binding site sequence preference is not exact. There may be some mismatches.
3. The majority of motifs are unknown to us.

This dissertation follows the style of *Journal of Computational Biology.*

4. Samples with biased nucleotide composition.

5. Corrupted samples (not every sequence contains a motif).

6. Regulatory sites can lie on either DNA strand.

A DNA motif is generally defined as a recurring pattern within a sequence of nucleotides. In real DNA sequence, it is usually a short segment that occurs frequently, but is not required to be an exact copy for each occurrence. A Motif can be visually represented by a motif logo (Figure 1.1), which is a summary of the possible nucleotide strings that correspond to the same motif. The motif logo length equals the length of those strings and, for each position, the logo represent the information content of that position. The total height of a motif logo in a position is proportional to the information content in that position, while the height of each letter is proportional to the frequency of the letter in that position. The sum of the heights of all letters in a position equals to the total height of the motif in that position.

The general motif finding problem can be defined as follows:

Input: A set of regulatory sequences that possibly bind to the same protein transcription factor.

Aim: Use a computational algorithm to search for the common binding site pattern that occurs frequently.

If an l-letter pattern appears exactly in every sequence, a simple enumeration of all patterns of length l that appear in the sequences gives the solution. However, the real

problem is not that simple because patterns in DNA sequences may include mutations, insertions or deletions of nucleotides. In fact, the motif finding problem has been proven to be NP-hard.

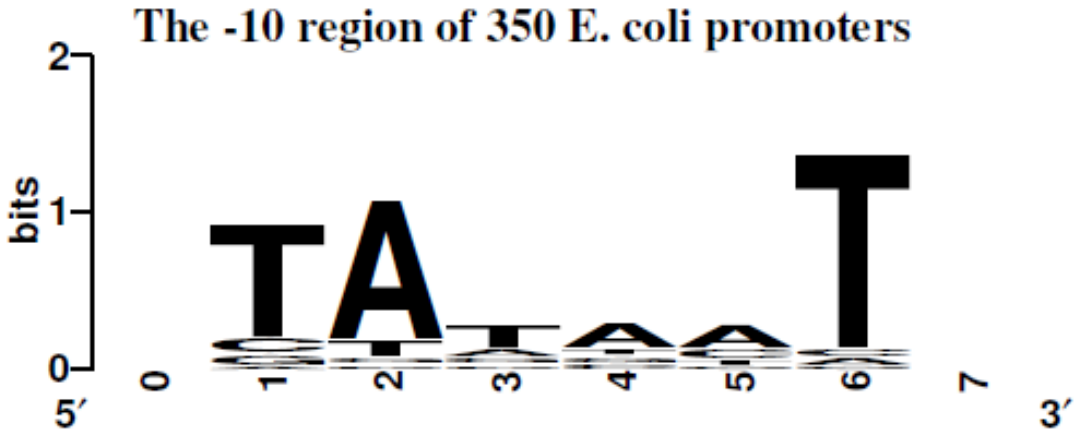
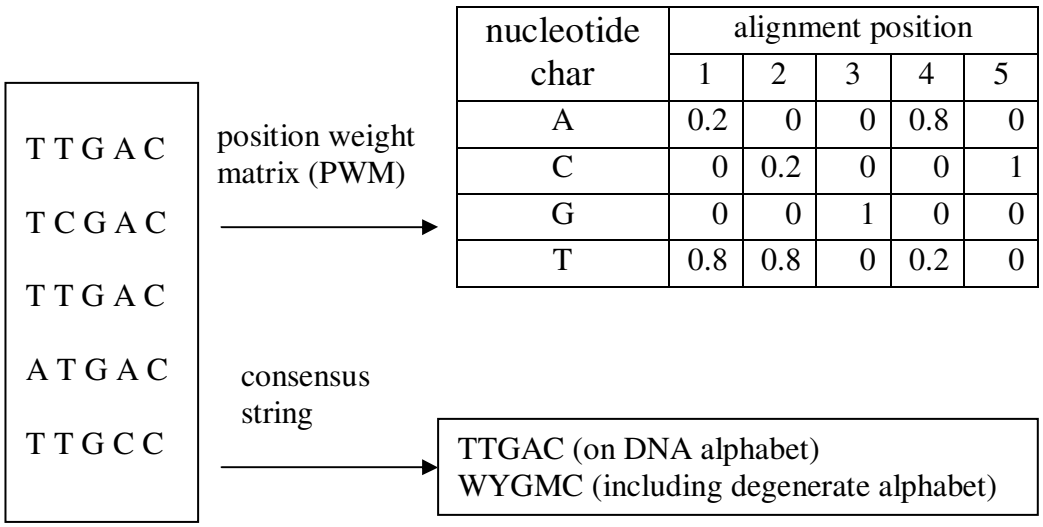


Figure 1.1: The graphic representation of an aligned set of 350 E. coli promoters. A logo displays the frequencies of bases at each position, as the relative heights of letters, along with the degree of sequence conservation as the total height of a stack of letters, measured in bits of information (T. D. Schneider and R. M. Stephens 1990).

Depending how the motif is modeled, most of the motif finding methods can be grouped into two categories: probabilistic approach and combinatorial approach. In probabilistic approaches, a motif is modeled to be a matrix that each column represents a probability distribution for the four letters in that position; while in combinatorial approaches, a

motif is usually modeled to be a string of characters. These characters can be any letters in DNA alphabet, or degenerate alphabet (Figure 1.2).



nucleotide char	alignment position				
	1	2	3	4	5
A	0.2	0	0	0.8	0
C	0	0.2	0	0	1
G	0	0	1	0	0
T	0.8	0.8	0	0.2	0

Figure 1.2: Example of different motif representations of five given binding sites. The position weight matrix shows the frequency of each nucleotide char in that position, the consensus string on DNA alphabet shows the most frequent nucleotide char in that position and the consensus string including degenerate alphabet shows the IUB code (degenerate base) in that position, for example, W represents A or T appearing in the first position.

2. Probabilistic Approaches

One of the earliest implementation of probabilistic approaches was a greedy algorithm to find the binding sites with the highest information content by Hertz et al. (1990). They

used this algorithm to identify a consensus motif that was present once in every sequence and their latest implementation (Consensus) Hertz and Stormo (1999) provided methods to estimate the statistical significance of a given information content score, and tested their algorithm to identify binding sites for the Escherichia coli CRP protein.

A well-known technique expectation-maximization (EM) is usually used in probabilistic motif finding algorithms. EM for motif finding was first introduced by Lawrence and Reilly and was then extended by Bailey and Elkan (1995) to identify motifs in unaligned biopolymer sequences in Multiple EM for Motif Elicitation algorithm (MEME). The MEME method assumes little is known in advance about any motifs that may be present in a set of biopolymer sequences, and it used real biopolymer subsequences as EM algorithm starting point, which increases the probability of finding globally optimum motifs. It also removed the constraint that exactly one occurrence of the shared motif in each sequence and probabilistically removed shared motifs to avoid reporting redundant motifs.

Another very popular statistical technique used in probabilistic motif finding is Gibbs sampling. The original Gibbs sampler for motif finding was developed by Lawrence et al. (1993) and it was only applied to protein sequences originally. It originally assumed that at least one instance of motif existed in every sequence. Gibbs sampling is a special case of a Markov Chain Monte Carlo method (MCMC) by sampling from unknown distributions by using Markov Chains and their properties of convergence to a stationary

distribution. Gibbs sampling is really easy to implement and it runs very fast, in linear time with the length of the sequences. It is more stable to initialization than EM methods, but also more dependent on all sequences exhibiting the motif. AlignACE (Roth et al. 1998) and MotifSampler (Thijs et al. 2001) are very useful motif finding applications built over Gibbs sampling. The original AlignACE used MAP (maximum *a priori* log-likelihood) score to evaluate different motifs sampled, which measures the degree of overrepresentation of a motif as compared to the expected random occurrence of that motif in the sequence under consideration. This measurement of scorning motifs was improved by Hughes et al. using group specificity, which avoids the main drawback of MAP score that some ubiquitously occurring but un-relevant motifs are scored too highly. MotifSampler incorporated a higher-order Markov-chain background model and used the probability distribution to estimate the number of motif occurrences in the a sequence. Other popular methods such as BioProspector Liu et al. and GibbsST Shida also applied Gibbs Sampling strategy with different modifications.

3. Combinatorial Approaches

The motif finding problem in combinatorial approaches can be formulated as:

Given sequence $S = \{x_1, x_2, \cdots, x_k\}$ and each sequence of length n

A motif is a consensus string of length l: $w_1 w_2 \cdots w_l$

The aim is to find the optimal motif *M*, best matches for *S*, which minimizes the following distances:

$$d(M,S) = \sum_i d(M,x_i),$$

where $d(M,x_i)$ = minimal hamming distance between *M* and any occurrence in x_i.

Based on the motif candidate search space, there are two groups in the category: pattern-driven algorithms and sample-driven algorithms. Pattern-driven algorithms (Queen et al. 1982; Waterman et al. 1984; Staden 1989; Pesole et al. 1992; Wolfertstetter et al. 1996; van Helden et al. 1998; Tompa 1999) usually use an exhaustive search over all possible strings of length l and report the one that minimizes the distance. Sample-driven algorithms ((Stormo and Hartzell 1989; Lawrence et al. 1993; Bailey and Elkan 1994; Hughes et al. 2000; Workman and Stormo 2000; Thijs et al. 2001) consider the candidate motifs appear in the sequences in *S* instead of enumerating every possible string of length l. The sample-driven algorithms have the advantage to suitable statistical models, but have the disadvantage that it is not possible to find the optimal motif unless the motif is very short (Leung and Chin 2005).

A straightforward algorithm for the pattern-driven approach takes $O(4^l lkn)$ time, thus this strategy is feasible only for small l. By considering only candidate motifs that are at most d substitutions away from a string appearing in the sample, an extended pattern-driven approach has been proposed to reduce the number of candidate motifs (Waterman et al. 1984; Galas et al. 1985). To further reduce the running time, another class of tree-based pruning techniques have been proposed (Marsan and Sagot 2000; Pavesi et al. 2001; Eskin and Pevzner 2002), while many approaches make use of the given

maximum distance d to develop heuristics that guarantee a high probability of finding the optimal motif (Buhler and Tompa 2002; Keich and Pevzner 2002; Price et al. 2003). A common weakness of these approaches is that they either cannot guarantee that the optimal motif is found or do not improve the worst case time complexity.

4. Evaluation Criteria and Benchmark Datasets

There are many motif finding algorithms developed based on varied and complex motif models and most authors test their algorithm using different biological sequences and synthetic data sets as well. Sinha and Tompa (2002) compared the performance of YMF to the algorithms MEME and AlignACE and observed that different tools performed better with different datasets. Tompa et al. (2005) assessed performance of thirteen motif finding algorithms, which provide both the standard evaluation criteria and benchmark datasets for assessing motif finding tools.

The evaluation criteria from Tompa et al. (2005) can be outlined as below.
In nucleotide level define the true positives (*nTP*), false positives (*nFP*), and others as follows:

- *nTP* is the number of positions that are in both predicted and known sites,
- *nFP* is the number of positions that are in predicted sites but not in known sites,
- *nFN* is the number of positions that are in known sites but not in predicted sites, and
- *nTN* is the number of positions that are not in predicted nor known sites respectively.

From these statistics, we compute the sensitivity (*nSn*) and others as follows:

- Sensitivity:

$$nSn = nTP/(nTP + nFN)$$

- Positive Predictive Value:

$$nPPV = nTP/(nTP + nFP)$$

- Specificity:

$$nSP = nTN/(nTN + nFP)$$

- Performance Coefficient:

$$nPC = nTP/(nTP + nFP + nFN)$$

- Correlation Coefficient:

$$nCC = \frac{nTP \cdot nTN - nFN \cdot nFP}{\sqrt{(nTP + nFN)(nTN + nFP)(nTP + nFP)(nTN + nFN)}}$$

A predicted site is defined to be overlapped with a known site if they overlap by at least one fourth of the known site, and similarly the site level statistics can be defined as:

- *sTP* is the number of known sites that have overlap with a predicted site,
- *sFP* is the number of predicted sites that do not have overlap with known sites, and
- *sFN* the number of known sites that do not have overlap with predicted sites.

From these statistics, we compute the sensitivity $sSn = sTP/(sTP + sFN)$, the positive predictive value $sPPV = sTP/(sTP + sFP)$, and the performance coefficient *sPC = sTP/(sTP + sFP + sFN).*

B. Our Contribution

Despite of the noticeable improvement in motif finding accuracy, current available motif finding methods are far from perfect, especially for higher organisms. In Chapter II – Chapter IV, we propose three new motif finding methods, which improve motif finding accuracy using three different approaches.

Since traditional motif finding formulations are NP-complete, a straightforward algorithm for the pattern-driven approach requires $O(4^l lkn)$ time, where k is the number of sequences, n is the length of each sequence and l is the motif length, which means this strategy is feasible only for small l. In Chapter II, we propose an improved pattern-driven algorithm that guarantees that all statistically significant motifs are found in $O(4^l lk)$ time. This algorithm saves a factor of n in time complexity over the original pattern-driven approach. This is a significant improvement since n can be as large as 3000 and is at least 200 or 300 in many promoter finding applications. It can be adapted to handle the case when a maximum distance d is given between a motif and its occurrences. It also extends the power of the pattern-driven approach to find all significant motifs of length around 12 or 13 (from the original limit of around 10), or substantially to around 20 while retaining most of the original sensitivity by allowing don't care positions but disallowing mismatches.

We show the success of this approach by testing test our algorithm on a large set of yeast samples constructed from co-expressed gene clusters from Tavazoie et al. (1999) and comparing to the best existing motif finding algorithms on a large benchmark set of samples from Tompa et al. (2005). The most advantage of this motif finding method is that we can guarantee the optimal motif is found with reduced time complexity.

In Chapter III, an improved algorithm based on skipping non-conserved positions in background Markov Chain is proposed. It is known in biology that there are often almost invariant positions that are critical for the binding process, thus we focus initially on positions that have fixed nucleotides to define core occurrences. While most approaches do not specifically take advantage of these positions, our model tries to capture them within positions that are not ignored. We compare the performance of our algorithm to other motif finding algorithms on a few benchmark data sets, and show that significant improvement in accuracy can be obtained. Furthermore, we applied Wilcoxon test to show that we have statistical improvements over some of the other tools.

In Chapter IV, a new strategy, Enrich, is proposed to improve the performance of motif finding algorithms. By modifying the existing benchmark datasets, we show that this strategy is able to improve the performance of five existing motif finding algorithms. The performance comparisons also indicate that this strategy would help to improve the quality of existing benchmark datasets as well.

CHAPTER II

IMPROVED PATTERN-DRIVEN ALGORITHMS

In order to guarantee that the optimal motif is found, traditional pattern-driven approaches perform an exhaustive search over all candidate motifs of length l. We develop an improved pattern-driven algorithm that takes $O(4^l lk)$ time, where k is the number of sequences in the sample and l is the motif length, which is independent of the length of each sequence n for large enough l and saving a factor of n in time complexity over the original pattern-driven approach. We further extend this strategy to allow arbitrary don't care positions within a motif without much decrease in solvable values of l. Testing this algorithm on a large set of yeast samples constructed from co-expressed gene clusters reveals that most biological motifs have many invariant or almost invariant positions and these positions can be used to define the motif while ignoring the other positions. This motivates the following two-stage strategy that extends the solvable values of l substantially for the pattern-driven approach: first use an $O(2^l lkn)$ algorithm to exhaustively search over all candidate motifs allowing arbitrary don't care positions but disallowing mismatches, then refine these motifs by allowing a limited amount of flexibility to model the almost invariant positions. We demonstrate that this seemingly restrictive motif definition is sufficiently powerful by showing that the performance of this algorithm is comparable to the best existing motif finding algorithms on a large benchmark set of samples.

A software program implementing these approaches (MotifEnumerator) is available at http://faculty.cs.tamu.edu/shsze/motifenumerator.

A. Introduction

There are roughly two types of general purpose motif finding algorithms. The first type includes sample driven approaches which identify the locations of the motif occurrences directly. The second type includes pattern-driven approaches which take advantage of the assumption that a motif can be specified by a central pattern and use it to reduce the search space. Although the sample-driven approach has more freedom to choose suitable statistical models (Stormo and Hartzell 1989; Lawrence et al. 1993; Bailey and Elkan 1994; Hughes et al. 2000; Workman and Stormo 2000; Thijs et al. 2001), the search space is usually so large that it is not possible to guarantee that the optimal motif is found unless the motif is very short (Leung and Chin 2005). In contrast, by assuming that a central string (in the DNA four-letter alphabet) can be used to describe the motif, it is possible for a pattern-driven approach to perform an exhaustive search over all 4^l candidate motifs for a moderately large motif length l and guarantee that the optimal motif is found (Queen et al. 1982; Waterman et al. 1984; Staden 1989; Pesole et al. 1992; Wolfertstetter et al. 1996; van Helden et al. 1998; Tompa 1999).

A straightforward algorithm for the pattern-driven approach takes $O(4^l lkn)$ time, where k is the number of sequences, n is the length of each sequence and l is the motif length,

thus this strategy is feasible only for small l. By considering only candidate motifs that are at most d substitutions away from a string appearing in the sample, an extended pattern-driven approach has been proposed to reduce the number of candidate motifs from 4^l to less than $\binom{l}{d}4^d kn$ (Waterman et al. 1984; Galas et al. 1985), and the reduction is significant when d is small relative to l. To further reduce the running time, another class of tree-based pruning techniques have been proposed (Marsan and Sagot 2000; Pavesi et al. 2001; Eskin and Pevzner 2002). Fraenkel et al. (1995) proposed to combine short candidate patterns to form longer patterns, while many approaches make use of the given maximum distance d to develop heuristics that guarantee a high probability of finding the best motif (Buhler and Tompa 2002; Keich and Pevzner 2002; Price et al. 2003).

A common weakness of these approaches is that they either do not improve the worst case time complexity of the straightforward algorithm or they cannot guarantee that the optimal motif is found. We have developed an improved pattern-driven algorithm that guarantees that all statistically significant motifs are found in $O(4^l lk)$ time. This algorithm is similar to the original pattern driven algorithm in exploring all 4^l candidate motifs of length l, but with the important difference that its time complexity is independent of the length of each sequence n (for large enough l), thus saving a factor of n in time complexity over the original pattern-driven approach. This is a significant improvement since n can be as large as 2000 and is at least 200 or 300 in many

promoter finding applications. The proposed algorithm extends the power of the pattern-driven approach to find all significant motifs of length around 12 or 13 (from the original limit of around 10). It can also be adapted to handle the case when a maximum distance d is given between a motif and its occurrences.

We further extend this strategy to allow arbitrary don't care positions within a motif without much decrease in solvable values of l. This is in contrast with many previous approaches that place various constraints on the don't care positions: Rigoutsos and Floratos (1998) imposed a constraint on the density of don't care positions and developed an algorithm to identify protein motifs, while Apostolico and Parida (2004) imposed maximality and irredundancy constraints on motifs and gave an algorithm to solve the problem in cubic time when mismatches are not allowed. Although these algorithms can find very long motifs, a common weakness is that a large number of statistically significant motifs may be missed due to the constraints. Apart from these algorithms, many other approaches identify sets of composite motifs that are separated by a variable number of don't care positions, but do not allow don't care positions within each individual motif (Marsan and Sagot 2000; van Helden et al. 2000; GuhaThakurta and Stormo 2001; Liu et al. 2001; Eskin and Pevzner 2002).

We allow arbitrary don't care positions within a motif and test our algorithm on a large set of yeast samples constructed from co-expressed gene clusters from Tavazoie et al. (1999). From the results, we observe that most biological motifs have many invariant or

almost invariant positions and these positions can be used to define the motif while ignoring the other positions. This motivates the following two-stage strategy: first use an $2^l lkn$ algorithm to exhaustively search over all candidate motifs allowing don't care positions but disallowing mismatches, then refine these motifs by allowing a limited amount of flexibility to model the almost invariant positions. With the much smaller exponential factor in the time complexity, this algorithm extends the solvable values of l substantially to around 20 while retaining most of the original sensitivity. We demonstrate that this seemingly restrictive motif definition is sufficiently powerful by showing that the performance of this algorithm is comparable to the best existing motif finding algorithms on a large benchmark set of samples from Tompa et al. (2005).

B. Problem Formulation

Our formulation makes a few simplifying assumptions: the central string is in the DNA four-letter alphabet and mutations occur at random positions within a motif. There are other approaches that do not have these restrictions, including those that use more general alphabets or profiles to represent a central pattern (Sinha and Tompa 2000; Price et al. 2003; Eskin 2004; Kel et al. 2004; Leung and Chin 2005) and those that take into account correlated positions within a motif (Barash et al. 2003; Zhou and Liu 2004).

We first give a formulation that allows mismatches but does not allow don't cares. Let $S = \{s_1, \ s_2, \ \ldots, \ s_k\}$ be a sample of k sequences each of length n and let l be the

length of a motif s. We put A and T together in a group and G and C together in another group. Let a be the number of A or T in s (thus $l-a$ is the number of G or C in s). Let p_1 be the probability of finding an A in the sample (which is the same as the probability of finding a T), and let p_2 be the probability of finding a C in the sample (which is the same as the probability of finding a G). The probability of s occurring with up to d substitutions at a given position of a random sequence is given by

$$p(l,a,d) = \sum_{i=0}^{d} \sum_{j=\max(0,a+i-l)}^{\min(a,i)} \binom{a}{j}\binom{l-a}{i-j}(1-p_1)^j {p_1}^{a-j}(1-p_2)^{i-j}{p_2}^{l-a-i+j} \quad ,$$

where j counts the number of substitutions within A or T positions while i counts the total number of substitutions. To compute the p-value for s, denote the distance between s and sequence s_i by $d(s,s_i) = \min\{d(s,s') | s' \in s_i\}$, where s' is a string of length l appearing in s_i and $d(x,y)$ is the distance (number of substitutions) between two strings x and y of length l. Fix a maximum distance d and let k' be the number of sequences s_i with $d(s,s_i) \le d$. The p-value of s with respect to d is given by

$$p(l,a,d,k') = \sum_{i=k'}^{k} \binom{k}{i}(1-(1-p(l,a,d))^{n-l+1})^i((1-p(l,a,d))^{n-l+1})^{k-i} \quad ,$$

which is an estimate of the probability of s occurring at least once with up to d substitutions in at least k' sequences when complex correlations between overlapping patterns are ignored. Note that, for simplicity, this equation only takes into account at most one motif occurrence in each sequence. We then estimate the e-value of s with respect to d by

$$e(l,a,d,k') = 4^l p(l,a,d,k') \ .$$

This equation ignores differences in the nucleotide composition of motifs which may not have comparable values of a, d and k' (but still takes into account the background nucleotide distribution) and assumes that $p(l,a,d,k')$ is the probability one wishes to attain for all motifs of length l. The above equations are generalizations of the equations in Buhler and Tompa (2002) to allow for biased background distribution and some of the sequences not having a motif occurrence. We define the *e*-value of s to be the minimum *e*-value over all d. The goal of the motif finding problem is to find all motifs s with *e*-value below a cutoff, and the occurrences of s are defined by finding the value of d that minimizes the e-value of s and recovering all occurrences in the sample that are within distance d of s (there can be more than one occurrence in some sequences). In difference from many other approaches that assume that d is given in advance (Marsan and Sagot 2000; Pevzner and Sze 2000; Pavesi et al. 2001; Buhler and Tompa 2002; Eskin and Pevzner 2002; Keich and Pevzner 2002; Price et al. 2003), our formulation does not assume that a fixed d is given and will automatically find the best value of d for each motif s independently.

To allow for don't care positions within a motif s, let l be the length of s and l' be the number of positions within s that contain a nucleotide character (i.e., there are $l - l'$ don't care positions). A string s' of length l that appears in the sample is defined to be an occurrence of s if the total number of substitutions within these l' positions is at

most d while ignoring the other $l - l'$ don't care positions. To estimate the statistical significance of a motif s, the *p*-value of s with respect to d is given by

$$p(l, l', a, d, k') = \sum_{i=k'}^{k} \binom{k}{i} (1 - (1 - p(l', a, d))^{n-l+1})^i ((1 - p(l', a, d))^{n-l+1})^{k-i} ,$$

where $p(l', a, d)$ is the same as before with l' substituting l. Since there is no need to allow don't cares at the two ends of s, the *e*-value of s with respect to d is given by

$$e(l, l', a, d, k') = \binom{l-2}{l'-2} 4^{l'} p(l, l', a, d, k') .$$

To allow don't cares while not allowing mismatches, simply set $d = 0$ in the above equations. Note that the notion of don't cares we use here is very different from the one in Buhler and Tompa (2002) since they used don't care positions to randomize their search procedure rather than defining motifs.

C. Algorithm when Mismatches are Allowed

We first develop an improved pattern-driven algorithm that allows mismatches but does not allow don't cares. The original pattern-driven approach considers each candidate motif in turn and looks for its occurrences by comparing it to every string of length l in the sample. To avoid these extensive comparisons, we encode each nucleotide by two bits and create an array D of size 4^l and a queue Q of size 4^l. Our algorithm consists of two stages: the first stage computes all $d(s, s_i)$ between each candidate motif s and

each sequence s_i (to be stored in D and reused for each i). We accumulate this information in another $4^l \times l$ array N which stores for each candidate motif s, the number of times that $d(s, s_i) = d$ for each d. The second stage computes the *e*-value of each candidate motif s from *N*. The first stage iterates over each sequence s_i and starts by initializing all values in *D* to l (Figure 2.1). For each string s appearing in s_i, set *D(s)* to 0 and insert s into Q. Repeat the following procedure that employs a depth-first search strategy: remove the first element s from Q and generate all neighbors s' of s that are one substitution away from s. For each s', if $D(s') > D(s)+1$, update $D(s')$ to $D(s)+1$ and add s' to Q (Figure 2.1). It is easy to see that when Q becomes empty, we have $D(s) = d(s, s_i)$ for all s. It is easy to see that it takes $(n-l+1)$ time to find all substrings in s_i and takes at most $(4^l \cdot 3l)$ to process all possible elements from Q, as each s appears at most once in Q and there are $3l$ strings that are one substitution away from s. Thus the total time to process each sequence s_i is $(n-l+1+4^l \cdot 3l) = O(4^l l)$, assuming that $n < 4^l$. As the processing of each sequence s_i is completed, the values in $D(s)$ are transferred to *N*. The second stage uses the values in *N* to compute the *e*-value of each candidate motif s (Figure 2.1). Since the binomial coefficients and the probability values can be preprocessed and stored in such a way that each *e*-value $e(l, a, d, k')$ can be obtained in constant time and the preprocessing time is negligible (polynomial in n and l), the entire procedure takes $O(4^l lk)$ time and $O(4^l l)$ space when l is large enough. Note that the assumption $n < 4^l$ is easily satisfied: with n as large as 2000, only $l > 5$ is needed.

```
Algorithm MotifEnumerator(l) {
  Q ← empty; for each (s, d) do { N(s, d) ← 0; }
  for each sequence s_i do {
    for each s do { D(s) ← l; }
    for each s ∈ s_i do { D(s) ← 0; insert s to the end of Q; }
    while Q is not empty do { remove s from the front of Q;
      for each s' with d(s, s') = 1 do {
        if D(s') > D(s) + 1 then {
          D(s') ← D(s) + 1; insert s' to the end of Q; } } }
    for each s do { N(s, D(s)) ← N(s, D(s)) + 1; } }
  for each s do { k' ← 0;
    for d ← 0 to l do { k' ← k' + N(s, d);
      compute e(l, a, d, k'), where a is the number of A or T in s; } } }
```

Figure 2.1: Algorithm MotifEnumerator for finding the e-values of all candidate motifs s of length l when mismatches are allowed but don't cares are not allowed.

When implemented carefully, it is possible to store all the arrays in 4G memory when l is 12 or 13 (which works on 32-bit systems). To further save memory, observe that since the values of $D(s)$ in Q are increasing, we can eliminate Q and replace it by a loop that generates neighbors s' only for those s with $D(s) = j - 1$ in iteration j. This strategy does not change the time complexity since neighbors are generated for at most 4^l strings over l iterations. Also, our approach can scan through all candidate motifs of length at most l with not much increase in running time (at most 4/3 times longer) when compared to checking only one l. In difference from many other approaches, there is no implicit restriction on the minimum number of motif occurrences or on the maximum

distance d between a motif and its occurrences. The algorithm explores all the possibilities to guarantee that the motif with the best *e*-value is found. When d is given, the above procedure can be used to implement the extended pattern-driven approach by stopping the first stage when the first element s in Q has $D(s) = d$ for each sequence s_i, resulting in a saving of a factor of n over the straightforward approach. Note that our neighbor generation process is similar to the one in Blanchette et al. (2002) except that their computation is based on a phylogenetic tree. Our procedure also has some similarity to the one in Price et al. (2003) except that our approach is exact and their approach is a heuristic.

We extend our algorithm to allow arbitrary don't care positions within a motif s. Since there is no need to allow don't cares at the two ends of s, a straightforward algorithm to enumerate all possible s of length l uses an array D of size $4^2 5^{l-2}$ to represent each s. For each sequence s_i, consider each string s' that appears in s_i and set $D(s) = 0$ for each of the 2^{l-2} possible strings s that can be generated from s' while allowing don't care positions. Then proceed in the same way as before while ignoring don't care positions during the neighbor generation process, resulting in an algorithm that takes $O(5^l lk)$ time and $O(5^l l)$ space. Alternatively, the following algorithm only needs $O(4^l l)$ space while having the same time complexity: for each value of s' and each way of choosing l' positions from l positions (while always choosing the two end positions), treat each string of length l with l' chosen positions as a string containing only the l' chosen positions and apply the original procedure on strings of length l'. Its time

complexity can be estimated more precisely as $O(\sum_{l'=1}^{l}\binom{l-2}{l'-2}4^{l'}l'k)$. When l is small (e.g., $l \leqslant 12$), the running time to consider motifs of length at most l with don't cares is similar to the original algorithm that considers motifs of length at most $l+1$ without don't cares, thus the modified strategy does not have a large effect on solvable values of l ($l \leqslant 11$ or 12 are solvable in reasonable time).

D. Algorithm when Mismatches are Not Allowed

We first give an algorithm that takes $O(lkn)$ time and space when both mismatches and don't cares are not allowed. Under these assumptions, each string s of length l that appears in the sample represents a candidate motif. We store these strings in a tree T of height l so that each s is represented by a path of length l from the root. Each internal node t of T can have at most four children $t.c$, one for each character c of the DNA alphabet, with the path from the root to t representing a prefix of one or more motifs; while each leaf node t of T represents a unique motif s, with $t.k'$ denoting the number of sequences that s occurs in (only at most one occurrence is counted in each sequence) and $t.i$ denoting the sequence number of the previous occurrence of s during the tree construction (Figure 2.2). To allow for arbitrary don't care positions, for each value of l' and each way of choosing l' positions from l positions (while always choosing the end positions), treat each string of length l with l' chosen positions as a string containing only the l' chosen positions and build a tree T of height l'.

The entire procedure takes $O(2^l lkn)$ time and $O(lkn)$ space, thus by disallowing mismatches, we extend the solvable values of l to around 20 (from around 12 when mismatches are allowed). Also, our approach can scan through all candidate motifs of length at most l with not much increase in running time (at most twice longer) when compared to checking only one l. Although the above procedure can be quite successful in identifying core motif occurrences, the requirement that each occurrence must be exactly the same except for the don't care positions is very strict, thus it is likely that some motif variants are missed. We use the following strategy to allow for a limited number of mismatches while avoiding the introduction of many false positives: let s be a motif of length l with m occurrences $o_1,...,o_m$ each of length l (there can be more than one occurrence in some sequences). We construct a refined motif s' as follows: for each position j, if there exists a nucleotide character c such that its total frequency at the *jth* position within the m occurrences is more than $m/2$, set the *jth* character of s' to c, otherwise set it to a don't care character (note that c is uniquely defined if it exists). Let $d' = \max\{d(s', o_i) | 1 \le i \le m\}$, where the don't care positions in s' are ignored to c, otherwise set it to a don't care character (note that c is uniquely defined if it exists). Let $d' = \max\{d(s', o_i) | 1 \le i \le m\}$, where the don't care positions in s' are ignored when computing distances. We define the occurrences of s' to be all strings of length l that appear in the sample and are within distance d' of s'. Note that this new set of occurrences of s' must include the original occurrences of s.

```
Algorithm MotifEnumerator(l) {
  T ← root with no children;
  for each sequence s_i do {
    for each s ∈ s_i do { t ← root of T;
      for j ← 1 to l do { c ← jth character of s;
        if t.c does not exist then { t.c ← new node with no children; t ← t.c;
          if j = l then { t.k' ← 0; t.i ← −1; } }
        else { t ← t.c; } }
      if t.i ≠ i then { t.k' ← t.k' + 1; t.i ← i; } } }
  for each leaf t of T do {
    compute e(l, a, 0, t.k'), where a is the number of A or T in the motif in t; }
```

Figure 2.2: Algorithm MotifEnumerator for finding the e-values of all candidate motifs *s* of length l when both mismatches and don't cares are not allowed.

E. Performance

1. Yeast Test Samples

To show that our model is reasonable and the *e*-values are comparable over different motif lengths, we first test our algorithm MotifEnumerator on artificial samples with 20 sequences each of length 600 containing an (*l, d*)-motif (Pevzner and Sze 2000), which is a motif of length l with d substitutions between the motif and its occurrences. In each case, we check all candidate motifs of length at most 12 with no implicit assumption on the minimum number of motif occurrences in very difficult (8, 2)-, (10, 2)- and (12, 3)-motifs. In each case, the motif found was always of the correct length and

the correct motif always had the best *e*-value. To ensure that MotifEnumerator can identify biological motifs, we test it on a large set of yeast samples constructed from co-expressed gene clusters from Tavazoie et al. (1999) and compare our results with those in Tavazoie et al. (1999) and Hughes et al. (2000). To allow for samples having sequences of similar but unequal lengths, we use the average sequence length to approximate n. To allow for motifs to appear in the reverse complementary direction, we assume that each sequence s_i is twice as long including both the forward and the reverse complementary sequences and replace the term $n-l+1$ by $2(n-l+1)$ in the *p*-value formulas. We further preprocess each input sample by removing low complexity repeats using very simple rules. To find a set *M* of suboptimal motifs that are sufficiently different from each other, we first discard all motifs with *e*-value above a cutoff. With *M* initially empty, consider each remaining motif s in increasing order of *e*-value and repeat the following: add s to *M* if there are no overlaps between its occurrences and any motif occurrences already in *M*.

This procedure finds a set of suboptimal motifs in one single run and it takes negligible time when compared to the previous stage since not many candidate motifs remain after the *e*-value cutoff is applied. For each cluster in Tavazoie et al. (1999), we extract upstream sequences of length 600 resulting in a total of 30 samples, each having from 50 to 200 sequences with a nucleotide bias of around 60% A or T and 40% G or C. We run our algorithm MotifEnumerator over all motif lengths $l \le 12$ and allow motifs to appear in the reverse complementary direction. The running time ranges from hours for the

smaller samples to days for the larger samples. Table 2.1(a) shows all strong motifs found, while Table 2.1(b) shows a small subset of weaker motifs that are known biological motifs. Our algorithm found almost all the motifs in Tavazoie et al. (1999) and was able to identify an extra Rpn4 motif that is absent in their paper (although its *e*-value is not very low, it appears in more than 20 sequences). This motif was identified in Hughes et al. (2000) when a different strategy of grouping genes by common names was used to construct samples. Some of the motifs were found in a different cluster from the one specified in Tavazoie et al. (1999), including M14a (found in cluster 2) and M4 (found in cluster 16). Although they did not find any motifs in cluster 16, we found variants of M3a/M4 and M3b in cluster 16. Two motifs listed in their paper were missing from our results, including M14b and STRE that have repeating letters and were probably eliminated during the removal of low complexity repeats.

One important observation from Table 2.1 is that for almost all the motifs found, the maximum distance d that minimizes the *e*-value was 0. The only strong motif found in Table 2.1(a) with $d = 1$ was Rap1, but another variant of it was also found with $d = 0$. Two motifs M1a and Rpn4 were found in Table 2.1(b) with $d = 1$, but they are very weak and may not be distinguishable from noise. This suggests that the most biological motifs can be represented accurately by invariant or almost invariant positions within the motif, which motivates an alternative formulation that disallows mismatches when arbitrary don't cares are allowed. With this restriction, the problem becomes easier to solve and longer motifs can be considered. To improve the sensitivity in finding

plausible motif occurrences, a limited number of mismatches can be allowed by adding a post-processing step to refine the initial motifs.

Table 2.1: Performance of MotifEnumerator on 30 samples constructed from co-expressed yeast gene clusters from Tavazoie et al. (1999). (a) All strong motifs found by MotifEnumerator on 30 samples constructed from co-expressed yeast gene clusters from Tavazoie et al. (1999). These motifs appear in at least 10 sequences with e-value below 10–5, where cl# denotes the cluster number, *d* denotes the maximum distance (between a motif and its occurrences) that minimizes the e-value, and don't care positions are denoted by '-'. All these motifs correspond to known biological motifs, as shown in notes. (b) A small subset of weaker motifs that are known biologically. Some of these motifs have higher e-values than over 10 other non-overlapping candidate motifs within the same run (these suboptimal motifs do not overlap with each other). M3a/M4 and Cbf1p appear in less than 10 sequences.

(a)

cl#	motif	*d*	*e*-value	notes
1	acatccgtacat	1	1.12e–25	Rap1
1	accca-acat-t	0	6.31e–10	Rap1
2	acgcgt-a	0	2.24e–22	MCB
2	tttcgcg	0	2.16e–09	SCB/M14a
3	gatgagatgag	0	1.44e–16	M3b
3	cgatgagc	0	9.76e–08	M3b
7	tttcc-aa---g	0	5.56e–08	ECB
8	ttcttg	0	7.20e–06	SCB
16	gcgatgag-t	0	1.15e–16	M3b
16	tgaaaaattt	0	7.58e–06	M3a/M4

(b)

cl#	motif	*d*	*e*-value	notes
1	tttctcact-t	1	3.18e–02	M1a
2	ttcttg	0	8.47e–05	SCB
2	tggcaaaatg	1	2.35e–03	Rpn4
3	tgaaaaatttt	0	1.41e–05	M3a/M4
7	c-aaa--gg-aa	0	7.99e–04	ECB
30	gtcacgtgc	0	1.20e–03	Cbf1p

2. Tompa Benchmark Test Samples

We test the no-mismatch version of MotifEnumerator on a large benchmark set of samples from Tompa et al. (2005), each having up to 35 sequences with sequence lengths ranging from 500 to 3000. Since many biological motifs in the test set contain moderately repeating patterns, we use a less extensive procedure than before to remove low complexity repeats that include single-nucleotide repeats of length at least six, two-nucleotide repeats with at least four repeating units, and three-nucleotide repeats with at least three repeating units, with no mismatches allowed within the repeats. We run MotifEnumerator over all motif lengths $l \leq 20$ and look for motifs only on the forward strand. In each case, the refined occurrences of the top motif with *e*-value below 1.0 are used for evaluation (it is possible that no motif is found). The running time ranges from hours for the smaller samples to days for the larger samples. Table 2.2 shows the performance of MotifEnumerator on both the mixed set of samples that was assessed in Tompa et al. (2005) and on the original three sets of samples of type real, generic and markov from which the mixed set is derived but were not assessed in Tompa et al. (2005).

On the mixed set, the overall performance of MotifEnumerator (with nCC=0.067) was roughly comparable to algorithms assessed in Tompa et al. (2005) that had overall performance ranging from above average to near-best, including AlignACE (Hughes et al. 2000) with nCC=0.068, MotifSampler (Thijs et al. 2001) with nCC=0.068, MEME

(Bailey and Elkan 1994) with nCC=0.073, Oligo/Dyad (van Helden et al. 1998; van Helden et al. 2000) with nCC=0.071, and ANN-Spec (Workman and Stormo 2000) with nCC=0.074. Only two algorithms definitely performed much better, including YMF (Sinha and Tompa 2000) with nCC=0.084 and Weeder (Pavesi et al. 2001) with nCC=0.156. Within the mixed set, MotifEnumerator followed a similar trend as most other algorithms, with better performance on samples of type generic and markov and worse performance on samples of type real.

In particular, on samples of type real, MotifSampler (Thijs et al. 2001) with nCC=0.076 and Weeder (Pavesi et al. 2001) with nCC=0.077 performed best among all the assessed algorithms, while YMF (Sinha and Tompa 2000) with nCC=0.013 performed much worse than MotifEnumerator with nCC=0.046. Overall, Weeder (Pavesi et al. 2001) had the best performance that was much higher than all the other assessed algorithms. When the samples were categorized by the organism from which the upstream sequences are obtained, MotifEnumerator also followed a similar trend as most other algorithms, with the best performance on yeast samples, medium performance on human and mouse samples and worst performance on fly samples.

Table 2.2: Performance of MotifEnumerator on benchmark test samples from Tompa et al. (2005) when arbitrary don't care positions are allowed but mismatches are not allowed. Each entry represents the nucleotide-level correlation coefficient (nCC) computed by comparing the refined occurrences of the top motif returned from MotifEnumerator (if one exists) to the known annotation in each sample and treating a subset of samples as if it was a single large sample. Each row represents a set of 56 samples (except for the set of type real, which contains 52 samples). Each set of type real, generic or markov contains motifs corresponding to one transcription factor with a particular type of background sequences. Tompa et al. (2005) did not perform assessments directly on these sets, but constructed another set of type mixed with 56 samples by picking one background type for each transcription factor (out of a total of two or three possibilities) so that samples within this set may have different background types. Assessments were performed only on this mixed set in Tompa et al. (2005), which corresponds to the row and the column labeled mixed, while ignoring the other 108 samples from the original sets. Each set is further subdivided into four subsets according to the organism from which the upstream sequences are obtained (except for the mixed subset, which contains samples of a particular type within the entire mixed set).

	mixed	fly	human	mouse	yeast	overall
mixed		−0.010	0.040	0.043	0.238	0.067
real	0.046	0.075	0.025	0.058	0.214	0.063
generic	0.091	−0.010	0.014	0.046	0.335	0.065
markov	0.065	−0.010	0.051	0.052	0.188	0.073

We also analyze the performance of MotifEnumerator on the original three sets of samples of type real, generic and markov from which the mixed set is derived. Each of these original sets contains about the same number of samples as the entire mixed set (Table 2.2). The most noticeable advantage of MotifEnumerator is that similar overall performance was obtained across all these original sets with distinct background types and thus MotifEnumerator does not seem to be affected much by differences in the background sequences. Also, there was a significant increase in the performance of MotifEnumerator on the fly samples within the real set, which is mainly due to a strong result on the dm01r sample (this sample was not assessed in Tompa et al. (2005)). Interestingly, Tompa et al. (2005) also reported that MotifSampler (Thijs et al. 2001) had similar performance over different background types within the mixed set and SeSiMCMC (Favorov et al. 2005) had strong performance on the fly samples within the mixed set (although SeSiMCMC (Favorov et al. 2005) had weak overall performance).

F. Discussion

Since allowing mismatches may still provide better sensitivity in some cases, both variants of MotifEnumerator are useful in different situations. The main advantage of allowing mismatches is that a one-step process can be used to guarantee that the optimal motif is found while automatically allowing appropriate variations if the resulting statistical evaluation is favorable. The time complexity of our algorithm contains an exponential factor and is independent of the length of each sequence n for large enough

l. Thus it is useful in most situations when the goal is to identify the conserved core region of a promoter.

When mismatches are not allowed, the search space is much smaller and it becomes possible to develop an algorithm with a much smaller exponential factor in the time complexity that only needs polynomial space instead of exponential space, thus allowing longer motifs to be considered while still guaranteeing that the optimal motif pattern is found. Although the tests above show that the algorithm is not very fast when l is around 20, it is extremely fast when l is small. For example, it takes seconds to run the algorithm for the smaller samples in Tompa et al. (2005) and minutes to hours for the larger samples over $l \leq 10$ or 12. To avoid missing important motif occurrences, an additional step has been introduced to find plausible motif occurrences while allowing limited mismatches. Although we have used a strict definition in this step to avoid introducing many false positives, it is also possible to use less strict definitions to allow more occurrences to be identified. In spite of the seemingly restrictive motif definition in disallowing mismatches initially, our algorithm does not seem to lose much sensitivity when compared to most other algorithms assessed in Tompa et al. (2005) that use more general motif models. Only Weeder (Pavesi et al. 2001) consistently performed much better than MotifEnumerator in almost all situations.

To further improve the algorithms, it may be desirable to allow a small amount of overlaps among suboptimal motif occurrences to avoid missing motifs. It is also

important to develop more accurate statistical formulas for samples that do not have sequences of similar lengths and for motifs with more than one occurrence per sequence. This has to be done very carefully since assigning scores that correspond to many occurrences on a sequence may not necessarily lead to an increase in sensitivity due to the larger flexibility that allows many other candidate motifs to have better scores. To further improve performance, it may be desirable to incorporate genome-specific information by using the overall genome nucleotide distribution, probably only in the non-coding regions, to serve as the background distribution. In many situations, there may be a need to find motifs that are significant in one sample but not in the other. This can be addressed by extracting motifs in one sample that have a good likelihood ratio with respect to another negative sample.

CHAPTER III

ALGORITHMS BASED ON SKIPPING NONCONSERVED POSITIONS IN BACKGROUND MARKOV CHAINS

One strategy to identify transcription factor binding sites is through motif finding in upstream DNA sequences of potentially co-regulated genes. Despite extensive efforts, none of the existing algorithms perform very well. We consider a string representation that allows arbitrary ignored positions within the non-conserved portion of single motifs, and use $O(2^l)$ Markov chains to model the background distributions of motifs of length l while skipping these positions within each Markov chain. By focusing initially on positions that have fixed nucleotides to define core occurrences, we develop an algorithm that is efficient enough to identify motifs of moderate lengths. We compare the performance of our algorithm to other motif finding algorithms on a few benchmark data sets, and show that significant improvement in accuracy can be obtained when the sites are sufficiently conserved within a given sample, while comparable performance is obtained when the site conservation rate is low.

A software program implementing this method (PosMotif) is available at http://faculty.cse.tamu.edu/shsze/posmotif.

A. Introduction

One important application of motif finding is the identification of transcription factor binding sites from upstream DNA sequences of potentially co-regulated genes, in which the most popular approaches either represent a motif by a positional weight matrix and use statistical optimization techniques to identify the most overrepresented patterns (Stormo and Hartzell, 1989; Lawrence et al.,1993; Bailey and Elkan, 1994; Thijs et al., 2001), or represent a motif by a string and use combinatorial techniques to identify frequent patterns (Queen et al., 1982; Waterman et al., 1984).

In addition to using information from the given upstream sequences, recent approaches utilize additional information, including the use of evolutionary relationships between orthologous upstream sequences through the phylogenetic footprinting technique (Blanchette et al.., 2002), the inclusion of negative samples to define discriminative motifs (Sinha, 2003), and the use of binding energy models and structural knowledge (Kaplan et al., 2005; Leung et al., 2005). To investigate the relationships between motifs, the single motif finding problem has also been generalized to the identification of composite motifs and cis-regulatory modules (Marsan and Sagot, 2000; van Helden et al., 2000; GuhaThakurta and Stormo, 2001; Liu et al., 2001; Eskin and Pevzner, 2002).

While most approaches that use the string representation either allow mismatches (Pevzner and Sze, 2000; Pavesi et al., 2001; Buhler and Tompa, 2002) or use degenerate

letters (Sinha and Tompa, 2002; Peng et al., 2006), other approaches also allow positions within a motif to be ignored, either by allowing spacers between motif segments (Sinha and Tompa, 2002) or by imposing density constraints to restrict the number of ignored positions (Wijaya et al., 2007). To improve motif finding accuracy, recent approaches incorporate nucleotide dependencies within motifs (Barash et al., 2003; Zhou and Liu, 2004; Chin and Leung, 2008).

We consider a string representation that allows arbitrary ignored positions within the nonconserved portion of single motifs of length l. For each combination of ignored positions, we use aMarkov chain to model the background distribution while skipping these positions, resulting in a total of $O(2^l)$ Markov chains that can model long range nucleotide dependencies. This approach is more general than using a single positional weight matrix or using a single string to model a motif.

To obtain an algorithm that is efficient enough to identify motifs of moderate lengths, we focus initially on positions that have fixed nucleotides to define core occurrences. This is based on the biological motivation that there are often almost invariant positions that are critical for the binding process. While most approaches do not specifically take advantage of these positions, our model tries to capture them within positions that are not ignored.

We compare the performance of our algorithm to other motif finding algorithms on a few benchmark data sets, and show that significant improvement in accuracy can be obtained even without extensive post processing when the sites are sufficiently conserved within a given sample, while comparable performance is obtained when the site conservation rate is low. We also perform additional post processing to improve the modeling of motifs.

B. Problem Formulation

We represent a motif of length l by a string $S = s_1 s_2 \cdots s_l$ in the alphabet {a,c,g,t,–}, where – represents an ignored position, with $s_1 \neq -$ and $s_l \neq -$. For a given sample *S* of sequences in the alphabet {a,c,g,t}, define the occurrences of *s* to be all strings of length l in *S* that match *s* in all positions, where – matches any letter in {a,c,g,t}. Thus each ignored position represents a potentially non-conserved position that is ignored in the motif modeling (see Figure 3.1 for an example motif that is represented by the string s = cgg----ct-t-g--cg).

For a particular combination of ignored positions within a string of length l out of $O(2^l)$ possibilities, we construct an m th order Markov chain *M* by skipping these positions. For each string *s* of length l, let $s' = s'_1 s'_2 \cdots s'_{l'} = s_{i_1} s_{i_2} \cdots s_{i_{l'}}$ be the string of length l' obtained from *s* by removing the ignored positions. Define the set of states of *M* as

$$\{(s'_j s'_{j+1} \cdots s'_{j+m-1} = w) \quad | 1 \le j \le l' - m + 1, w \in \{a,c,g,t\}^m\}\}$$

and create a transition from the state $(s'_j\, s'_{j+1} \cdots s'_{j+m-1} = w_1 w_2 \cdots w_m)$ to the state $(s'_{j+1}\, s'_{j+2} \cdots s'_{j+m} = w_2 w_3 \cdots w_{m+1})$.

occurrence 1	cggcggtctttcgtccg
occurrence 2	cggagcactgttgagcg
occurrence 3	cggacaactgttgaccg
occurrence 4	cggcgcactctcgcccg
motif	cgg----ct-t-g--cg

Figure 3.1: Example of representing four occurrences of the Gal4 binding site in the yeast sample yst02r from Tompa et al. (2005) by a motif with eight ignored positions (represented by -). The known consensus of the Gal4 binding site is cggnnnnnnnnnnnccg (Sinha and Tompa 2002).

We can visualize *M* as a leveled structure in which each row represents all states with the same w and the j th column represents the j th level that contains all states with the same j (see Figure 3.2). For a given background sample *BS* of sequences in the alphabet {a,c,g,t} and the association $s'_j\, s'_{j+1} \cdots s'_{j+m} = w_1 w_2 \cdots w_{m+1}$, we estimate the transition probabilities by

$$P(s'_{j+1} s'_{j+2} \cdots s'_{j+m} \mid s'_j s'_{j+1} \cdots s'_{j+m-1}) = \frac{\text{numberof occurrences of } s_{i_j} s_{i_{j+1}} \cdots s_{i_{j+m-1}} s_{i_{j+m}} \text{ in BS}}{\sum_{x \in \{a,c,g,t\}} \text{numberof occurrences of } s_{i_j} s_{i_{j+1}} \cdots s_{i_{j+m-1}} x \text{ in BS}}$$

where $s_{i_j} s_{i_{j+1}} \cdots s_{i_{j+m-1}}$ is a substring of the original string s that includes the ignored

positions. Note that when ignored positions are not allowed, this Markov background model is similar to the one used by other motif finding algorithms (Bailey and Elkan, 1994; Pavesi et al.., 2001; Thijs et al., 2001; Sinha and Tompa, 2002).

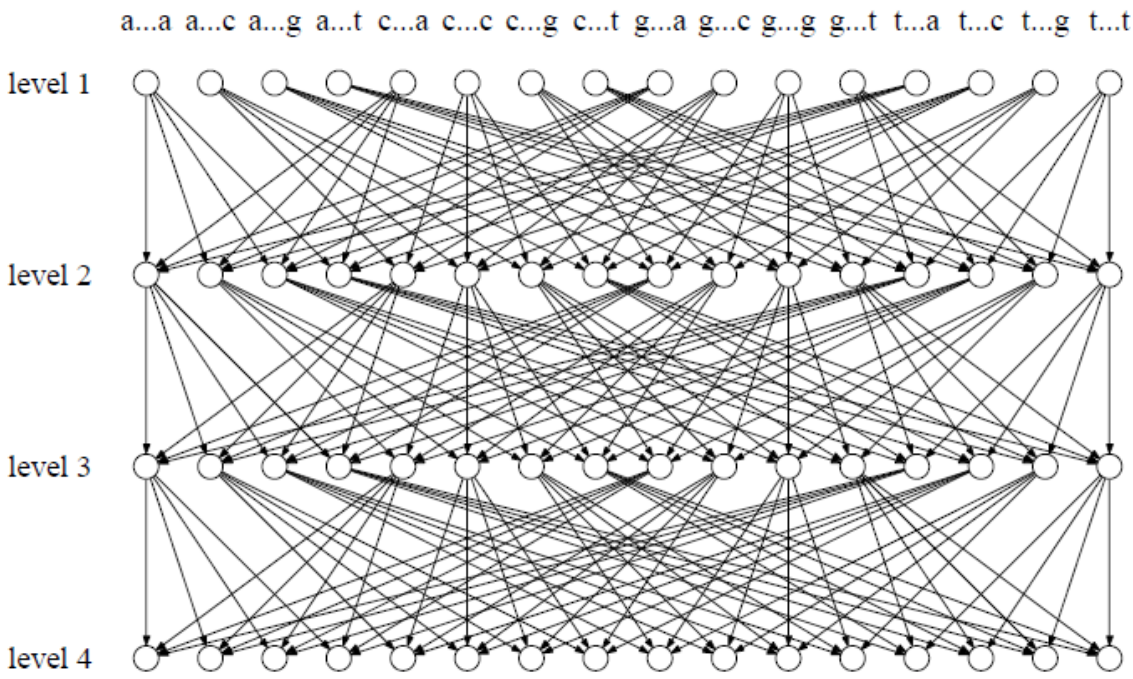


Figure 3.2: Illustration of a 2nd order Markov chain M for strings of length l with $l'=5$ positions that are not ignored, represented by $s'_1 s'_2 \cdots s'_5$ after removing the ignored positions. The states of M are of the form ($s'_j s'_{j+1} = w$), with the jth row representing the jth level that contains all states with the same j and each column representing all states with the same w. Each column is labeled by a particular combination of values of s'_j and s'_{j+1} as $s_{i_j} \cdots s_{i_{j+1}}$, with potentially different number of ignored positions between them in the original string $s = s_1 s_2 \cdots s_l$ for different j.

Let S be a sample of k sequences each of length n that represent upstream DNA sequences of potentially co-regulated genes. By ignoring correlations between overlapping occurrences, we estimate the probability of s occurring at a given position of S by

$$P(s'_1 s'_2 \cdots s'_m) = \frac{\text{number of occurrences of } s_{i_1} s_{i_1+1} \cdots s_{i_m-1} s_{i_m} \text{ in } BS}{\text{total number of strings of length } i_m - i_1 + 1 \text{ in } BS}$$

and $s_{i_1} s_{i_{11}+1} \cdots s_{i_m-1} s_{i_m}$ is a substring of s that includes the ignored positions. The probability of s occurring at least n' times in a sequence is estimated by

$$P(s, n') = 1 - \sum_{i=0}^{n'-1} \binom{n-l+1}{i} P(s)^i (1 - P(s))^{n-l+1-i}.$$

Let k' be the number of sequences that s occurs at least n' times. We estimate the P-value of s by the probability of s occurring at least n' times in at least k' sequences as

$$P(s, n', k') = 1 - \sum_{i=0}^{k'-1} \binom{k}{i} P(s, n')^i (1 - P(s, n'))^{k-i}.$$

Since positions at the two ends of s are never ignored, we estimate the E-value of s by

$$E(s, n', k') = \binom{l-2}{l'-2} 4^{l'} P(s, n', k').$$

By assuming that $P(s,n',k')$ is the probability to be attained for all motifs of length l that have l' positions that are not ignored, this equation allows direct comparison of motifs. To allow for samples having sequences of similar but different lengths, we use the average sequence length to approximate n. The goal is to identify all motifs s with

E-value below a cutoff over all combinations of ignored positions and different parameters l and n'.

C. Algorithms

1. Pre-processing

Given a sample *BS* containing background sequences, we first perform preprocessing so that the transition probabilities can be efficiently computed. Given string length l and Markov order m, we compute and store the number of occurrences of all strings of length p from 1 to l in *BS* with $m+1$ positions that are not ignored, by considering each combination of $p-m-1$ ignored positions among p positions and scanning over all strings s of length p that appear in *BS*. We remove the ignored positions from s to obtain a string s' of length $m+1$, and update the number of occurrences of s' that represents the number of occurrences of the corresponding string of length p before removing the ignored positions (Figure 3.3).

Given a sample *S*, by ignoring small differences in occurrences around sequence starts and ends, the initial portion of the probability $P(s)$ of a string $s = s_1 s_2 \cdots s_l$ of length l occurring at a given position of *S* can be estimated by using the approximation

$$P(s'_1 s'_2 \cdots s'_{m+1}) \approx P(s'_1 s'_2 \cdots s'_m) P(s'_2 s'_3 \cdots s'_{m+1} \,|\, s'_1 s'_2 \cdots s'_m)$$

where $s' = s'_1 s'_2 \cdots s'_{l'} = s_{i_1} s_{i_2} \cdots s_{i_{l'}}$ is the string of length l' obtained from s by removing the ignored positions. This simplifies the procedure since there is no need to compute $P(s'_1 s'_2 \cdots s'_m)$ while $P(s'_1 s'_2 \cdots s'_{m+1})$ and $P(s'_{j+1} s'_{j+2} \cdots s'_{j+m} \mid s'_j s'_{j+1} \cdots s'_{j+m-1})$ for each j can be computed from the number of occurrences of the stored strings.

```
Algorithm PreProcess
input: background sample BS with b sequences and the ith sequence of
    length n_i, string length l, Markov order m;
output: number of occurrences of all strings of length at most l in BS with
    m + 1 positions that are not don't care positions;
{
    for p ← 1 to l do {
        for each combination of m + 1 positions that are not don't care
          positions among p positions do {
            for each string s' of length m + 1 in the alphabet {a,c,g,t} do {
                initialize occur(s') to 0 that will represent the number of
                    occurrences of the corresponding string s before removing
                    don't care positions; }
            for i ← 1 to b do {
                for j ← 1 to n_i do {
                    let s = s_1 ··· s_p be the string of length p that starts at
                        position j in the ith sequence of BS;
                    let s' = s'_1 ··· s'_{m+1} be the string of length m + 1 obtained
                        from s by removing don't care positions;
                    occur(s') ← occur(s') + 1; } }
            store all occur(s'); } }
}
```

Figure 3.3: Algorithm to preprocess the background samples.

The procedure takes $O\left((4^m + l\,|BS|)\sum_{p=1}^{l}\binom{p}{m+1}\right)$ time and $O\left(4^m \sum_{p=1}^{l}\binom{p}{m+1}\right)$ space. Although the size of the background sample $|BS|$ is large, it is practical since the Markov order m is small. Note that the memory requirement is lower than the case $O(2^l)$ when the Markov chains are constructed explicitly, and the number of occurrences computed can be reused when considering strings of length larger than l.

2. Algorithm Based on Skipping Non-conserved Positions

Given a sample S and string length l, we consider each combination of ignored positions among l positions, and enumerate all motifs with these ignored positions by scanning over all strings s of length l that appear in S and removing the ignored positions from s to obtain s'. We insert s' into a search tree T, in which each motif is represented by a path from the root to a leaf, each internal node has at most four children that correspond to each letter in {a,c,g,t}, and each leaf represents a motif that may have multiple occurrences(see Figure 3.4).

To avoid repetitive motifs that have excessive number of overlapping occurrences, given a motif s and a parameter n' that specifies the number of occurrences that are counted for each sequence, we let k' be the number of sequences that have at least n' non-overlapping occurrences of s. Within each leaf of T, we compute k' during the tree construction by remembering the number of non-overlapping occurrences found so far in

a sequence and the location of the last occurrence. Although more accurate formulas for computing *P*-values are available for this non-overlapping model (Leung et al.,2005), they have high time complexity and we use our original overlapping approximation to compute *P*-values and *E*-values within each leaf of *T*.

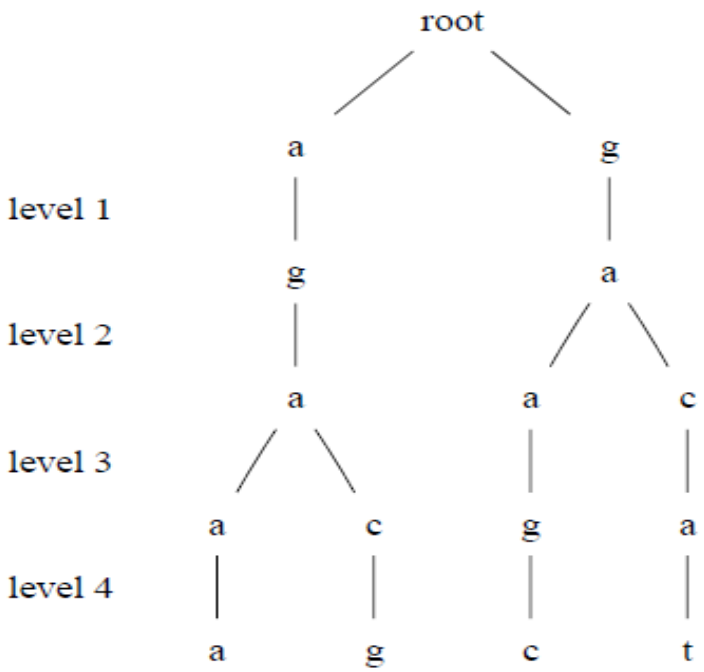


Figure 3.4: Illustration of the search tree T constructed for the sequence aagggaacagtc that stores all motifs of length 9 while ignoring the 2nd, 3rd, 5th and 8th positions, including the motifs a--g-aa-a, a--g-ac-g, g--a-ca-t and g--a-ag-c that appear from the left to the right in the sequence. Each motif is represented by a path from the root to a leaf while skipping these positions. Each horizontally marked level has a corresponding level in Figure 3.2.

```
Algorithm PosMotif
input: sample S with k sequences and the ith sequence of length n_i, string
    length l;
output: all motifs of length l with E-value below a cutoff;
{
    for each combination of don't care positions among l positions do {
        initialize the search tree T to empty;
        for i ← 1 to k do {
            for j ← 1 to n_i do {
                let s = s_1 ··· s_l be the string of length l that starts at position
                    j in the ith sequence of S;
                let s' = s'_1 ··· s'_l' be the string of length l' obtained from s by
                    removing don't care positions;
                insert s' into T; } }
        for each leaf of T do {
            compute the E-value of the corresponding motif; } }
}
```

Figure 3.5: The main PosMotif Algorithm to compute e-values of each candidate motif from the input samples.

For each combination of ignored positions out of $O(2^l)$ possibilities, it takes $O(l\,|S|)$ time and space to construct the search tree T, where $|S|$ is the sample size (Figure 3.5). For each of the $O(|S|)$ leaves that corresponds to one motif s, each of the $O(l)$ terms in the P-value formula for $P(s)$ can be obtained in constant time from the preprocessing results. By computing the binomial coefficients and each term within the summation of the two P-value formulas recursively and obtaining the powers $(1-P(s))^{n-l+1}$ and

$(1-P(s,n'))^k$ by a recursive halving approach, the *P*-values and the *E*-values of *s* can be computed in $O(l+n'+k+\log|S|)$ time. Thus the overall time complexity of the entire algorithm is $O(2^l|S|(l+n'+k+\log|S|))$, which is practical for l up to 18 or 20 and moderate sample size |*S*| (Figure 3.5).

3. Post-processing

We consider all motifs of different lengths up to a maximum l from the main algorithm with *E*-value below a cutoff, and perform initial post-processing (Figure 3.6) by merging pairs of motifs with a shift of at most one starting position. While there exist two motifs with the same number of occurrences in each sequence and the starting position of each occurrence of one motif is one position before the starting position of each occurrence of the other motif, we merge the two motifs into one motif and set its *E*-value to the lower *E*-value among them. While there exist two motifs with the same number of occurrences and the same set of starting positions in each sequence, we remove the motif with the higher *E*-value. Note that the motifs that are merged do not need to have the same or similar lengths, and the motifs after merging may have length larger than l.

To perform this step efficiently, we sort the motifs by the locations of their occurrences and investigate those that are close in locations. At the end, we sort the motifs in increasing order of the E-value and report them. Since not many motifs have low *E*-values, the running time is small when compared to the main algorithm. Note that this

strategy is different from the one in Apostolico and Parida (2004) since we do not remove non-maximal motifs and do not impose density constraints.

```
Algorithm PostProcess
input: set of motifs of different lengths with E-value below a cutoff;
output: set of motifs after initial postprocessing;
{
    while there exist two motifs with the same number of occurrences in
      each sequence and the starting position of each occurrence of one
      motif is one position before the starting position of each occurrence
      of the other motif do {
        merge the two motifs into one motif and set its E-value to the lower
            E-value among them; }
    while there exist two motifs with the same number of occurrences and
      the same set of starting positions in each sequence do {
        remove the motif with the higher E-value; }
    sort the motifs in increasing order of E-value;
}
```

Figure 3.6: Algorithm to post-process the prediction results by merging motifs of same occurrences or strictly consecutive occurrences.

Since previous approaches show that additional post-processing such as using motif redundancy (Pavesi et al., 2001; Wijaya et al., 2007) can lead to improved accuracy, we follow Peng et al. (2006) and use a hybrid ranking strategy to perform further post processing (Figure 3.7). For each motif s from among the top r motifs after the initial post processing step, we compute the number of neighboring motifs s' of s with the

cost of alignment between s and s' below a cutoff, and add the occurrences of s' to s, with overlapping occurrences combined into one site. Note that these added occurrences are possibly of different lengths, resulting in a general motif model, and this step helps to remove our initial restriction that nucleotides must be fixed in positions that are not ignored.

```
Algorithm PostProcess2
input: set of r motifs after initial postprocessing;
output: set of motifs after further postprocessing;
{
    for each motif s do {
        compute the number of neighboring motifs s' of s with the cost of
            alignment between s and s' below a cutoff;
        add the occurrences of s' to s, with overlapping occurrences combined
            into one site; }
    for each motif s do {
        let rank1(s) be the rank of s (between 1 and r) when the motifs are
            sorted in decreasing order of the number of neighboring motifs;
        let rank2(s) be the rank of s (between 1 and r) when the motifs are
            sorted in increasing order of the E-value;
        let rank(s) = rank1(s) + rank2(s); }
    sort the motifs in increasing order of rank;
    for each motif s do {
        remove all motifs s' with lower rank and the percentage of neighbors
            shared by s and s' with respect to s is above a cutoff; }
}
```

Figure 3.7: Algorithm to combine the redundant motifs from the results after the initial post-processing step.

We compute two different ranks for each motif s, including $rank1(s)$, which is the rank of s (between 1 and r) when the motifs are sorted in decreasing order of the number of neighboring motifs, and $rank2(s)$, which is the rank of s (between 1 and r) when the motifs are sorted in increasing order of the E-value, and sort the motifs in increasing order of the hybrid rank $rank(s) = rank1(s) + rank2(s)$. To avoid the situation in which all top motifs are very similar, for each motif s, we remove all motifs s' with worse rank when the percentage of neighbors shared by s and s' with respect to s is above a cutoff.

D. Performance

1. Experiment Setups and Evaluation Criteria

For our algorithm, we consider two variants, including PosMotif1, which combines algorithm PosMotif with algorithm PostProcess that performs initial post processing, and PosMotif2, which combines algorithm PosMotif with algorithms PostProcess and PostProcess2 that perform both initial and further post processing. We compare our performance to YMF (Sinha and Tompa, 2002), which uses a statistical approach that performs very well on samples of type mixed from Tompa et al. (2005), to MEME (Bailey and Elkan, 1994), which is one of the most popular motif finding algorithms that use the expectation maximization strategy, to MotifSampler (Thijs et al., 2001), which uses a Gibbs sampling strategy that performs very well on samples of type real from

Tompa et al. (2005), and to Weeder (Pavesi et al., 2001), which uses a combinatorial approach that has the best accuracy as assessed by Tompa et al. (2005).

For each algorithm, we use the default parameters as much as possible. We follow Sinha and Tompa (2002) and run YMF over motif lengths from 6 to 10, allowing for at most two degenerate symbols and at most 11 spacers for motifs of length 6 and no spacers for motifs of length larger than 6, while using a 3rd order Markov background constructed from upstream sequences of entire species. We further use FindExplanator from YMF to extract independent motifs for each length, and sort these motifs of different lengths by z-score while extracting occurrences on both strands.

We run MEME with the *anr* option, with motif lengths of up to 20 while considering only the forward strand and using a 5th order Markov background constructed from upstream sequences of entire species.

We run MotifSampler 20 times for each motif length 6, 8, 10 and 12 while considering only the forward strand and using a 3rd order Markov background constructed from upstream sequences of entire species.

We run Weeder with the large mode over motif lengths 6, 8, 10 and 12 while allowing sites to be on both strands and using appropriate frequency tables of the given species.

We run PosMotif1 and PosMotif2 using background Markov chains of order $m = 2$ constructed from upstream sequences of entire species while considering only the forward strand. Since it is not necessarily more sensitive to count arbitrary number of motif occurrences in a sequence during the computation of *P*-values, we use a parameter N' to control the maximum number of motif occurrences that are counted for each sequence. We count at most one non-overlapping occurrence per sequence unless the number of sequences k is very small: $N' = 4$ for $k = 1$, $N' = 2$ for $k = 2$ or 3, and $N' = 1$ for $k \geq 4$ k . For each sample *S*, we iteratively consider each possible n' from 1 to N' , which put emphasis on occurrences in different sequences. We restrict the motif length l to at most 18 before post processing and collect all motifs with *E*-value below l for post processing. The above parameters are determined by testing a few combinations and choosing one that gives satisfactory performance on samples of type real from Tompa et al. (2005). For each motif *s*, we define the occurrences of *s* to be all strings in *S* that match *s*, which is independent of n'.

In the second post processing step, we start with top $r = 100$ motifs from the first post processing step. We follow Peng et al. (2006) to define the alignment cost and neighboring motifs as follows: 1 for a mismatch of two letters in {a,c,g,t}, 0.7 for an indel of a letter in {a,c,g,t}, 0.5 for matching – with a letter in {a,c,g,t}, and no cost for other combinations. We allow gaps to appear only at the beginning or the end of an alignment. We consider two motifs to be neighbors only when their difference in length is at most 0.2 times their maximum length, and define the cutoff for alignment cost to be

0.2 times their maximum length. We further define the cutoff for removal of worse ranked motifs to be 50% of shared neighbors. Note that motifs may become longer after initial post processing in PosMotif1 and can contain variable length occurrences after further post processing in PosMotif2.

When constructing background Markov chains, we use background upstream sequences of the same length as the sequence length in a given sample. When processing samples that contain sequences from multiple species, background frequencies are added across multiple species for Weeder, while background upstream sequences are collected together across multiple species before constructing the background Markov chain for the other algorithms.

For each prediction on a given sample, we compute the nucleotide level statistics *nTP*, *nFP*, *nFN* and *nTN*, which are the number of positions that are in both predicted and known sites, the number of positions that are in predicted sites but not in known sites, the number of positions that are in known sites but not in predicted sites, and the number of positions that are not in predicted nor known sites respectively. From these statistics, we compute the sensitivity *nSn*, the positive predictive value *nPPV*, the specificity *nSp*, the performance coefficient *nPC*, and the correlation coefficient *nCC* (see the detailed definition in Chapter I).

By following Tompa et al. (2005) to define an overlap between a predicted site and a known site if they overlap by at least one-fourth of the known site, we also compute the site level statistics *sTP*, *sFP* and *sFN*, which are the number of known sites that have overlap with a predicted site, the number of predicted sites that do not have overlap with known sites, and the number of known sites that do not have overlap with predicted sites respectively. From these statistics, we compute the sensitivity *sSn*, the positive predictive value *sPPV*, and the performance coefficient *sPC* (see the detailed definition in Chapter I).

We use the top motif from each algorithm for performance evaluation. To evaluate the accuracy of each algorithm on a set of samples, we treat it as if it was a single large sample (Tompa et al., 2005). To further evaluate whether our algorithm leads to significant improvements, we use the Wilcoxon matched-pairs signed-ranks test (Wilcoxon, 1947) over a set of samples with $P = 0.05$ as significance cutoff, in which values of *nPC*, *nCC* and *sPC* on each sample within the set are paired from two algorithms.

2. Benchmark Datasets

We test each algorithm on three sets of biological samples, including samples of type real from Tompa et al. (2005), in which each sample contains motifs that correspond to one transcription factor in the TRANSFAC database (Wingender et al., 1996), samples

from the SCPD database (Zhu and Zhang, 1999), which is a promoter database that contains yeast regulons, and samples from the ABS database (Blanco et al., 2006), in which each sample contains experimentally validated binding sites that have been manually curated from at least two orthologous vertebrate promoters.

Figure 3.8 shows performance comparisons of the algorithms on samples of type real from Tompa et al. (2005), in which there are a total of 52 samples from four species, including fly, human, mouse and yeast, with each sample containing up to 35 upstream sequences from one species and sequence lengths ranging from 500 to 3000. Note that these samples contain real upstream sequences, which are different from the samples of type mixed used in Tompa et al. (2005). When all samples from different species are considered together, the *P*-values from the Wilcoxon test show that there are no significant performance differences between PosMotif and the other algorithms. When the samples from each species are considered separately, Table 3.1 shows that there are considerable accuracy fluctuations. This is especially true for fly, which contains only six samples. All the algorithms have high accuracy on yeast, with PosMotif generally performing better on yeast. The Wilcoxon test is not performed within each species since the number of samples is small.

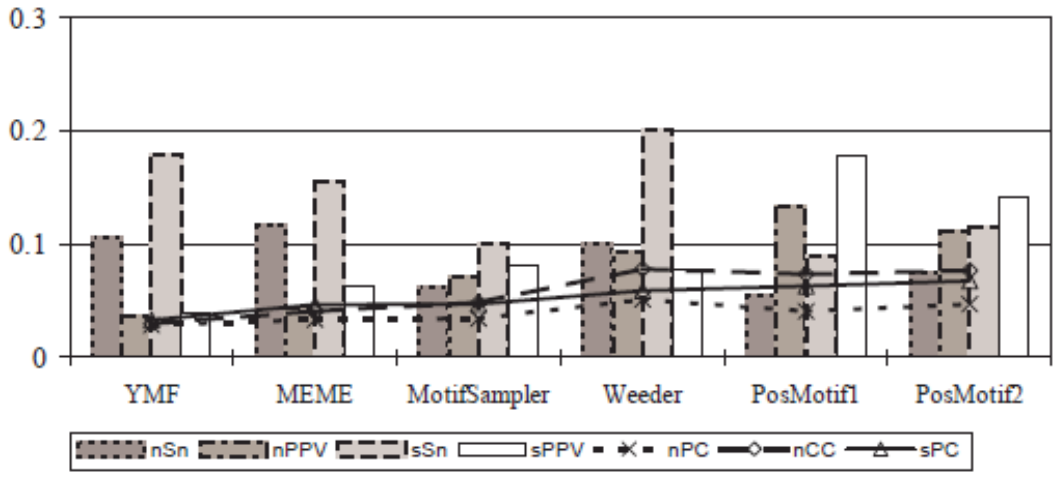


Figure 3.8: Performance of PosMotif and other motif finding algorithms on samples of type real from Tompa *et al.* (2005). For each algorithm, bars denote nSn, nPPV, sSn and sPPV from left to right, lines marked by crosses denote nPC, lines marked by diamonds denote nCC, and lines marked by triangles denote sPC, obtained by treating a set of samples as if it was a single large sample.

Table 3.1: nCC values of motif finding algorithms on samples of type real from Tompa *et al.* (2005) within each species, including fly, human, mouse and yeast. In each case, the highest value is in bold.

nCC	YMF	MEME	MotifSampler	Weeder	PosMotif1	PosMotif2
fly	0.023	**0.092**	−0.0034	0.038	0.019	0.012
human	0.037	0.010	0.046	**0.050**	0.040	0.039
mouse	0.019	**0.080**	0.029	0.074	0.048	0.042
yeast	0.021	0.085	0.11	0.23	0.28	**0.32**

Figure 3.9 shows performance comparisons of the algorithms on samples that contain at least three genes in the SCPD database (Zhu and Zhang, 1999), in which there are a total of 35 samples, with each sample containing up to 25 upstream sequences in yeast and each sequence of length 1000. The P-values from the Wilcoxon test show that PosMotif performs better than the other algorithms in most cases (except for YMF when the performance differences are insignificant), with PosMotif2 generally performing better than PosMotif1 (Table 3.2).

Table 3.2: P-value from the Wilcoxon matched-pairs signed-ranks test of PosMotif on samples that contain at least three genes in the SCPD database (Zhu and Zhang, 1999). Each algorithm on the left is compared against each algorithm on the top, with — indicating insignificant differences.

P-value		YMF	MEME	MotifSampler	Weeder	PosMotif1
PosMotif1	nPC	—	—	—	0.04	
	nCC	—	0.02	0.03	0.03	
	sPC	—	—	—	0.03	
PosMotif2	nPC	—	0.008	0.007	0.005	0.006
	nCC	—	0.005	0.005	0.01	0.02
	sPC	—	0.03	0.04	0.004	0.002

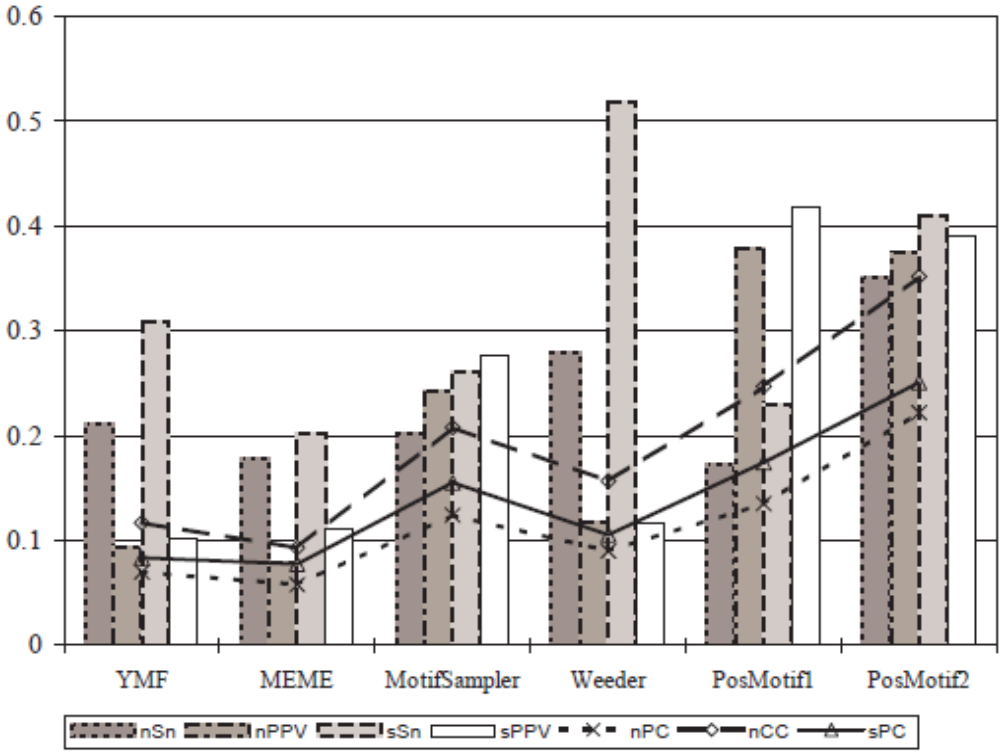


Figure 3.9: Performance of PosMotif and other motif finding algorithms on samples that contain at least three genes in the SCPD database (Zhu and Zhang, 1999). The notations are the same as in Figure 3.8.

Figure 3.10 shows performance comparisons of the algorithms on samples from the ABS database (Blanco et al., 2006), in which there are a total of 68 samples, with each sample containing up to 95 upstream sequences in multiple species from among human, mouse, rat and chicken, and each sequence of length 500. The P-values from the Wilcoxon test show that PosMotif performs significantly better than YMF in all cases, and it performs

significantly better than the other algorithms in most cases, with no significant performance differences between PosMotif2 and PosMotif1 (Table 3.3).

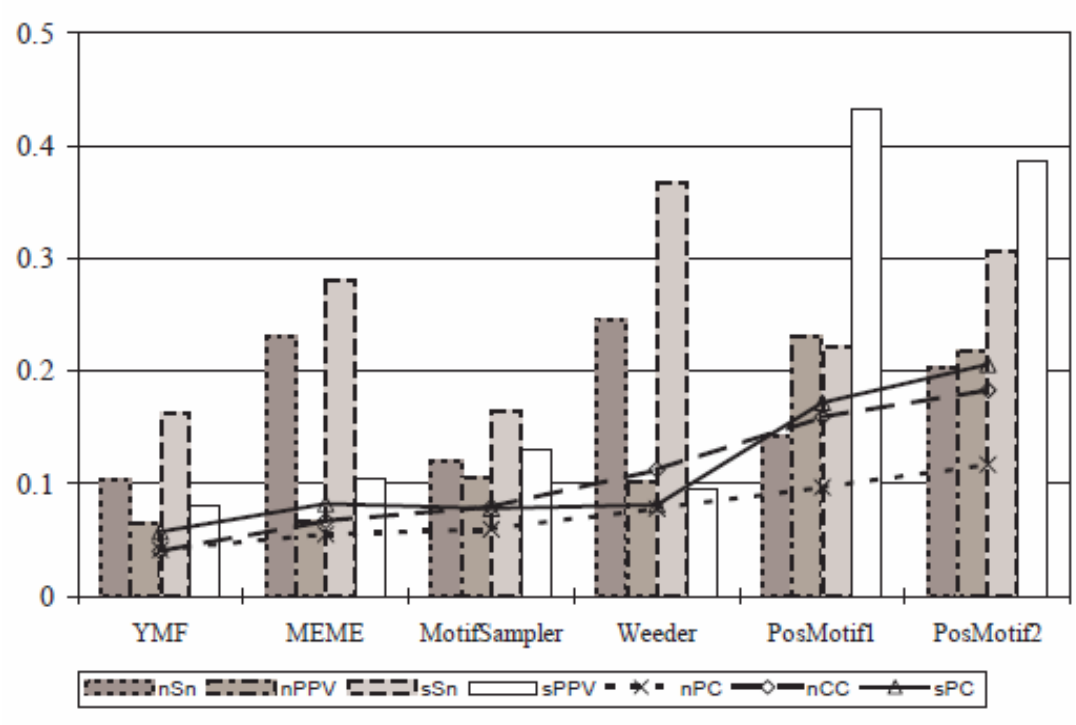


Figure 3.10: Performance of PosMotif and other motif finding algorithms on samples from the ABS database (Blanco *et al.*, 2006). The notations are the same as in Figure 3.8.

Table 3.3: P-value from the Wilcoxon matched-pairs signed-ranks test of PosMotif on samples from the ABS database (Blanco *et al.*, 2006). The notations are the same as in Table 3.2.

P-value		YMF	MEME	MotifSampler	Weeder	PosMotif1
PosMotif1	nPC	0.02	—	—	—	
	nCC	0.01	—	—	0.03	
	sPC	0.01	—	0.02	0.007	
PosMotif2	nPC	0.003	—	0.04	0.02	—
	nCC	0.008	0.03	—	0.02	—
	sPC	0.002	0.03	0.005	0.002	—

The above results show that PosMotif1 has high accuracy even before extensive post processing is performed when positions that are not ignored still contain fixed nucleotides. The second post processing step in PosMotif2 is useful, but does not always lead to significantly better accuracy. In general, the nucleotide level statistics *nPC* and *nCC* correlate well with each other, the site level statistics *sSn* and *sPPV* correlate well with the nucleotide level statistics *nSn* and *nPPV* respectively, and the site level statistic *sPC* correlates well with both the nucleotide level statistics *nPC* and *nCC*. To obtain good performance, appropriate tradeoffs have to be maintained between optimizing *nSn* and *nPPV* (or between *sSn* and *sPPV*), in which the former aims to reduce false negatives while the latter aims to reduce false positives.

To investigate the effectiveness of the post processing, we also tested the post processing steps on our previous motif-finding algorithm MotifEnumerator (Sze. S. and Zhao. X. 2006). Similarly to PosMotif, we consider two variants, including MotifEnumerator1, which combines algorithm MotifEnumerator with algorithm PostProcess that performs initial post processing, and MotifEnumerator2, which combines algorithm MotifEnumerator with algorithms PostProcess and PostProcess2 that perform both initial and further post processing.

We compare the performance of Motifenumerator to MEME (Bailey and Elkan, 1994), to Weeder (Pavesi et al.., 2001), and to AlignAce (Hughes et al., 1998). For each algorithm, we use the default parameters as much as possible. We run MEME with the *anr* option, with motif lengths of up to 20 on the default forward strand, and using a 5th order Markov background constructed from upstream sequences of entire species. We run Weeder with the large mode over motif lengths 6, 8, 10 and 12 on the default forward strand and using appropriate frequency tables of the given species.We run AlignAce with the minimum motif length as 6 and all other parameters as default.

In addition, we applied the dust routine on each benchmark datasets to further improve the performance.

Figure 3.11 shows performance comparisons of the algorithms on samples of type real from Tompa et al. (2005). When all samples from different species are considered

together, the *P*-values from the Wilcoxon test show that there are no significant performance differences between MotifEnumerator and the other algorithms except that both invariants of MotifEnumerator are significantly better than AlignAce on nCC values. The Wilcoxon test is not performed within each species since the number of samples is small.

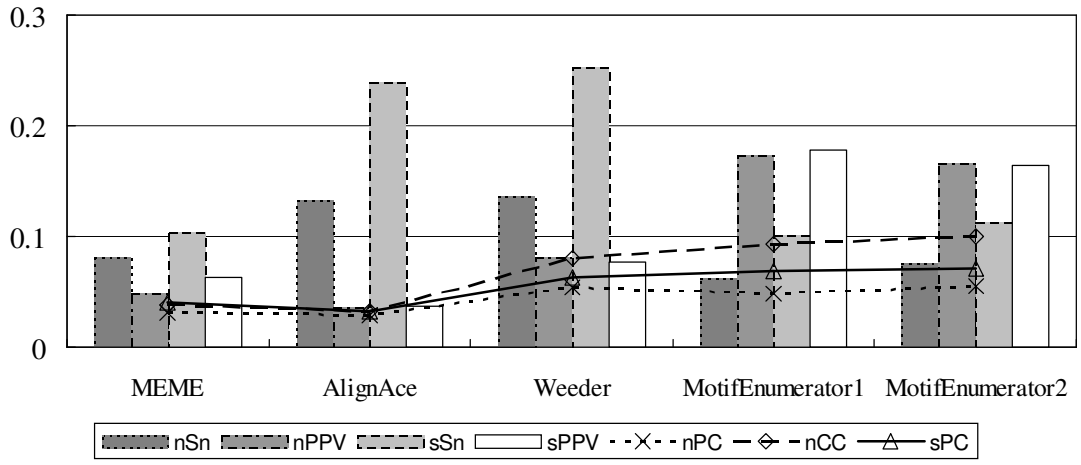


Figure 3.11: Performance of MotifEnumerator and other motif finding algorithms on samples of type real from Tompa *et al.* (2005). For each algorithm, bars denote nSn, nPPV, sSn and sPPV from left to right, lines marked by crosses denote nPC, lines marked by diamonds denote nCC, and lines marked by triangles denote sPC, obtained by treating a set of samples as if it was a single large sample.

Figure 3.12 shows performance comparisons of the algorithms on samples that contain at least three genes in the SCPD database (Zhu and Zhang, 1999). The P-values from the

Wilcoxon test show that MotifEnumerator performs better than the other algorithms in most cases, with MotifEnumerator2 generally performing better than MotifEnumerator1 (Table 3.4).

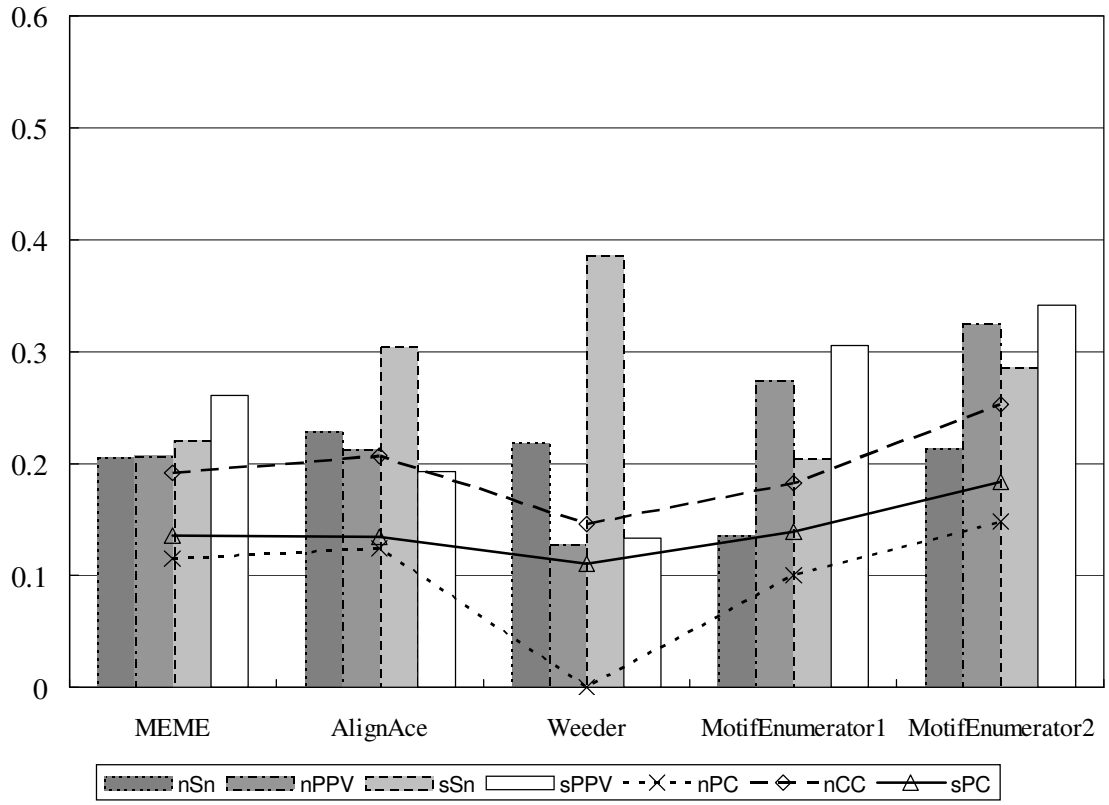


Figure 3.12: Performance of MotifEnumerator and other motif finding algorithms on samples that contain at least three genes in the SCPD database (Zhu and Zhang, 1999). The notations are the same as in Figure 3.8.

Table 3.4: P-value from the Wilcoxon matched-pairs signed-ranks test of MotifEnumerator on samples that contain at least three genes in the SCPD database (Zhu and Zhang, 1999). Each algorithm on the left is compared against each algorithm on the top, with — indicating insignificant differences.

P-value		MEME	AlignAce	Weeder	MotifEnumerator1
MotifEnumerator1	nPC	—	—	—	
	nCC	—	—	—	
	sPC	—	—	—	
MotifEnumerator2	nPC	—	—	0.05	0.0007
	nCC	0.04	—	0.02	0.001
	sPC	—	—	0.03	0.04

Figure 3.13 shows performance comparisons of the algorithms on samples from the ABS database (Blanco et al., 2006). The P-values from the Wilcoxon test show that MotifEnumerator performs significantly better than the other algorithms in most cases, with MotifEnumerator2 generally performing better than MotifEnumerator1 (Table 3.5).

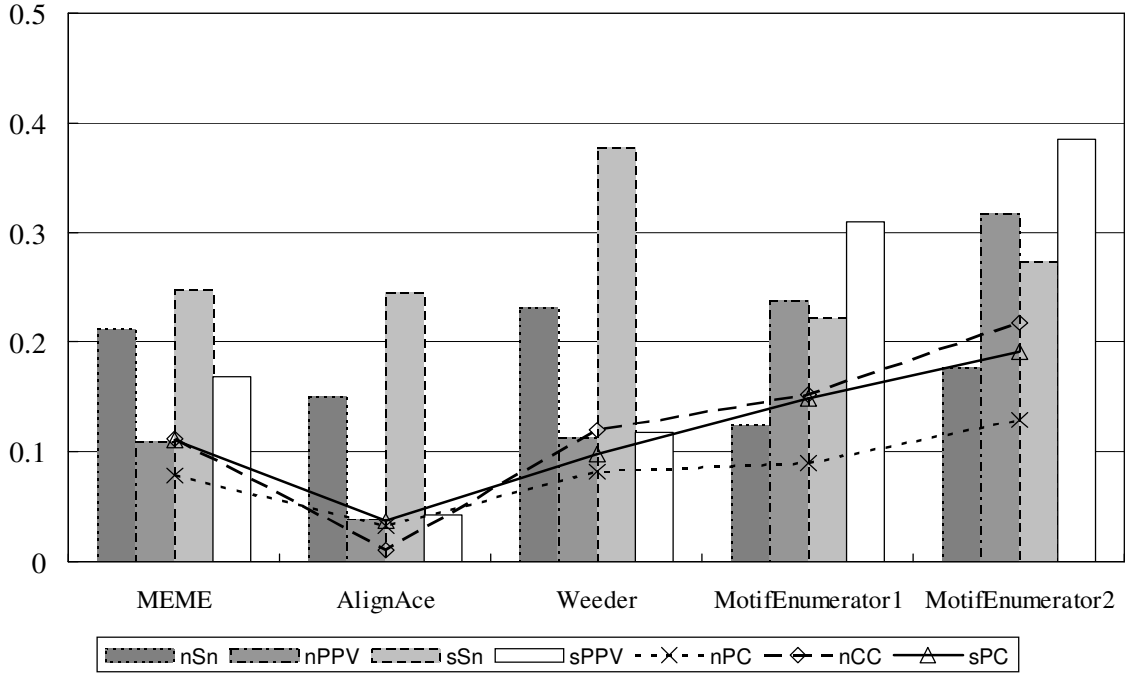


Figure 3.13: Performance of MotifEnumerator and other motif finding algorithms on samples from the ABS database (Blanco *et al.*, 2006). The notations are the same as in Figure 3.8.

The above results on MotifEnumerator showed that the post processing algorithms are effective most of the time and greatly improved the performance of old MotifEnumerator algorithm.

Table 3.5: P-value from the Wilcoxon matched-pairs signed-ranks test of MotifEnumerator on samples from the ABS database (Blanco *et al.*, 2006). The notations are the same as in Table 3.2.

P-value		MEME	AlignAce	Weeder	MotifEnumerator1
MotifEnumerator1	nPC	—	0.006	—	
	nCC	—	0.006	—	
	sPC	—	0.02	—	
MotifEnumerator2	nPC	—	0.01	0.06	0.0003
	nCC	0.03	6e-5	0.01	0.0006
	sPC	—	0.003	0.008	0.01

To further investigate the relationship between motif conservation and algorithm performance, we combine overlapping occurrences of each motif into one site and align these occurrences by using ClustalW (Thompson et al., 1994) with default parameters. We remove columns in the alignment in which less than 50% of the characters are gap characters and compute the consensus nucleotide in each remaining column while ignoring gap characters. Over a given set of samples, we define the motif conservation rate to be total number of nucleotides that are the same as the consensus nucleotide within a column divided by the total number of nucleotides in all the columns. Note that this procedure ignores the possibility that a motif can contain sites on both strands, but it should give a good approximation. Although some number of gap characters can appear in an alignment, they are rare and the above score reflects the conservation of core regions in a motif.

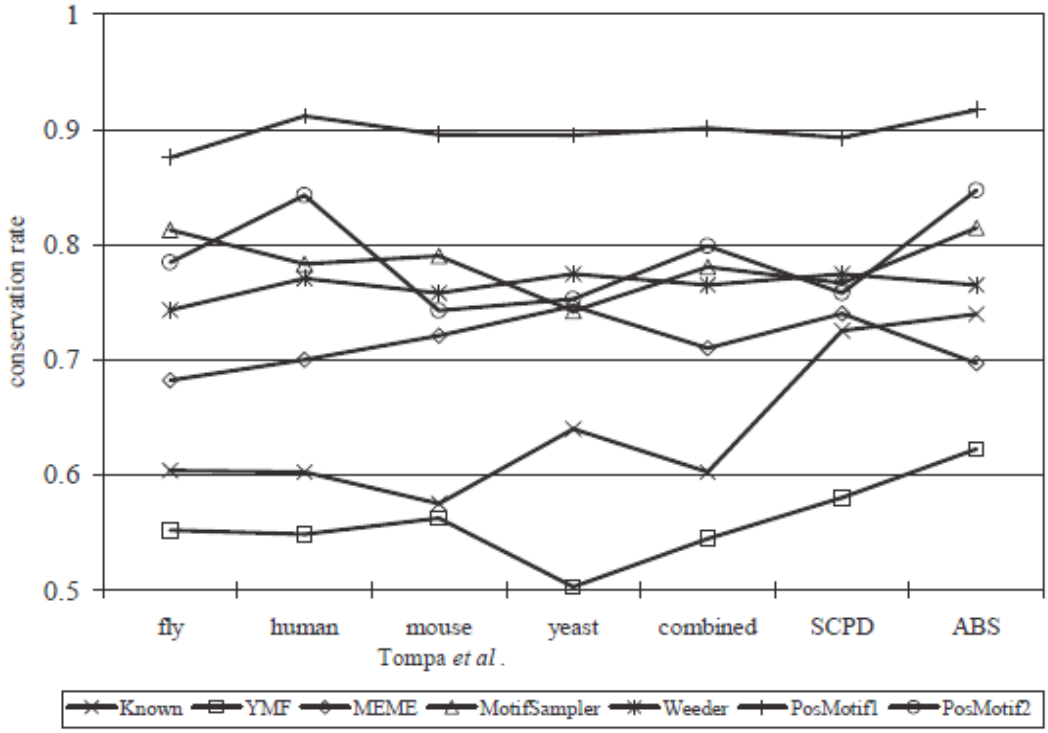


Figure 3.14: Conservation rate of known sites and top motifs from motif finding algorithms on each set of samples.

Figure 3.14 shows that although the known sites generally have low conservation rates, each algorithm has its own focus on finding motifs within a narrow range of conservation rates due to the specific motif model being used and the parameter settings. Among the three data sets, the high site conservation rate on the samples from the SCPD and ABS databases makes it easier for PosMotif to improve motif finding accuracy. Within the samples of type real from Tompa et al. (2005), the site conservation rate of the yeast samples is higher than that of the other species, which explains the better

performance of PosMotif on the yeast samples. Since our model focuses initially on identifying invariant core positions, it is most successful in identifying sites that have a high enough conservation rate within a sample, in which case the invariant positions are more prevalent and are better captured by our model. This is especially true on the samples from the SCPD and ABS databases, in which the site conservation rate is much more prevalent and are better captured by our model. This is especially true on the samples from the SCPD and ABS databases, in which the site conservation rate is much higher than that on the samples of type real from Tompa et al. (2005). Note that these performance differences are mostly due to the differences in the site conservation rate and are not species-specific.

E. Discussion

We have shown that by skipping non-conserved positions, many background Markov chains can be used simultaneously to better model long range nucleotide dependencies within motifs. Our initial focus on positions that have fixed nucleotides allows the development of an efficient algorithm that can find long motifs in moderately sized samples, due to a small base of two in the exponential part of the time complexity. The later post processing step gives rise to a general motif model in which each motif can contain variable length occurrences.

It takes minutes to hours to find motifs of length up to 18 or 20 for the smaller samples to one or two days for the larger samples. When the maximum motif length is lowered to smaller values such as 12, the algorithm becomes very fast and takes only seconds for many samples. Since the running time approximately doubles when l is increased by 1, it takes about twice as long to obtain all motifs of length at most l when compared to obtaining motifs for only one l.

To further improve accuracy, it is possible to develop more accurate formulas that have low time complexity for computing P-values and E-values, or consider more detailed models initially by allowing mismatches or degenerate letters within motifs. Another strategy is to use phylogenetic information on samples that contain sequences from multiple species.

CHAPTER IV

ALGORITHM BASED ON ADDING MORE DNA UPSTREAM SEQUENCES FROM OTHER SIMILAR PROTEINS

We proposed a new strategy to improve the performance of identifying the transcription factor binding sites in DNA sequences via similar genes. The idea is to add more upstream sequences to the input sample from the genes that are sufficiently similar to the input genes. We have tried this strategy in one large benchmark datasets and tested on five famous motif-finding tools. The results showed great improvements for each tool on the enriched benchmark datasets compared to the original benchmark datasets.

A. Introduction

Most existing motif-finding algorithms are tested on datasets that contain upstream sequences from several co-regulated genes, as co-regulated genes are known to share similar regulatory mechanism and their promoter region might contain common binding sites for transcription factors. However, as there are still lots of genes are unknown to be co-regulated or not, the collection of currently known co-regulated genes is just a subset of the whole co-regulated genes set. Sometimes this collection did not contain enough information to detect the real motif pattern. Therefore, most of the existing motif-finding tools perform much better in yeast and other lower organisms than in higher organisms, because through knowledge can be obtained on the lower organisms. Based on this

observation, we are trying to enrich the existing datasets by adding more information from other sufficiently similar genes, so that the real motif pattern can stand out. We can make a hypothesis that similar genes, even currently we don't know if they are co-regulated, may have similar patterns/features in their upstream sequences. So to avoid the limitations of the current available experiment data about the co-regulated genes, we proposed to add more upstream sequences from sufficiently similar genes.

B. Methods

1. Running BLAST

To find sufficiently similar genes, we can use BLAST to search on the sequences. If the given sample contains the corresponding gene information, we could obtain the corresponding upstream sequences from similar genes using the input gene information as the query (TBLASTN); otherwise we need two steps. The first step is trying to find the corresponding gene information by using the upstream sequences from the input sample as the query(BLASTN). It is possible that nothing may be found, and in this case we won't process any further to the current input sequences. If we can identify the gene information from the first step, then we can then use TBLASTN to search for similar genes. For each running of BLAST, we need to save the results for further processing as we only want to keep the ones that are interested to us.

2. Processing the Results from BLAST

The processing for the results from BLASTN is straight-forward, since we only need to find the exact match against the input upstream sequence, or return nothing if there is no such match. We define the exact match as follows:

i) the identity rate = 100 %

ii) the positive rate = 100%

iii) the aligned length=100% of input length

After we find the exact match, we then go to the corresponding gene bank file and extract the corresponding cds information, which will be inputs to the TBLASTN program.

The processing for the results from TBLASTN is similar to BLATN, except that we need to find suitable hits in this case. Obviously we don't want the exact match or the very similar matches, as they may come from the same gene and can not provide additional information to help identify the binding sites in the upstream sequences. On the other hand, we don't want the genes that are too different either, as they may have absolutely very different features, which can introduce noise for us to identify the binding sites. What we want is those genes that they are sufficiently similar so as to contain some common features as represented by their upstream sequences.

The criteria we find most suitable for processing TBLASTN results is as follows:

i) 50% <= the identity rate <= 80%

ii) 50% <= the positive rate <= 80%

Since gene bank files will be needed to process the results from BLAST running, and the gene banks files are usually too much. We provide two options of processing the results. If you already have a database containing the gene bank files you will be interested, you can run the processing algorithm locally and search gene bank files in the database you specified. If you don't have such a database or you don't know if your database is large enough to contain all the possible gene bank files, you can run the processing algorithm with an option to download the gene bank files from online NCBI GenBank to your local machine. The second option would first search for gene bank files in your local directory and then search in online NCBI genebank if it is not found locally.

A mini database containing all geneBank files for the testing data sets we used are available for downloading, as well as the scripts to run BLAST and to process BLAST results.

3. Modifying the Datasets by Adding More Sequences

After we processed the results from TBLASTN, for each input upstream sequence, there might be multiple candidate upstream sequences from similar genes. Especially these

candidate upstream sequences for the upstream sequences from the same data sets may contain duplicates with each other. To avoid this, we used the cd-hit to remove the duplicates or highly similar upstream sequences, in the preference that the original upstream sequences will be kept. After this, we add the selected candidate upstream sequences to the end of the dataset.

C. Performance

We have picked five most popular motif finding tools to run on both the original datasets and the modified datasets with added sequences. We picked MEME (Bailey and Elkan, 1994), which is one of the most popular motif finding algorithms that use the expectation maximization strategy, Weeder (Pavesi et al., 2001), which uses a combinatorial approach that has the best accuracy as assessed by Tompa et al. (2005), MotifSampler (Thijs et al., 2001), which uses a Gibbs sampling strategy that performs very well on samples of type real from Tompa et al. (2005), YMF (Sinha and Tompa, 2002), which uses a statistical approach that performs very well on samples of type mixed from Tompa et al. (2005), and AlignAce (Hughes et al., 1998), which is a famous tool using a Gibbs sampling algorithm with the weight matrix motif model.

For each of these tools, we use the default parameters as much as possible and use the same parameters for running on both the enriched datasets and the original datasets.

We run MEME with the *anr* option, with motif lengths of up to 20 on the default forward strand, and using a 5th order Markov background constructed from upstream sequences of entire species.

We run Weeder with the large mode over motif lengths 6, 8, 10 and 12 on the default forward strand and using appropriate frequency tables of the given species.

We run MotifSampler 20 times for each motif length 6, 8, 10 and 12 on the default forward strand and using a 3rd order Markov background constructed from upstream sequences of entire species.

We run YMF over motif lengths from 6 to 10, allowing for at most two degenerate symbols and at most 11 spacers for motifs of length 6 and no spacers for motifs of length larger than 6, while using a 3rd order Markov background constructed from upstream sequences of entire species.

We run AlignAce with the minimum motif length as 6 and all other parameters as default.

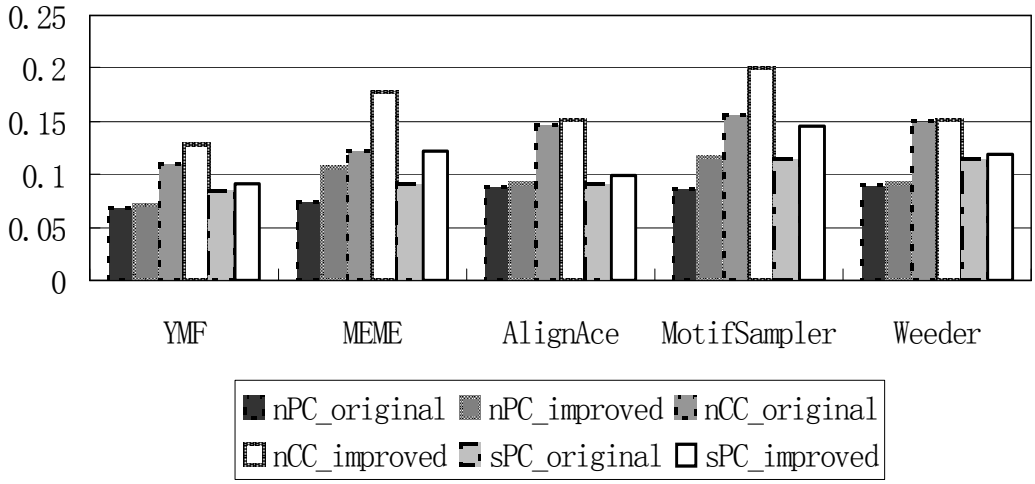


Figure 4.1: Performance of motif finding algorithms on samples that contain at least three genes in the SCPD database (Zhu and Zhang, 1999) and on the enriched version of samples. For each algorithm, bars denote nPC_original, nCC_original, and sPC_original are the performance on the original samples, while bars denote nPC_improved, nCC_improved, and sPC are the performance on the enriched version of the original samples.

Figure 4.1 shows performance comparisons of the algorithms that contain at least three genes in the SCPD database (Zhu and Zhang, 1999) and on the enriched version samples of samples. There are total 35 samples in this SCPD benchmark datasets, with each sample contains up to 25 upstream sequences in yeast and each of length 1000. The nPC, nCC and sPC values from the comparisons show that all these five algorithms have performance improvements on the enriched samples, with YMF, MEME and

MotifSampler show great improvements on all three values, AlignAce and Weeder show good improvement on the sPC values and small improvements on the nPC and nCC values.

D. Discussion

We have shown that by adding suitable upstream sequences into the original given sample, the performance of motif finding algorithms can be greatly improved. Our focus is to demonstrate that this idea is useful to improve the performance of most motif finding algorithms and provide helpful guide to the future benchmark datasets creation.

One possible future task is to run on some other existing difficult benchmark datasets, such as Tompa Benchmark and try to see the how the improvements can be. This will also give us hints how to evaluate different benchmark datasets, as if the performance on one benchmark datasets can be improved a lot by adding more relative sequences, this benchmark datasets may need improvements as well.

Another possible future task is try to analyze the helpful sequences been added to the original samples and find out the underlying possible causes. The sequences might come from different genes or species. This analysis would help us to better understand the transcription mechanism.

CHAPTER V

CONCLUSION AND FUTURE WORK

In this dissertation, we have developed three different methods to improve the performance of the motif finding problem.

In Chapter II, we have proposed an improved pattern-driven algorithm, MotifEnumerator, which has a reduced the time complexity from $O(4^l lkn)$ to $O(4^l lk)$ over the traditional exact pattern-driven approaches, where k is the number of sequences, n is the length of each sequence and l is the motif length. It saves a factor of n in time complexity when l is large enough. This is a significant improvement since n can be as large as 3000. It also extends the power of the pattern-driven approach to find all significant motifs of length around 12 or 13 (from the original limit of around 10), or substantially to around 20 while retaining most of the original sensitivity by allowing don't care positions but disallowing mismatches. The accuracy performance of this algorithm is comparable to the best existing motif finding algorithms on a large benchmark set of samples. To further improve MotifEnumerator, it may be desirable to allow a small amount of overlaps among suboptimal motif occurrences to avoid missing motifs. It is also useful if more accurate statistical formulas can be obtained so as to improve the accuracy of the motif scores.

In Chapter III, we have demonstrated another new algorithm with post processing, PosMotif, which uses a motif representation that allows arbitrary ignored positions within the non-conserved portion of single motifs, and uses Markov chains to model the background distributions of motifs of certain length while skipping these positions within each Markov chain. We have applied two post processing steps considering redundancy information in this algorithm and tested it on three large benchmark sets of samples. The performance comparisons with other five existing motif finding algorithms show significant improvement in motif prediction accuracy and the Wilconxon test show statistical improvements over the other tools. To further improve accuracy, it is possible to consider more detailed models initially by allowing mismatches or degenerate letters within motifs, or to use phylogenetic information on samples that contain sequences from multiple species.

In Chapter IV, we have illustrated a new method, Enrich, to improve the performance of motif finding algorithms by adding relative sequences to the input samples. By modifying the existing benchmark datasets, we show that this strategy is able to improve the performance of existing motif finding algorithms. The performance comparisons also indicate that this strategy would help to improve the quality of existing benchmark datasets as well. To further demonstrate this strategy, it may be useful to test on more motif finding algorithms and more benchmark datasets.

Some other possible future work might try to combine the last two methods we proposed to further improve the performance of motif finding algorithms. For example, we can use both post processing method and sample enriching strategy for any motif finding algorithm. It is also desired to formally evaluate the existing motif finding benchmark data sets and guide the direction of the future benchmark datasets creation.

REFERENCES

Apostolico, A., Parida, L. 2004. Incremental paradigms of motif discovery. *J. Comp. Biol.* 11, 15–25

Bailey, T.L., Elkan, C.P. 1994. Fitting a mixture model by expectation maximization to discover motifs in biopolymers. *Proc. 2nd Int. Conf. Intelligent Systems Mol. Biol.* 28–36

Barash, Y., Elidan, G. Friedman, N., Kaplan, T. 2003. Modeling dependencies in protein-DNA binding sites. *Proc. 7th Ann. Int. Conf. Res. Comp. Mol. Biol.* 28–37

Blanchette, M., Schwikowski, B., Tompa, M. 2002. Algorithms for phylogenetic footprinting. *J. Comp. Biol.* 9, 211–223

Blanco, E., Farr´e, D., Alb`a,M.M.,Messeguer,X. and Guig´o, R. 2006. ABS: a database of Annotated regulatory Binding Sites from orthologous promoters. *Nucleic Acids Res.* 34, D63–67.

Buhler, J., Tompa, M. 2002. Finding motifs using random projections. *J. Comp. Biol.* 9, 225–242

Chin, F. and Leung, H.C.M. 2008. DNA motif representation with nucleotide dependency. *IEEE/ACM Trans. Comput. Biol. Bioinformatics* 5, 110–119.

Crooks GE, Hon G, Chandonia JM, Brenner SE 2004. WebLogo: A sequence logo generator, *Genome Research* 14, 1188-1190

Eskin, E. 2004. From profiles to patterns and back again: a branch and bound algorithm for finding near optimal motif profiles. *Proc. 8th Ann. Int. Conf. Res. Comp. Mol. Biol.* 115–124

Eskin, E., Pevzner, P.A. 2002. Finding composite regulatory patterns in DNA sequences. *Bioinformatics* 18, S354–363

Favorov, A.V., Gelfand, M.S., Gerasimova, A.V., Ravcheev, D.A., Mironov, A.A., Makeev, V.J. 2005. A Gibbs sampler for identification of symmetrically structured, spaced DNA motifs with improved estimation of the signal length. *Bioinformatics* 21, 2240–2245

Fraenkel, Y.M., Mandel, Y., Friedberg, D., Margalit, H. 1995. Identification of common motifs in unaligned DNA sequences: application to Escherichia coli Lrp regulon.Comp. *Appl. Biosci*. 11, 379–387

Galas, D.J., Eggert, M., Waterman, M.S. 1985. Rigorous pattern-recognition methods for DNA sequences. Analysis of promoter sequences from Escherichia coli. *J. Mol. Biol.* 186, 117–128

GuhaThakurta, D., Stormo, G.D. 2001. Identifying target sites for cooperatively binding factors. *Bioinformatics* 17, 608–621

Hertz GZ, Hartzell GW, Stormo GD 1990. Identification of consensus patterns in unaligned DNA sequences known to be functionally related. *Comput Appl Biosci*. 6, 81-92.

Hertz GZ, Stormo GD 1999. Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. *Bioinformatics* 15, 563-577.

Hughes, J.D., Estep, P.W., Tavazoie, S., Church, G.M. 2000. Computational identification of cis-regulatory elements associated with groups of functionally related genes in Saccharomyces cerevisiae. *J. Mol. Biol.* 296, 1205–1214

Kaplan, T., Friedman, N. and Margalit, H. 2005. *Ab initio* prediction of transcription factor binding sites using structural knowledge. *PLoS Comput. Biol.* 1, E1.

Keich, U., Pevzner, P.A. 2002. Finding motifs in the twilight zone. *Bioinformatics* 18, 1374–1381

Kel, A., Tikunov, Y., Voss, N., Wingender, E. 2004. Recognition of multiple patterns in unaligned sets of sequences: comparison of kernel clustering method with other methods. *Bioinformatics* 20, 1512–1516

Lawrence, C.E., Altschul, S.F., Boguski, M.S., Liu, J.S., Neuwald, A.F., Wootton, J.C. 1993. Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment. *Science* 262, 208–214

Leung, H.C., Chin, F.Y. 2005.Finding exact optimal motifs in matrix representation by partitioning. *Bioinformatics* 21, SII86–92

Liu, X., Brutlag, D.L., Liu, J.S. 2001. BioProspector: discovering conserved DNA motifs in upstream regulatory regions of co-expressed genes. *Pac. Sym. Biocomp.* 127–138

Marsan, L., Sagot, M.-F. 2000. Algorithms for extracting structured motifs using a suffix tree with an application to promoter and regulatory site consensus identification. *J. Comp. Biol.* 7, 345–362

Modan, K. D., Ho-Kwok D. 2007. A survey of DNA motif finding algorithms, *BMC Bioinformatics* 8, S21

Pavesi, G., Mauri, G., Pesole, G. 2001: An algorithm for finding signals of unknown length in DNA sequences. *Bioinformatics* 17, S207–214

Peng, C.-H., Hsu, J.-T., Chung, Y.-S., Lin, Y.-J., Chow, W.-Y., Hsu, D.F. and Tang, C.Y. 2006. Identification of degenerate motifs using position restricted selection and hybrid ranking combination. *Nucleic Acids Res.* 34, 6379–6391.

Pesole, G., Prunella, N., Liuni, S., Attimonelli, M., Saccone, C. 1992. WORDUP: an efficient algorithm for discovering statistically significant patterns in DNA sequences. *Nucleic Acids Res.* 20, 2871–2875

Pevzner, P.A., Sze, S.-H. 2000. Combinatorial approaches to finding subtle signals in DNA sequences. *Proc. 8th Int. Conf. Intelligent Systems Mol. Biol.* 269–278

Price, A., Ramabhadran, S., Pevzner, P.A. 2003. Finding subtle motifs by branching from sample strings. *Bioinformatics* 19, SII149–155

Queen, C.,Wegman, M.N., Korn, L.J. 1982. Improvements to a program for DNA analysis: a procedure to find homologies among many sequences. *Nucleic Acids Res.* 10, 449–456

Rigoutsos, I., Floratos, A. 1998. Combinatorial pattern discovery in biological sequences: the TEIRESIAS algorithm. *Bioinformatics* 14, 55–67

Roth, F.R., Hughes, J. D., Estep, P. E., and Church G.M. 1998. Finding DNA regulatory motifs within unaligned non-Coding sequences clustered by whole-genome mRNA quantitation, *Nature Biotechnol.* 16, 939-945

Schneider TD, Stephens RM 1990. Sequence Logos: A new way to display consensus sequences. *Nucleic Acids Res.* 18, 6097-6100

Sinha, S. 2003. Discriminative motifs. *J. Comput. Biol.* 10, 599–615.

Sinha, S., Tompa, M. 2000. A statistical method for finding transcription factor binding sites. *Proc. 8th Int. Conf. Intelligent Systems Mol. Biol.* 344–354

Sinha, S. and Tompa, M. 2002. Discovery of novel transcription factor binding sites by statistical overrepresentation. *Nucleic Acids Res.* 30, 5549–5560.

Staden, R. 1989. Methods for discovering novel motifs in nucleic acid sequences. *Comp. Appl. Biosci.* 5, 293–298

Stormo, G.D., Hartzell, G.W. 1989. Identifying protein-binding sites from unaligned DNA fragments. *Proc. Natl. Acad. Sci. USA* 86, 1183–1187

Sze S.-H. and Zhao X. 2005. Improved pattern-driven algorithms for motif finding in DNA sequences. *Proceedings of the 2005 Joint RECOMB Satellite Workshops on Systems Biology and Regulatory Genomics.* 198-211.

Tavazoie, S., Hughes, J.D., Campbell, M.J., Cho, R.J., Church, G.M. 1999. Systematic determination of genetic network architecture. *Nature Genet.* 22, 281–285

Thijs, G., Lescot, M., Marchal, K., Rombauts, S., De Moor, B., Rouz´e, P., Moreau, Y. 2001. A higher-order background model improves the detection of promoter regulatory elements by Gibbs sampling. *Bioinformatics* 17, 1113–1122

Thompson, J.D., Higgins, D.G. and Gibson, T.J. 1994. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position specific gap penalties and weight matrix choice. *Nucleic Acids Res.* 22, 4673–4680.

Tompa, M. 1999. An exact method for finding short motifs in sequences, with application to the ribosome binding site problem. *Proc. 7th Int. Conf. Intelligent Systems Mol. Biol.* 262–271

Tompa, M., Li, N., Bailey, T.L., Church, G.M., De Moor, B., Eskin, E., Favorov, A.V., Frith, M.C., Fu, Y., Kent, W.J., Makeev, V.J., Mironov, A.A., Noble, W.S., Pavesi, G., Pesole, G., R´egnier, M., Simonis, N., Sinha, S., Thijs, G., van Helden, J., Vandenbogaert, M., Weng, Z., Workman, C., Ye, C., Zhu, Z. 2005. Assessing computational tools for the discovery of transcription factor binding sites. *Nature Biotech.* 23, 137–144

Van Helden, J., Andr´e, B., Collado-Vides, J. 1998. Extracting regulatory sites from the upstream region of yeast genes by computational analysis of oligonucleotide frequencies. *J. Mol. Biol.* 281, 827–842

Van Helden, J., Rios, A.F., Collado-Vides, J. 2000. Discovering regulatory elements in noncoding sequences by analysis of spaced dyads. *Nucleic Acids Res.* 28, 1808–1818

Waterman, M.S., Arratia, R., Galas, D.J. 1984. Pattern recognition in several sequences: consensus and alignment. *Bull. Math. Biol.* 46, 515–527

Wijaya, E., Rajaraman, K., Yiu, S.-M. and Sung, W.-K. 2007. Detection of generic spaced motifs using submotif pattern mining. *Bioinformatics* 23, 1476–1485.

Wilcoxon, F. 1947. Probability tables for individual comparisons by ranking methods. *Biometrics* 3, 119–122.

Wingender, E., Dietze, P., Karas, H. and Kn¨uppel, R. 1996. TRANSFAC: a database on transcription factors and their DNA binding sites. *Nucleic Acids Res.* 24, 238–241.

Wolfertstetter, F., Frech, K., Herrmann, G., Werner, T. 1996. Identification of functional elements in unaligned nucleic acid sequences by a novel tuple search algorithm. *Comp. Appl. Biosci.* 12, 71–80

Workman, C.T., Stormo, G.D. 2000. ANN-Spec: a method for discovering transcription factor binding sites with improved specificity. *Pac. Sym. Biocomp.* 467–478

Zhou, Q., Liu, J.S. 2004. Modeling within-motif dependence for transcription factor binding site predictions. *Bioinformatics* 20, 909–916

Zhu, J. and Zhang,M.Q. 1999. SCPD: a promoter database of the yeast *Saccharomyces cerevisiae*. *Bioinformatics* 15, 607–611.

VITA

Name: Xiaoyan Zhao

Address: Department of Computer Science & Engineering

Texas A&M University

2128 TAMU

College Station, TX 77843

Email Address: realxfan@gmail.com

Education: B.S., Computer Science, Beijing Normal University, 2002

Ph.D., Computer Science, Texas A&M University, 2010