Effective graph-based algorithms for weak motif discovery in genomic sequences

Sun, Hequan

2014

Sun, H. (2014). Effective graph-based algorithms for weak motif discovery in genomic sequences. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/64820>

<https://doi.org/10.32657/10356/64820>

121
4x
D/Blue

EFFECTIVE GRAPH-BASED
ALGORITHMS FOR WEAK MOTIF
DISCOVERY IN GENOMIC SEQUENCES

SUN HEQUAN

School Of Computer Engineering

A thesis submitted to the Nanyang Technological
University in partial fulfilment of the requirements for
the degree of Doctor of Philosophy

2014

Acknowledgments

First of all, I express my sincere gratitude to three of my supervisors, including Associate Professor Hsu Wen Jing, Professor Jagath C. Rajapakse and Dr Malcolm Low Yoke Hean. Especially, I would like to thank my current main supervisor, Associate Professor Hsu Wen Jing, who kindly took over the main supervision work of my thesis after Professor Jagath C. Rajapakse and Dr. Malcolm Low Yoke Hean. Dr. Low supervised my work from August 2008 to July 2011, but for some reasons he left NTU. Prof. Jagath, as my main supervisor from July 2011 to November 2012, also could not manage to finalize my work due to a healthy reason. All of them spent numerous efforts in leading me to this research group. They have input a lot of efforts to establish the research direction for me, and finally we make our work published in two leading journals in this field. I also appreciate that all my supervisors have spent so much time on revising drafts of all my papers carefully and enthusiastically. I appreciate that they have taught me so many effective research skills from numerous discussions. I have also received kind care and help from them on my daily life, which makes me feel comfortable and thus concentrate on the research. Words of Prof. Hsu are always encouraging which would accompany me for ever. To sum up, without them, I would not find the way to finish this thesis.

Meanwhile, I express special thanks to Dr. Yang Xiao and Dr. Sy Loi Ho, former students of Prof. Jagath, because I would not quickly find the breakthrough points of my research without learning from their guiding introductions and discussions. I thank Dr. Tan Ching Wai for his help on starting my research work. I have obtained many useful suggestions on research from him. Besides, I thank Dr. Wong Li Pei, Mr. Dushan Nawoda Wadduwage, Ms. Zhao Wen Bo and Mr. Liu Fan for many discussions on the implementation of proposed algorithms. I thank all the staffs of PDCC and BIRC, especially Ms. Ng-Goh Siew Lai, Ms. Tay Siew Eng, Mr. Poliran Kenneth Caballes and Mr. Tan Sing Yau, for their sincere (technical) help.

I thank Professor Peng Qin Ke from Xi'an Jiaotong University (my supervisor when I was a master student). Any discussions with him make my attitude to difficult positions become more positive. I thank all of my lab-mates, including Dr. Li Zengxiang, Mr. Hu Nan, etc. Their kindness always make me feel lighthearted and full of energy. I thank many friends known on the basketball court, including Prof. Wu Yuan from Nanyang Business School, Prof. Pui Mun Lee from Singapore Institute of Management, Dr. Cai Zhen Han (CEE), Dr. Qian Kai (CEE), and Dr. Du Lu Ping (EEE), Fang Wei etc, I am deeply influenced by their combatant spirit and humor on the court. I also thank Dr. Zhou Yong and Dr. Gao Ting Ting for providing me a comfortable place to enjoy good rest and sleep for keeping healthy.

I thank NTU for awarding me the research scholarship to support my study.

I thank Dr. Sandve, G. K., Dr. Abul, O., Dr. Walseng, V. and Dr. Drabløs, F for providing the improved benchmark and the website tool for evaluating our algorithms. I also thank Dr. Hu, J., Dr. Li, B. and Dr. Kihara, D. for providing the ECRDB70B-X datasets.

I thank three examiners for providing many useful comments for improving the thesis.

Last but not least, I would like to thank my family. My parents, brothers and fiancée support me unconditionally on my study as well as daily life. It is precious and honored for me to have them on the road of life. I wish them health and happiness forever.

Contents

Acknowledgments	i
Contents	iii
Abstract	vi
List of Acronyms	viii
List of Frequently Used Symbols	ix
List of Figures	x
List of Tables	xii
1 Introduction	1
1.1 Background	2
1.1.1 Gene expression and regulation	2
1.1.2 Motif and weak motif	3
1.2 Motif discovery and weak motif discovery	5
1.3 Motivation	7
1.4 Main contributions	8
1.5 Thesis synopsis	9
2 Literature Review	11
2.1 Introduction	11
2.1.1 Profile and string models	13
2.2 Profile-based algorithms	15
2.2.1 Expectation maximization algorithms	15
2.2.2 Gibbs sampling algorithms	21
2.2.3 Other algorithms	26
2.3 String-based algorithms	27

2.3.1	Pattern-driven algorithms	28
2.3.2	Sample-driven algorithms	36
2.4	Gapped motif discovery	39
2.5	Summary	41
3	Tree-structured Algorithms for WMD in Exact Datasets	44
3.1	Introduction	44
3.1.1	Inefficiency of DPCFG algorithm	45
3.2	TreeMotif-BF: tree construction in a breadth-first manner	47
3.2.1	Graph representation	48
3.2.2	Tree construction	49
3.2.3	Motif refinement	52
3.2.4	TreeMotif-BF: an example	53
3.2.5	Time and space complexity	56
3.3	TreeMotif-DF: tree construction in a depth-first manner	58
3.3.1	List construction	58
3.3.2	TreeMotif-DF: an example	59
3.3.3	Time and space complexity	62
3.4	Experimental results	65
3.4.1	Synthetic data	65
3.4.2	Biological data	88
3.5	Summary	94
4	Graph-based Recursive Algorithm for WMD in Exact Datasets	96
4.1	Introduction	96
4.2	RecMotif algorithm	98
4.2.1	Reference selection	100
4.2.2	Graph construction with one reference sequence	100
4.2.3	Recursive graph construction	100
4.2.4	RecMotif: an example	102
4.2.5	Time and space complexity	106
4.3	Experimental results	108

4.3.1	Synthetic data	108
4.3.2	Biological data	112
4.4	Summary	114
5	Improved Graph-based Algorithms for WMD in Noisy Datasets	116
5.1	Introduction	116
5.2	Strategies for handling noisy datasets	119
5.2.1	Selection of reference sequences	119
5.2.2	Determination of noisy sequences	121
5.3	nTreeMotif algorithm	124
5.3.1	Graph representation	124
5.3.2	Recursive tree construction	126
5.4	nRecMotif algorithm	130
5.4.1	Recursive graph construction	130
5.4.2	Initialization and motif refinement	134
5.4.3	nRecMotif: an example	134
5.5	Time and space complexity of nTreeMotif and nRecMotif	140
5.6	Experimental results	141
5.6.1	Synthetic data	141
5.6.2	Biological data	149
5.7	Summary	152
6	Summary and Future Work	153
6.1	Summary and discussion	153
6.2	Future work	155
6.2.1	Significance of motifs	155
6.2.2	Algorithmic efficiency	156
6.2.3	Gapped motif discovery	157
6.3	Conclusion	157
	Appendices	159
	Author's Publications	160
	References	161

Abstract

This thesis aims to improve *weak motif discovery* in genomic sequences. The task is of primary significance and urgency because motifs provide the basis for biologists to derive knowledge about gene functions. The knowledge could reveal mechanisms of diseases and lead to novel molecular targets for inventing therapeutic drugs. Nevertheless, due to prohibitive cost, traditional wet-lab techniques are no longer adequate for large scale data. In this regard, computational approaches can render valuable help.

Computational discovery of weak motifs, however, remains challenging. Because many false instances of a degenerate motif can easily disguise the true ones, in spite of intensive research, performance of the existing algorithms for this problem is far from being satisfactory. Approximate algorithms based on Expectation Maximization or Gibbs Sampling can miss true instances; exact ones based on clique finding in graphs or generating-and-validating patterns (candidate motifs) consume a large amount of time/space. Thus, there is much room for improving the algorithms.

We propose three novel algorithms for discovering (weak) motifs from *exact* datasets, where each sequence contains at least one motif instance. TreeMotif-BF is a tree-structured algorithm, whose novelty lies in the construction of trees of motif instances in a breadth-first manner. Experiments demonstrate that TreeMotif-BF is more scalable than the other existing algorithms, in terms of the length of motifs. However, TreeMotif-BF and many algorithms have difficulty in discovering very weak motifs due to enormous space requirement. Thus, the algorithm TreeMotif-DF constructs trees in a depth-first manner, overcoming the space limitation. Another algorithm RecMotif finds cliques of motif instances in recursively constructed graphs also in a depth-first manner. RecMotif reduces space requirement significantly. Besides, it further improves efficiency in execution time for solving open challenge problems.

We also propose two recursive algorithms for discovering motifs from *noisy* datasets, where some of the input sequences may contain no motif instances. The two generalized algorithms nTreeMotif and nRecMotif are improved from TreeMotif-BF and RecMotif respectively. The algorithms are based on efficient exclusion of noisy sequences and the improved construction of trees/graphs. nTreeMotif and nRecMotif preserve accuracy and efficiency of TreeMotif-BF and RecMotif respectively for dealing with exact datasets. Moreover, they are more scalable in terms of the number of noisy sequences than the existing algorithms.

The novel graph-based algorithms have successfully met the research objective. They can effectively discover weak motifs from datasets for which the existing algorithms have difficulty handling. Thus, they should be useful new additions to the repertoire of tools for bioinformatics.

List of Acronyms

Acronym	Description
DNA	Deoxyribonucleic Acid
DPCFG	Dynamic Programming for Clique Finding in Graphs
EM	Expectation Maximization
FDR	False Discriminative Rate
FIFO	First In First Out
GA	Genetic Algorithm
GMD	Gapped Motif Discovery
GS	Gibbs Sampling
HMM	Hidden Markov Model
IUPAC	International Union of Pure and Applied Chemistry
MCL-WMR	Markov CLuster algorithm for Weak Motif Recognition
MCP	Motif Challenge Problem
MD	Motif Discovery
MEME	Multiple EM for Motif Elicitation
MITRA	Mismatch TRee Algorithm
MSS	Median String Search
OOPS	One Occurrence of a motif Per Sequence
PF	Phylogenetic Footprinting
PMS	Planted Motif Search
PSSM	Position Specific Scoring Matrix
PSFM	Position Specific Frequency Matrix
PSWM	Position Specific Weight Matrix
PWM	Position Weight Matrix
RNA	Ribonucleic Acid
TF	Transcription Factor
TFBS	Transcriptional Factor Binding Site
TSS	Transcription Starting Site
WMD	Weak (l, d)-Motif Discovery
gWMD	Gapped Weak Motif Discovery
ZOOPS	Zero or One Occurrence of a motif Per Sequence

List of Frequently Used Symbols

Symbol	Description
d	the maximum number of mismatches between a motif and its instances
k	the number of sequences containing a motif instance in a noisy dataset
l	the length of a motif
(l, d)	the model of a motif in an exact dataset
$(l, d)-k$	the model of a motif in a noisy dataset
m	the number of sequences in a dataset
n	the length of sequences in a dataset
$nASP$	the nucleotide level average of nSn and $nPPV$
nCC	the nucleotide level correlation coefficient
nPC	the nucleotide level performance coefficient
$nPPV$	the nucleotide level positive predictive value
nSn	the nucleotide level sensitivity
nSP	the nucleotide level specificity
p	the probability of two l -mers within $2d$, or weakness of an (l, d) -motif
s	the dataset of m sequences
s_i	the i^{th} sequence of a dataset s
$s_{i,j}$	the base of sequence s_i that is at position j
v_i	the set of all the l -mer substrings of s_i
$v_{i,j}$	the l -mer substring of sequence s_i that starts at position j
$D()$	the function for computing Hamming distance of two strings
$D_{max}()$	the function for computing maximum Hamming distance of many strings
$L_{v_{i,j}}$	the list at vertex $v_{i,j}$ (TreeMotif-DF)
$N(l, d)$	the d -neighborhood of an l -mer string
N_p	the number of sequences to prepare Hamming distance tables
P_i	the set of strings selected from v_i according to a reference vertex
$S_{i,t}$	the status of sequence s_t in *ref-depth i (nTreeMotif or nRecMotif)
V	the number of valid sequences for a reference node
Ω	the alphabet of a sequence

Note* ref-depth: the depth of a node of the tree of reference

List of Figures

2.1	Models of a motif: (a) four motif instances with $l=6$; (b) base counts on each site and the consensus; (c) PSFM. (Note* 'x': no dominant base.)	13
3.1	Example: nodes for list construction by DPCFG	46
3.2	TreeMotif-BF: illustration of tree construction and clique refinement	54
3.3	TreeMotif-DF: illustration of tree construction	60
3.4	Comparison: effect of increasing n on nPC	70
3.5	Comparison: effect of increasing n on nSn	72
3.6	Comparison: effect of increasing n on execution time	74
3.7	Comparison: effect of increasing p on execution time	82
3.8	QQ-plot of nCC of TreeMotif-BF and Weeder	92
3.9	Comparison: performance of algorithms on a real benchmark dataset	93
4.1	RecMotif: an example	102
4.2	RecMotif: graph construction for vertex A	104
4.3	Comparison: effect of increasing n on execution time. (RecMotif included)	110
4.4	Comparison: effect of increasing p (RecMotif included) on execution time	113
5.1	Tree of reference sequences	120
5.2	nRecMotif: graph for vertex A	136
5.3	nRecMotif: graph for vertex U	137
5.4	nRecMotif: graph for vertex H	138
5.5	nRecMotif: graph for vertex Z	139
5.6	nRecMotif: another graph for vertex U	140
5.7	Comparison: effect of increasing n on nPC on noisy datasets	143

5.8	Comparison: effect of increasing n on execution time on noisy datasets	144
5.9	Comparison: effect of increasing $m-k$ on nPC on noisy datasets	146
5.10	Comparison: effect of increasing $m-k$ on execution time on noisy datasets	147

List of Tables

1.1	Example: motif instances of a dataset [64]	4
2.1	IUPAC alphabet of DNA/RNA	16
2.2	Example: dyad motifs in <i>P. horikoshii</i>	40
2.3	Typical motif discovery algorithms	43
3.1	Comparison: time and space complexities	64
3.2	Comparison: memory consumption	65
3.3	Comparison: motif challenge problem	68
3.4	Comparison: weakness indicated by value E	69
3.5	Comparison: effects of increasing m on execution time	75
3.6	Comparison: effects of varying (l, d) with fixed p on execution time	79
3.7	Comparison: (l, d) -motifs for increasing p	80
3.8	Comparison: cases that TreeMotif algorithms cannot handle (for $n=600$ and $m=20$)	83
3.9	Percentage of bases in different samples of DNA	84
3.10	Comparison: effect of varying $(G+C)$ -content on discovery accuracy	85
3.11	Example: discovered cliques	86
3.12	Base counts of cliques (given in Table 3.11)	87
3.13	Comparison: performance of TreeMotif algorithms on real datasets	89
3.14	Comparison: performance of algorithms on another real benchmark dataset	91
4.1	Comparison: time and space complexities (RecMotif included)	107
4.2	Comparison: motif challenge problem (RecMotif included)	109
4.3	Comparison: effects of increasing m on execution time (RecMotif included)	111

4.4	Comparison: effects of varying (l, d) with fixed p on execution time (Rec-Motif included)	112
4.5	Comparison: effect of varying $(G+C)$ -content on execution time	114
5.1	Time and space complexities of algorithms for handling noisy datasets . .	141
5.2	Comparison: effect of increasing (l, d) on execution time on noisy datasets	148
5.3	Comparison: performance of algorithms on ECRDB70B-X datasets . . .	151
6.1	Future work	158

Chapter 1

Introduction

Genome sequencing technology has continuously improved over the past two decades [116, 117]. By virtue of the technology, genome projects have led to a large number of sequences [25, 130, 176]. The genomic sequences carry nature-encoded information about fundamental biological processes, which is useful for researchers to reveal disease mechanisms and invent novel therapeutic drugs [15, 149, 189]. However, due to the prohibitive capital cost and time involved in manual annotation, the rich information in data remains largely hidden [140]. In this regard, computational techniques have become especially helpful in biological and biomedical research [140].

These recent developments have expedited the formation of research area of bioinformatics, of which the research objectives are three-fold [109]. The first objective is to establish large databases that can be easily updated with new discoveries and conveniently available when required. As raw data provide no straightforward explanations by themselves, researchers have to carry out further analysis to extract hidden information therein. Thus, the second objective is to develop automated tools that are capable of analyzing the data accurately and efficiently. For this purpose, researchers may employ techniques from other disciplines, such as mathematics, statistics and computer science [140]. With tools available, the third objective is to apply them on the raw data to convert hidden information into descriptive knowledge in life sciences.

In short, bioinformatics assists biologists in data management and knowledge discovery. The field of research, however, is facing many challenging problems. For example, one fundamental problem in bioinformatics is the computational discovery of weak motifs in regulatory sequences [131, 174]. This problem has drawn considerable interests as motifs play an essential role in deciphering biological processes such as gene expression [18, 37, 47]. However, current achievements in this problem are far from being satisfactory. Existing approximate techniques often miss target motifs, while exact techniques consume a large amount of time or space to derive the motifs. The scalability of exact techniques, such as on the size of datasets, requires further improvements [131].

To overcome the current limitation in weak motif discovery, this thesis develops improved effective and efficient algorithms. To understand the biological significance of motifs, we first give a brief introduction on gene expression and regulation. Built upon that basis, we present motivations to this research in the context of the state of the art in motif discovery.

1.1 Background

1.1.1 Gene expression and regulation

Motifs are indispensable elements for facilitating gene expression, via which the central dogma of molecular biology describes the flow of genetic information from DNA to RNA and from RNA to protein [26, 52, 113, 169]. All known life forms, including eukaryotes such as *multicellular organisms* or prokaryotes such as *bacteria* and *archaea*, employ gene expression to develop their molecular machineries [179]. On one hand, an organism produces RNAs and proteins to regenerate itself during its development. On the other hand, genes in the same organism can express with a different isoform, allowing the organism to synthesize different proteins and adapt to the changes in the environment.

The expression diversity gives rise to cellular differentiation, morphogenesis, and versatility and adaptability of an organism, which biologists attribute to (gene) regulation by regulatory factors [108]. Specifically, regulatory factors can control the subprocesses of gene expression, namely, the transcription, translation, and post-translational modification of a protein, on their timing, location and quantity [124].

Regulatory factors have to interact with the regulatory sequence of a gene to perform regulation. For example, certain *transcriptional factors* (TFs) bind to a promoter located in the upstream of a gene, and they can stimulate the transcription of the gene [92, 115]. TFs are diverse and have distinctive functions, for instance, activators, repressors, and terminators. Correspondingly, their binding sites in the promoter are different in composition and length. These *Transcriptional Factor Binding Sites* (TFBSs) provide the basis for interaction between TFs and regulatory sequences [37, 46, 47, 89, 178].

In summary, as a type of motifs, TFBSs mediate the collaboration of regulatory participants, and hence indirectly influence the transcription of a gene [179]. Therefore, the study of TFBS motifs is of primary significance for biologists in the study of the regulatory mechanism of gene expression.

1.1.2 Motif and weak motif

TFBSs are a type of conserved regions shared by regulatory sequences. In general, conserved regions are of distinctive biological significance as revealed by evolutionary analysis. For instance, they could be used to infer phylogenetic relationships of different species [10, 89, 93]. The conserved regions are known as motifs relevant to a biological process in genetics [119]. We discuss a motif (of TFBSs) as *monad* or *composite*.

A monad motif represents a number of instances appearing in a set of sequences with statistical significance, which are contiguous segments of the sequences (containing several to tens of bases) [89]. Table 1.1 shows a set of motif instances. Another typical

Table 1.1: Example: motif instances of a dataset [64]

Name of sequence	start position of motif	motif instance and its surroundings		length of sequence
<i>cloacin-df13</i>	97	...TCA	tactgtgtatatatacagta	TTT... 200
<i>colicin-e1</i>	97	...TTA	tgctgtatataaaaccagtg	GTT... 200
	112	...AAC	cagtgtgtatatgtacagta	TTT...
<i>colicin-ia</i>	99	...TAA	tactgtatatgtatccatat	GCG... 200
<i>reca</i>	71	...TGA	tactgtatgagcatacagta	TAA... 200
<i>recn</i>	71	...CTT	tactgtatataaaaccagtt	TAT... 200
	93	...TTA	tactgtacacaataacagta	ATG...
	111	...CAG	taatgggttttcatacagga	AAC...
<i>sula</i>	85	...ATG	tactgtacatccatacagta	ACT... 200
<i>umu-operon</i>	91	...GAC	tactgtatataaaaacagta	TAA... 200
<i>uvra</i>	60	...CAA	tactgtatattcattcaggt	CAA... 200
<i>uvrb</i>	71	...ATG	aactgtttttttatccagta	TAA... 200
<i>uvrd</i>	102	...CAA	atctgtatatataccagct	TTT... 200
<i>colicin-ib</i>	99	...TAG	tactgtatatgtatccatat	ACG... 200
<i>colicin-a</i>	34	...TTT	tactgtatataaacacatgt	GAA... 136
	48	...AAC	acatgtgaatatatacagtt	TTT...
<i>lexa</i>	55	...TTT	tgctgtatatactcacagca	TAA... 173
	76	...CAT	aactgtatatacacccaggg	GGC...
<i>muc-operon</i>	49	...CGA	tactgtataaataaacagtt	ATT... 158
<i>hima/uvrc</i>	none		-	200

monad motif is TATA box with a core consensus of 5'-TATAAA-3' in the promoter region of genes in *archaea* and *eukaryotes*.

A composite motif represents a group of adjacent TFBSs for several TFs, which work cooperatively as a team to realize a regulation [3, 7, 84, 121, 177, 181, 190, 200]. Therefore, it can be modeled as a set of submotifs that appear with gaps/spacers between neighboring ones [48]. A *dyad* motif is the simplest composite motif, which combines two monad motifs that appear in each sequence at roughly fixed distance [61, 88].

Intuitively, a motif is a subsequence of a (regulatory) sequence. Let Ω denote the DNA alphabet $\{A, C, G, T, X\}$ where 'X' indicates an arbitrary base [162]. Mathematically, a motif is an l -mer string $\{x_i\}_{i=1}^l$ where $x_i \in \Omega$. A feature of motifs is that they may mutate or rearrange the orders of some sites so as to influence molecular functions for evolutions [127, 191]. For example, instances in Table 1.1 have small numbers of mismatches with each other; *CRP* dataset contains highly degenerate motif instances with a consensus

TGTGAXXXGXTCA [162]. A motif such as the preceding example is said to be subtle as pairwise (Hamming) distances between instances are large.

In general, mutations, insertions, or deletions can result in large differences among instances of a common ancestor motif. The ancestor motif is referred to as a *weak* motif [48, 82, 131, 187]. This thesis focuses on discovery problems involving weak motifs.

1.2 Motif discovery and weak motif discovery

As a crucial link in revealing complex gene regulatory activities [196], effective and efficient discovery of (weak) motifs from long genomic sequences has become extremely valuable [135]. Biologists used to annotate motifs in genomic sequences by carrying out wet-lab experiments [58]. Currently, however, a huge number of genomic sequences have become available. Discovery of common motifs in these sequences can no longer be performed in the traditional way [99, 147, 176]. Alternatively, biologists can discover motifs by turning to computational means that have already led to success in the design of drugs, vaccines, and anti-microbial agents [8].

Computational motif discovery remains challenging especially for weak motifs [131, 187]. At the data level, target (short) motifs can be easily overwhelmed by long background sequences. A motif becomes more delicate with more mutations, insertions, and deletions. At the algorithmic level, the challenge comes from the sharp expansion of false instances of a weak motif in background sequences [131]. Exact algorithms can have difficulty in eliminating false motif instances efficiently especially when the weak motif is long (with tens of bases). Meanwhile, a combination of several true motif instances and a few false ones might lead to higher statistical significance than that shown by all the true ones. Thus, approximate algorithms based on statistical analysis alone are likely to miss true motif instances [27, 97].

We can discuss these challenges after giving an accurate description of *motif discovery* (MD). From the perspective of biological science, MD is to find a pattern common to a set of subsequences of background (DNA) sequences, which share biological property of interests such as being transcriptional binding sites for a regulatory factor (*Definition 1*) [10, 187]. From the perspective of computer science, MD is to find a set of statistically significant l -mer substrings of given strings over an alphabet $\{A, C, G, T\}$, which possibly show mismatches to each other (*Definition 2*). Statistical values can be information content [148] and E-value [9] etc.. The former statistic checks if a group of candidate instances of a motif are well conserved, while the latter measures if a single candidate instance appears at random.

Definitions 1 and 2 rely on an assumption that an ancestor motif evolves with mutations (but without deletions and insertions) [56]. Suppose each motif instance can show up to d mutations to the ancestor motif. It is important to note that even for small d , the differences between motif instances can be large. This poses the *Motif Challenge Problem* (MCP) of discovering weak motifs [131]. MCP is to find a 15-mer motif from 20 input sequences, of which each consists of 600 bases and contains a 15-mer motif instance with exactly 4 mutations to the target motif (*Definition 3*).

MCP is a specified benchmark for weak motif discovery [83]. To be more general, suppose there are m sequences composed of n bases, of which k sequences contain at least one l -mer motif instance. The (monad) *weak (l, d) -(k)-motif discovery* (WMD) problem is to find the starting positions of all l -mer motif instances, of which each can have up to d mismatches with the target motif (*Definition 4*) [164].

Definition 4 uses parameter k to indicate the *exactness* of a dataset. If $k=m$ (every sequence of a dataset contains motif instance), the dataset is *exact*. One Occurrence of the motif Per Sequence (*OOPS*) is one case of exact datasets. If $1 < k < m$ (several sequences of a dataset contain no motif instance), the dataset is *noisy*. Zero or One Occurrence of the motif Per Sequence (*ZOOPS*) is one case of noisy datasets [11].

Definition 4 does not take into consideration insertions/deletions occurring in motifs. However, insertions/deletions exist in reality and lead to motifs with spacers/gaps. Suppose g is the number of spacers/gaps (not exceeding a maximum length) within a motif, a dataset containing such a motif is referred to as *gapped*. Discovering a weak motif from a gapped dataset is to find an (l, g, d) -(k)-motif from m sequences of length n (gWMD, Definition 5) [48, 111]. Here, l, d, k follow the same meanings as those in Definition 4.

Definitions 4 and 5 require specified values of l, d, k or g in practice. This is difficult because prior knowledge on target motifs is always inadequate. To overcome this, biologists can provide interested ranges of the parameters; bioinformatics researchers then solve MD with strategies deriving proper combinations of them [27].

In summary, the larger the values of m, n , or d/l are, the more challenging the WMD becomes [131, 164]. Besides, discovering weak motifs from noisy datasets, i.e., $k < m$, is much more complex and difficult than from exact ones [48, 165].

1.3 Motivation

It has been introduced in the preceding sections that discovering motifs can help explain gene expression. Motif discovery has drawn considerable research interest in the last few decades [174, 187]. Consequently, a host of approximate or exact algorithms have been proposed [41, 68, 145, 146, 174]. However, weak (l, d) -motif discovery, as one of the specific problems, remains challenging due to many factors [27]. Each of the existing algorithms for WMD has strengths and limitations [164].

Approximate algorithms for WMD are efficient in execution time, but they often miss target signals. Exact pattern-driven algorithms could also suffer from this drawback. This is because they require that all motif instances converge to an exact consensus, which is not always satisfied by weak motifs. For example, the motif *TGTGAXXXGXTACA* in *E.coli CRP* dataset shows inexactness [162]. Exact sample-driven algorithms such as

those based on graph theory can discover all motif instances. However, they may consume a large amount of execution time or space. Moreover, most of the existing algorithms aim to handle exact datasets while only a handful can deal with noisy ones.

Therefore, there is much room for improving the algorithms for WMD, which underlie more reliable computational tools for detecting motifs.

1.4 Main contributions

We present exact graph-based algorithms with reduced time or space requirements. The new algorithms are more robust than the existing ones, such as those with auxiliary gene expression data [106] or those driven by patterns [136]. Besides, they can be adapted to similar problems such as protein/RNA motif discovery. The contributions include:

- (i) Tree-structured algorithms, TreeMotif-BF and TreeMotif-DF (or TreeMotif algorithms), discover weak motifs from exact datasets. TreeMotif algorithms convert the graphical representation of motif instances into the tree-structured representation. A tree whose branches contain nodes from every sequence represents motif instances.

The dynamic tree construction in a breadth/depth-first manner is novel to motif discovery. This feature enables TreeMotif algorithms to be more efficient and scalable in handling weak motifs than the existing algorithms.

- (ii) In order to reduce the high space requirement that limits the applications of WMD algorithms, another novel graph-based algorithm, RecMotif, is proposed also for exact datasets. RecMotif finds cliques in recursively constructed graphs by using the sample substrings of the input sequences. The vertices of the graphs are divided into two classes. One is the set of reference vertices, and the other is the set of

candidate vertices. The latter is selected according to the former for further graph construction.

The novelty of RecMotif lies in the recursive graph construction based on reference vertices, leading to a series of reference vertices representing the instances of the target motif. As RecMotif proceeds in a depth-first manner, it decreases the space requirement. Moreover, it also demonstrates better efficiency in execution time than TreeMotif algorithms and the other known algorithms.

- (iii) Two algorithms, nTreeMotif and nRecMotif, are extended from TreeMotif-BF and RecMotif respectively for discovering weak motifs from noisy datasets. nTreeMotif constructs trees of motif instances recursively while nRecMotif constructs cliques of motif instances in recursively built graphs.

nTreeMotif and nReMotif use two novel strategies for handling noisy datasets. One is a branch-and-bound strategy for the selection of reference sequences, and the other is for the determination of noisy sequences. With the enhancing strategies and the improved tree/graph construction, nTreeMotif/nRecMotif can discover all the (weak) motifs from a noisy dataset more efficiently than the existing algorithms.

1.5 Thesis synopsis

This thesis consists of six chapters. Chapter 1 describes the biological background and significance of motif discovery from the role of motifs in regulating gene transcription. Thereafter, it presents the challenges in WMD, followed by the motivations and main contributions of the thesis.

Chapter 2 gives a review of the existing MD algorithms and the major breakthroughs of this research.

Chapter 3 presents two tree-structured algorithms, TreeMotif-BF and TreeMotif-DF, for discovering weak motifs from exact datasets. After analyzing time and space complexities of the two algorithms, this chapter compares them with the other known algorithms based on experiments on synthetic and real benchmark datasets.

Chapter 4 introduces another algorithm, RecMotif, also for discovering weak motifs from exact datasets. RecMotif finds a motif by constructing cliques in a graph of candidate motif instances. Also, theoretical analysis and experimental comparisons are given to show the efficiency of RecMotif.

Chapter 5 presents two algorithms, nTreeMotif and nRecMotif, for discovering weak motifs from noisy datasets. nTreeMotif and nRecMotif are extended from TreeMotif-BF and RecMotif respectively. This chapter presents experiments for testing nTreeMotif and nRecMotif on noisy datasets, based on which their advantages and disadvantages are discussed through comparisons with the other known algorithms.

Chapter 6 concludes this thesis and proposes possible directions for future work.

Chapter 2

Literature Review

This chapter presents a review of the existing algorithms for (weak) motif discovery. We will first discuss the existing algorithms for monad motif discovery, which aims to extract a set of relatively short and contiguous substrings of a number of DNA sequences, such as the upstream regions of co-regulated genes [41, 87, 145, 174]. We will also briefly discuss composite motif discovery, which aims to discover motifs that appear in groups.

2.1 Introduction

Motifs could be extracted from genomic sequences by carrying out wet-lab experiments [58, 90]. For example, DNA affinities can be characterized in vitro by using protein-binding microarrays [205]. Besides, Chromatin Immunoprecipitation (ChIP) is also a powerful experimental technique to investigate the interactions between proteins and DNA. In the first step of this technique, protein and associated chromatin in a cell lysate are temporarily bound. Then, the DNA-protein complexes (chromatin-protein) are decomposed and DNA fragments are associated with the protein(s) which are selectively immunoprecipitated. Finally, the associated DNA fragments are purified and their sequences are determined. These sequences are associated with the proteins of interest. [39]. Specifically, cross-linked ChIP is mainly suited for mapping the DNA target of transcription factors [73].

Based on ChIP techniques, genome-wide ChIP-seq analysis has been increasingly used to characterize transcription factor binding [90]. However, such methods can only give peak regions of hundreds of base pairs in length where motifs are included [201]. Various tools are required to further extract binding motifs from the regions. For example, Peak-motifs is a pipeline providing a statistically reliable, time efficient framework for analyzing the such ChIP-Seq peak datasets [118]. This pipeline takes as input a number of peak regions, discovers motifs, compares them with motif databases, predicts binding site positions and returns a structured report with visualization.

Several limitations about algorithms for analyzing ChIP-seq data have been listed [90]. For example, one limitation is that most of the algorithms are not scalable for increasing the ChIP-seq peak regions (corresponding to length of background sequences) where the motifs are hidden. Another limitation is that, as with conventional computational motif discovery methods, many ChIP-seq based ones are unreliable to separate functional motifs from statistical artifacts. It takes considerable time and effort to distinguish false motifs from true ones.

Even when candidate regions are short, existing iterative motif discovery methods, such as MEME [9], cannot efficiently handle the huge amount of data from ChIP-seq/ChIP-chip experiments [110]. More effective and robust computational techniques for motif discovery are in need.

A motif discovery algorithm consists of three principal components, i.e., a model for representing the candidate motifs, a strategy for deriving and updating the model of candidate motifs, and an objective function for ranking the candidate motifs [76, 145, 160, 174]. Based on each of the components respectively, one can divide the algorithms for motif discovery into different groups.

We classify the algorithms for motif discovery according to the motif models. Meanwhile, many algorithms aim to tackle the weak (l, d) -motif discovery problem, of which

(a)	Instance 1	A	A	T	T	G	A
	Instance 2	A	G	G	T	C	C
	Instance 3	A	G	G	A	T	G
	Instance 4	A	G	G	C	G	T

(b)	position	1	2	3	4	5	6
	A	4	1	0	1	0	1
	C	0	0	0	1	1	1
	G	0	3	3	0	2	1
	T	0	0	1	2	1	1
	Consensus	A	G	G	T	G	x

(c)	position	1	2	3	4	5	6
	A	1.00	0.25	0.00	0.25	0.00	0.25
	C	0.00	0.00	0.00	0.25	0.25	0.25
	G	0.00	0.75	0.75	0.00	0.50	0.25
	T	0.00	0.00	0.25	0.50	0.25	0.25

Figure 2.1: Models of a motif: (a) four motif instances with $l=6$; (b) base counts on each site and the consensus; (c) PSFM. (Note* 'x': no dominant base.)

MCP is one specified benchmark [131]. We also discuss whether an algorithm is capable of detecting weak motifs.

2.1.1 Profile and string models

There are two kinds of motif models, which have been widely utilized in various kinds of algorithms, including (1) mismatch string or simply string model or word model [75, 95, 112]; and (2) matrix or profile model [10, 14, 17, 41, 50, 111, 120, 127, 163, 168, 195].

The string model is an l -mer consensus of all motif instances. Each base of the consensus is the dominant base that occurs most frequently on each aligned site of all motif instances. Figure 2.1 (a) shows four aligned motif instances, and Figure 2.1 (b) shows the count of each base on each site and the consensus. The matrix model contains 4 by l elements. Each element represents the significance of each base on each site. Examples include position specific scoring/frequency/weight matrix (PSSM, PSFM, PSWM) or position weight matrix (PWM), etc. Figure 2.1 (c) shows an example of PSFM.

The string representation is intuitive but incomplete; the profile representation is

comprehensive but complicated. It is feasible to determine the global optima in the space of strings; while it is usual that only local optima can be achieved in the space of profiles. For example, *string-based algorithms* can simply enumerate all the possible l -mer candidate motifs and verify their quorums in the dataset. More sophisticated algorithms can carry out the ‘enumerate-verify’ process after minimizing the space of candidates with the aid of input sequences. Meanwhile, for *profile-based algorithms*, it is almost impossible to optimize 4^l elements with continuous values. Instead, they calculate and refine the matrix models by statistical techniques, such as maximum-likelihood estimation or Bayesian inference [68].

Both kinds of algorithms have their own strengths and limitations. String-based algorithms can determine a globally optimal motif due to the intrinsic feature of exhaustive search, and thus are applicable for discovering weak motifs. Such algorithms are extremely effective when the motif is short in length, say $l < 10$, or when the motif shows up identically in each sequence. However, they have two limitations. Firstly, they are limited in applications when the length of the motif l becomes larger as it is prohibitively time-consuming to enumerate and verify all the 4^l possible candidates. For example, when l is 15, the number of candidate motifs is approximately 1×10^9 ; when $l = 20$, it is 1×10^{12} . In such cases, it might be helpful to divide a single large solution space into several small subspaces, and some are excluded for verification with the expectation that the target motif presents only in the retained ones. Secondly, as motifs usually evolve with mutations, insertions, or deletions, such algorithms could report random motifs instead of the target one [131].

The above drawbacks of string-based algorithms explain why researchers have proposed a large number of approximate profile-based ones. Such algorithms can detect long motifs by iteratively refining the profile matrix. They have excellent efficiency in execution time even in large datasets consisting of hundreds of sequences. However, because

profile-based algorithms usually apply local search strategies, there is a higher probability for them to report suboptimal motifs than string-based ones, especially when the target motif is weak [27, 97, 131, 198].

Despite the shortcomings, researchers have been spending increasingly more effort in improving the performance of string/profile-based algorithms. For the convenience of discussion in the following, let $s = \{s_i\}_{i=1}^m$ be a set of m DNA sequences. Each sequence s_i consists of a series of n_i letters over Ω , i.e., $s_i = (s_{i,j})_{j=1}^{n_i}$, where $n_i > l$ and $s_{i,j} \in \Omega$.

2.2 Profile-based algorithms

In [94], the authors use the PWM model to represent motif instances, which reflects the content of the motif instances comprehensively. They randomly initialize a PWM from the input sample sequences, and refine/update the PWM iteratively until some maximal statistical score such as information content is achieved.

Other researchers have proposed algorithms by using the same model but different search strategies, e.g., MEME [9], CONSENSUS [64], Gibbs Motif Sampler [93, 122], etc. Meanwhile, improved IUPAC-PWM was introduced to represent motifs [70]. For nucleic acids, the alphabet of IUPAC code is shown in Table 2.1. The large deviation techniques were used in the calculation of the statistical significance of an IUPAC-PWM [65].

The search strategies that have been widely adopted by profile-based motif discovery algorithms include expectation maximization (EM) or Gibbs sampling (GS) [9, 51, 93, 94, 101, 123, 170, 172]. Considering that the two strategies play a vital role in motif discovery [38, 161], we introduce them in details together with improvements.

2.2.1 Expectation maximization algorithms

The EM algorithm was first utilized in general motif discovery problem [94]. Two iterative steps lead to the EM algorithm, including the expectation (E) step and the maximization

Table 2.1: IUPAC alphabet of DNA/RNA

Code	Description	Code	Description
<i>A</i>	Adenine	<i>K</i>	<i>T/U</i> , or <i>G</i>
<i>C</i>	Cytosine	<i>W</i>	<i>T/U</i> , or <i>A</i>
<i>G</i>	Guanine	<i>S</i>	<i>C</i> or <i>G</i>
<i>T</i>	Thymine	<i>B</i>	<i>C</i> , <i>T/U</i> , or <i>G</i> (not <i>A</i>)
<i>U</i>	Uracil	<i>D</i>	<i>A</i> , <i>T/U</i> , or <i>G</i> (not <i>C</i>)
<i>R</i>	Purine (<i>A</i> or <i>G</i>)	<i>H</i>	<i>A</i> , <i>T/U</i> , or <i>C</i> (not <i>G</i>)
<i>Y</i>	Pyrimidine (<i>C</i> , <i>T/U</i>)	<i>V</i>	<i>A</i> , <i>C</i> , or <i>G</i> (not <i>T/U</i>)
<i>M</i>	<i>C</i> or <i>A</i>	<i>N</i>	Any base (<i>A</i> , <i>C</i> , <i>G</i> , <i>T/U</i>)

(M) step. The algorithm executes the two steps alternatively with an initial point (PWM) until reaching a convergence criterion. EM algorithm can handle problems with missing information, which it substitutes with the corresponding expected value. Specifically, the missing information corresponds to the positions of motif instances for motif discovery. To illustrate how EM proceeds in details, let us define the following parameters with the constraints of $a \in \Omega$, $1 \leq k \leq l$ and $q \geq 1$:

- $\rho_{a,k}^{(q)}$ (and the corresponding expectation $E\rho_{a,k}^{(q)}$): the (estimated) probability of base a showing up at position k of the motif instances at the end of q^{th} iteration;
- $f_{a,k}^{(q)}$: the observed probability of base a showing up at position k of the motif instances at the end of q^{th} iteration;
- $\rho_{a,o}^{(q)}$ (and the corresponding expectation $E\rho_{a,o}^{(q)}$): the (estimated) probability of base a showing up in the background sequences at the end of the q^{th} iteration, where the background sequences consist of the bases excluding those contained in the current motif instances;
- $f_{a,o}^{(q)}$: the observed probability of base a in the background sequences at the end of the q^{th} iteration.

The goal of EM is to determine the optimal values of these parameters so that they can maximize the likelihood estimation function $\log(L)$ in Eq. 2.1. Parameter m is the number of sequences in the dataset, and $\ln()$ is the natural logarithm.

$$\log(L) = m \sum_{k=1}^l \sum_{a \in \Omega} f_{a,k} \ln(\rho_{a,k}) + \sum_{i=1}^m (n_i - l) \sum_{a \in \Omega} f_{a,o} \ln(\rho_{a,o}) \quad (\text{Eq. 2.1})$$

To go further, more parameters are required as follows. Let:

- $M_i = (M_{i,k})_{k=1}^l$ denote the current candidate motif in s_i and $M_{i,k}$ denote the base on the k^{th} position in M_i , where $i \in [1, m]$;
- $z_{i,j}^{(q)}$ denote the estimated probability (after the q^{th} iteration) that M_i begins at position j in s_i , where $j \in [1, n_i - l + 1]$, $i \in [1, m]$ and $\sum_{j=1}^{n_i - l + 1} z_{i,j}^{(q)} = 1$;
- $Y_{i,j}$ denote an indicator variable for $z_{i,j}^{(q)}$. It equals 1 if M_i starts at position j in s_i and 0 otherwise, i.e., $z_{i,j}^{(q)} = P(Y_{i,j} = 1 | \rho^{(q-1)}, s_i)$;
- $P^0(Y_{i,j} = 1)$ denote the prior probability that M_i begins at position j in s_i , which is assumed to be uniformly distributed, i.e., $P^0(Y_{i,j} = 1) = 1/(n_i - l + 1)$;
- $C_{a,k}$ denote the base counts at the k^{th} position of motif instances $\{M_i\}_{i=1}^m$;
- $C_{a,o}$ denote the base counts in the background sequences $s - \{M_i\}_{i=1}^m$, i.e., the whole dataset after excluding all the candidate motifs.

Given the estimated ρ at the end of the $(q-1)^{th}$ iteration, i.e., $E\rho_{a,k}^{(q-1)}$, and a starting position j of a motif instance, the probability that it is sequence s_i that has a candidate motif M_i starting from position j is calculated by Equation Eq. 2.2:

$$P(s_i | E\rho^{(q-1)}, Y_{i,j} = 1) = \prod_{k=1}^l E\rho_{a,k}^{(q-1)} \quad (\text{Eq. 2.2})$$

Meanwhile, the conditional probability that M_i starts at position j in s_i is calculated by Equation Eq. 2.3:

$$z_{i,j}^{(q)} = P(Y_{i,j} = 1 | E\rho^{(q-1)}, s_i) \quad (\text{Eq. 2.3})$$

According to Bayes' rule, $P(A|B) = P(B|A) * P(A)/P(B)$. Then, by combining Equations Eq. 2.2 and Eq. 2.3, the probability $z_{i,j}^{(q)}$ can be calculated by using Equation Eq. 2.4:

$$\begin{aligned} z_{i,j}^{(q)} &= [P(s_i | E\rho^{(q-1)}, Y_{i,j} = 1) P^0(Y_{i,j} = 1)] / \sum_{k=1}^{n_i-l+1} [P(s_i | E\rho^{(q-1)}, Y_{i,k} = 1) P^0(Y_{i,k} = 1)] \\ &= P(s_i | E\rho^{(q-1)}, Y_{i,j} = 1) / \sum_{k=1}^{n_i-l+1} P(s_i | E\rho^{(q-1)}, Y_{i,k} = 1) \end{aligned} \quad (\text{Eq. 2.4})$$

The probability $z_{i,j}^{(q)}$ can be used to weight the number of bases. Let $EC_{a,k}^{(q)}$ denote the estimate of $C_{a,k}$ at the end of the q^{th} iteration. Then the expected number of bases at each position of a motif can be calculated by Equation Eq. 2.5,

$$EC_{a,k}^{(q)} = \sum_{i=1}^m \delta(a, k) z_{i,j}^{(q)} \quad (\text{Eq. 2.5})$$

Parameter $\delta(a, k) = 1$, if base a appears at position k of M_i and M_i starts at j in s_i ; otherwise $\delta(a, k) = 0$ for $1 \leq k \leq l$.

Similarly, the estimate of $C_{a,o}$, i.e., $EC_{a,o}^{(q)}$, at the end of the q^{th} iteration, is represented by Equation Eq. 2.6, where n_a is the number of base a appearing in $s_i - M_i$ (bases outside the candidate motif).

$$EC_{a,o}^{(q)} = \sum_{i=1}^m n_a z_{i,j}^{(q)} \quad (\text{Eq. 2.6})$$

With $EC_{a,k}^{(q)}$ and $EC_{a,o}^{(q)}$, the expected probabilities $E\rho_{a,k}^{(q)}$ and $E\rho_{a,o}^{(q)}$ can be updated by using Equations Eq. 2.7 and Eq. 2.8:

$$E\rho_{a,k}^{(q)} = EC_{a,k}^{(q)} / m \quad (\text{Eq. 2.7})$$

$$E\rho_{a,o}^{(q)} = EC_{a,o}^{(q)} / \sum_{i=1}^m (n_i - l) \quad (\text{Eq. 2.8})$$

Suppose the q^{th} iteration of EM is to proceed with the estimate of $\rho^{(q-1)}$, which means $E\rho^{(q-1)}$ is known at the corresponding stage. Algorithm 1 shows the EM procedure.

Algorithm 1 Expectation maximization

input : a dataset of sequences $\{s_i\}_{i=1}^m$, motif length l , and parameter $\varepsilon > 0$.
output: a set of motif instances and their starting positions.
 select a set of initial instances at random
 initialize $E\rho^{(0)}$ at random and $q \leftarrow 0$
repeat
 increase q by 1
 E step:
 calculate expectations $\{\{z_{i,j}^{(q)}\}_{j=1}^{n_i-l+1}\}_{i=1}^m$ using $\{\{E\rho_{a,k}^{(q-1)}\}_{a \in \Omega}\}_{k=1}^l$
 M step:
 calculate $E\rho_{a,k}^{(q)}$, $E\rho_{a,o}^{(q)}$ using $\{\{z_{i,j}^{(q)}\}_{j=1}^{n_i-l+1}\}_{i=1}^m$
until $|E\rho^{(q)} - E\rho^{(q-1)}| < \varepsilon$
return all motif instances (which are indicated by $\{\{z_{i,j}^{(q)}\}_{j=1}^{n_i-l+1}\}_{i=1}^m$)

Further extensions have been made since EM was proposed for motif discovery. MEME incorporates three ideas into the original EM algorithm, including, (1) taking the substrings in the sample sequences as the starting points; (2) modifying the heuristic

search to allow multiple occurrences of the motifs; (3) erasing the motifs according to a random probability after instances are found [9]. paraMEME is a parallelized implementation of MEME which aims to reduce execution time [60].

Blekas *et al.* introduced a greedy strategy combined with EM for updating motifs [21]. This algorithm shows comparable performance to MEME while avoiding redundant strategies. For example, it is capable of detecting multiple motifs without erasing discovered motifs.

An EM-like procedure *fdrMotif* updates the motif models by controlling the false discovery rate (FDR) [101]. Specifically, *fdrMotif* attempts to capture the background model with local dependence using a Markov process with the expectation that the motif model is different from the obtained model. When updating a PWM, it also considers the number of binding sites, which is not discussed in the original EM algorithm. Algorithm *fdrMotif* and MEME can find comparable true instances, while *fdrMotif* can reduce the number of false positives and thus shows a higher sensitivity, especially when many sequences without motif instances are included in a dataset.

Monte Carlo simulation is applied to help EM avoid local optima [16, 188]. Other algorithms modify EM by improving motif models. For example, CMfinder combines EM with a covariance matrix model representing RNA motifs in unaligned sequences, which shows strong robustness [199]; LOGOS designs Bayesian Markovian model, which captures the structural characteristics inside the motifs [197].

These algorithms based on EM work excellently when motifs have few mutations, deletions or insertions (indels, which lead to gapped motifs) [94]. However, they are effective only when handling strong motifs. When those factors lead to weak motifs, the results they produce may fall into local optima [41, 131, 161].

EM can also be used as a refinement to optimize PWMs that are achieved by other strategies. For instance, there are algorithms PROJECTION [27] and ProfileBranching

[134], which are applicable to weak motif discovery. As the algorithms attempt to solve the problem by first manipulating the sequences in the datasets, we introduce them as sample-driven algorithms in Section 2.3.

Overall, EM is a statistical search technique firstly used for the detection of *general* motifs (that are highly conserved). With fixed initial PWMs, EM can always produce the same results [161]. In other words, if EM does not start from a proper PWM, it cannot guarantee the discovery of the target motif [9]. With the aid of strategies for choosing initial PWMs, EM is applicable to detecting weak motifs [27].

2.2.2 Gibbs sampling algorithms

Besides EM, Gibbs Sampling (GS) is another statistical technique used in motif discovery [161]. The basic idea underlying GS is that it chooses $m-1$ sequences at random, and predicts the motif instance in the remaining sequence based on the instances of the chosen sequences. There are two main iterative steps. One is to update the PWM with several candidate instances, and the other is to randomly select new candidate instances according to some statistic. When the statistical significance of the selected candidate instances does not change anymore, the candidate instances are reported. To explain this technique, we introduce the following notations. Let

- $v_{i,j} = (s_{i,j+k-1})_{k=1}^l$ denote the l -mer substring starting at position j of s_i , where $s_{i,j+k-1}$ is the base at position k in $v_{i,j}$ and $\{j\}_{j=1}^{n_i-l+1}$;
- $v_i = \{v_{i,j}\}_{j=1}^{n_i-l+1}$ denote the set of all the l -mer substrings in s_i ;
- $C_{a,k}$ denote the count of base a appearing at position k of a set of substrings $v_{i,j}$, where $a \in \Omega$, $j \in [1, n_i - l + 1]$ and $i \in [1, m]$;
- C_a denotes the count of the base a appearing in the subset $s - s_z$, where s_z is a sequence for which the position of the motif instance is predicted by using the instances (which could be randomly selected initially) of the other sequences.

During one observation, it is possible that a base might not be observed on a position, i.e., $C_{a,k}=0$. As a result, the probability of the base appearing at that position will be zero in the PWM. To supplement a base in this case [93], the base-dependent pseudo-count b_a (which is much smaller than a normal count) is added to $C_{a,k}$ such that all elements in a PWM are non-zero. $B/||\Omega||$ is usually used as the value of b_a , where $B \approx m^{0.5}$ [93].

The matrix $\rho = [\rho_{A,k}, \rho_{C,k}, \rho_{G,k}, \rho_{T,k}]'$ represents a PWM, where $(C_{a,k} + b_a)/(m - 1 + B)$ gives the value of $\rho_{a,k}$ for $1 \leq k \leq l$. Correspondingly, let p_a denote the frequency that the base a appears in $s - s_z - \{v_{i,j_i}\}_{i=1}^m$ (i.e., the set excluding all the current motif instances and the sequence for which the motif instance is to be predicted), where $p_a = C_a / \sum_{a' \in \Omega} C_{a'}$. Let $p = \{p_A, p_C, p_G, p_T\}$ denote all the base frequencies of the background sequences.

Let v_{z,j_z} denote a sequence segment of length l that starts at position j_z in s_z , and $Q_{v_{z,j_z}}$ denote the probability of generating v_{z,j_z} according to $\rho_{a,k}$. $Q_{v_{z,j_z}} = \prod_{k=1}^l \rho_{a,k}$, where $a \in \Omega$ is the base at the k^{th} position of v_{z,j_z} . Similarly, let $P_{v_{z,j_z}}$ denote the probability of generating substring v_{z,j_z} by using p_a . $P_{v_{z,j_z}} = \prod_{k=1}^l p_a$, where a indicates the base at the k^{th} position of v_{z,j_z} . Let $A_{v_{z,j_z}} = Q_{v_{z,j_z}} / P_{v_{z,j_z}}$ denote the weight assigned to v_{z,j_z} . Then candidates are selected randomly according to the normalized weights $A_{v_{z,j_z}} / \sum_{j_z=1}^{n_z-l+1} A_{v_{z,j_z}}$. That is, v_{z,j_z} with the highest weight is most likely to be selected. Algorithm 2 gives the Gibbs Sampling procedure.

Researchers have extended GS widely in general DNA motif discovery problem, since it showed success in the protein motif discovery problem. A software Gibbs Motif Sampler can find multiple motifs one by one (with a mask strategy), which means that it finds one motif in each iteration and removes it from the current sequence before finding the next one [93, 122]. The following paragraphs discuss some aspects for improving Gibbs Sampling, including involving gene expression data, bettering motif model or background model, etc.

Algorithm 2 Gibbs sampling

input : a dataset of sequences $\{s_i\}_{i=1}^m$, motif length l , and parameter $\varepsilon > 0$.
output: a set of motif instances and their starting positions.
 randomly initialize a candidate motif in each s_i starting at j_i for all $\{i\}_1^m$
 $q \leftarrow 0; F^{(q)} \leftarrow 0$
repeat
 increase q by 1
 select one sequence s_z at random, where $z \in [1, m]$
 updating step:
 for each $s_i, i \neq z$ **do**
 calculate $\rho_{a,k}, p_a$ (according to the current positions $j'_i s$)
 end for
 sampling step:
 for each segment v_{z,j_z} in $s_z, \{j_z\}_1^{n_z-l+1}$ **do**
 calculate $Q_{v_{z,j_z}}, P_{v_{z,j_z}}, A_{v_{z,j_z}}$
 end for
 select v_{z,j_z} in s_z at random according to probability $A_{v_{z,j_z}} / \sum_{j_z=1}^{n_z-l+1} A_{v_{z,j_z}}$
 update j_z
 $F^{(q)} = \sum_{k=1}^l \sum_{a \in \Omega} C_{a,k} * \log(\rho_{a,k} / p_a)$
until $|F^{(q)} - F^{(q-1)}| < \varepsilon$
return motif instances $\{v_{z,j_z}\}_{z=1}^m$

The transcript levels of genes show reactions to the changing environment. Such genes can be monitored and clustered from comparisons at transcript levels, on which regulatory elements show mediations, and possibly vice versa [193]. By combining expression analysis (of calculating transcript abundance) and sequence alignment, Gibbs Sampling is improved and embodied in an algorithm called AlignACE [143]. AlignACE considers both strands of DNA sequences and can also search for multiple motifs that are over-represented in the upstreams of similarly expressed genes.

Chen *et al.* improved Gibbs Sampling on the motif model with gene expression data [35]. They derive the PSSM by incorporating weighted-contribution from each motif instance. The weight is measured according to the fold changes in expression obtained from microarray experiments, or the binding ratio obtained from the genome-wide location analysis. As a candidate instance with a larger fold change is more likely to be a true one than others [107], such an instance contributes more to the PSSM. This PSSM then becomes more powerful for separating true motif instances from false ones. Other motif discovery algorithms relying on gene expression data have also been proposed [30, 40, 85]. It is crucial to note that the successful discovery of motifs guided by expression similarities depends on a degree, to which the hypothesis is true that the coexpression of genes is associated with the appearances of common motifs [15, 125, 165].

Microarray binding experiments have verified that there is interdependence among pairwise positions in the binding sites [28, 86]. Several algorithms extend Gibbs Sampling by improving motif models based on this information. For example, [204] facilitates motif discovery by exploiting the interdependence by using Markov chains. Meanwhile, [34] extends AlignACE to W-AlignACE by deriving more accurate PWMs by introducing a sequence weighting-scheme, similar to [35].

In addition, it is revealed that an appearance of a base in a sequence can have impact on the appearances of its neighboring bases [45]. Therefore, several algorithms extend

Gibbs Sampling by improving the background models. For example, BioProspector uses a third Markov background model to calculate the probability of generating a potential motif candidate during updating and sampling [106]. BioProspector can handle noisy datasets and those with some sequences containing multiple motif instances, by setting threshold scores for incorporating zero, one or several candidate instances. Another priority of BioProspector is that it can find (gapped) dyad motifs.

Similarly, Gibbs Sampling is also improved by designing reliable higher order Markov background model to update the probability of generating a candidate motif in a sequence [170, 171]. Moreover, Bayesian segmentation algorithm is applied to produce a position-specific background model for inferring the number and locations of motif instances in each sequence simultaneously [105, 172].

The improvements on motif/background model discussed in the preceding paragraphs correspond to the betterment of Q_{v_z,j_z} or P_{v_z,j_z} . They impact the sampling parameter A_{v_z,j_z} , which is responsible for the selection of candidate instances. Another algorithm GibbsST introduces a comprehensive modification on A_{v_z,j_z} [81]. Specifically, $(Q_{v_z,j_z}/P_{v_z,j_z})^{1/T}$ leads to the new parameter A_{v_z,j_z} , where T is a positive parameter requiring optimization. GibbsST optimizes T by using the simulated annealing technique. Moreover, GibbsST determines a certain number of candidate instances for updating the motif model. ‘A certain number’ means that there is a high probability to include several target instances when there exist relatively few false ones. The authors show that these improvements alleviate the influence of local optima.

The Ant Colony Optimization approach has also been used in motif finding [22]. For instance, MFACO incorporates a modified form of the Gibbs Sampling technique as a local heuristic optimization search step [22]. Moreover, it searches both in the space of starting positions as well as in the space of motif patterns so that it has more chances to discover potential motifs. Similar approaches based on the artificial bee colony have also been introduced [42, 180].

Other extensions on Gibbs Sampling also exist. For example, the phylogeny information is utilized to facilitate the discovery of regulatory motifs [114, 153]; the statistical significance of the resulted motifs is analyzed after combining Gibbs Sampling with a Markov method [123].

In summary, improvements on Gibbs Sampling might lead to higher discovery accuracies. However, such algorithms cannot guarantee the discovery of motifs due to the nature of stochastic, especially when the motifs are weak.

2.2.3 Other algorithms

Besides EM and GS, other strategies for updating the profiles have also led to a number of algorithms which show feasibility. For example, from a set of possibly co-regulated genes with a common biological function, profiles can be constructed for one genome by using the experimentally determined functional sites [59]. Thereafter, comparative genomic analysis is carried out on all related genomes to derive their profiles.

The algorithm MDscan applies a progressive and regressive greedy strategy on PWM refinement [107]. First, MDscan enumerates l -mer strings on the ‘top’ sequences (possibly containing strong motif signals) as seeds. For each seed, it determines the other substrings of the ‘top’ sequences that are at a certain distance from the seed and converts all of them to one PWM. Second, it updates each PWM by adding in substrings in the non-top sequences or removing substrings contributing the current PWM, if the adding/removing can increase a scoring function. MDscan is a typical algorithm which hybridizes the profile- and string-based approaches. Other greedy algorithms also exist such as PhyloCon combines a greedy strategy with phylogenetic information [186].

iGAPK is improved from the initial framework of GAPK based on a genetic algorithm [183, 184]. A new fitness function, termed as relative model mismatch score, is introduced in iGAPK to characterize the conservation and rareness properties of DNA

motifs simultaneously. This algorithm requires some identified binding sites extracted from orthologs, which might limit its application in practice.

Genetic algorithm and random projection strategy are combined to identify (l, d) -motifs [72]. The algorithm starts with random projection to find good starting positions of candidate motifs by introducing position-weighted function and hashing each l -mer in the input sequences into the a k -dimensional space ($k < l$). The projection deduces a set of candidate motifs. Then, the algorithm uses the candidate motifs as the initial population of genetic algorithm to make a series of iterations to refine motif candidates. For motifs longer than ten bases, the algorithm shows improved accuracy over PROJECTION designed by Buhler *et al.* [27]. However, if some nucleotides in the sample are more frequent than the others (i.e., there is a biased nucleotide distribution), this algorithm becomes ineffective.

2.3 String-based algorithms

It is straightforward to represent a set of motif instances by one consensus string. Then, the problem become one that identifies an optimal string (which can be measured by its distance to all the possible instances).

Median string search (MSS) is one of the earliest strategies to find a consensus motif, which minimizes the distance cost of candidate consensus to all input sequences [80, 102]. In an exhaustive form, MSS aligns all combinations of l -mers from $AA...A$, $AA...C$, ..., to $TT...T$ with l -mer substrings of given sequences. It uses an objective function of total distance to determine the significance of an l -mer candidate consensus which is the sum of the smallest Hamming distances of the candidate to each of given sequences. The smaller the total distance is, the higher the probability (that the candidate is a true motif) is. Thus, the result includes l -mer candidates that lead to the minimum total distance to a list of l -mer substrings of given sequences. However, as mutations could

occur in motif instances, some sub-optimal candidates can preserve better than the true ones, although it is very unlikely. As a result, MSS is likely to miss true instances. Meanwhile, this brute-force enumeration is not applicable in finding long motifs as the problem is NP-hard [102, 103].

Therefore, a large number of algorithms present improvements based on finding motifs by generating and verifying new candidate patterns. We refer to these algorithms as *pattern-driven algorithms*. Besides pattern-driven algorithms, there is another class of algorithms that derive (instances of) motifs only relying on substrings of given sequences, e.g., WINNOWER [131], etc. We refer to these algorithms as *sample-driven algorithms*. In Sections 2.3.1 and 2.3.2, we discuss some existing string-based algorithms according to whether they are pattern-driven or sample-driven.

2.3.1 Pattern-driven algorithms

Naive exhaustive enumeration is applicable only to finding short motifs such as the cases with $l \leq 15$ [24, 129, 194]. It is prohibitive to perform an exhaustive verification of all the candidate motifs to guarantee the discovery of a long target motif [102].

Researchers have presented various methods to improve the efficiency of the exact enumeration [18, 156–158, 173, 175]. There are mainly two strategies to reduce the number of candidate motifs for verification within a given dataset [24]. The first strategy is to associate all the candidate motifs in particular data structures, e.g., *prefix/suffix trees*. In this way, the verification of a candidate motif determines that of the others that associate with the candidate motif. The second one is to enumerate only candidate motifs that show some relationship to selected *references* (of a number of l -mer substrings of given sequences). This strategy excludes the non-related candidate motifs directly. We introduce some typical pattern-driven algorithms by classifying them as *tree-based algorithms* or *reference-based algorithms*.

2.3.1.1 Tree-based algorithms

The algorithms [32, 77, 111, 132, 144] introduce prefix/suffix trees to avoid enumerating all candidate l -mer strings. Prefix/suffix trees explore candidate motifs in a lexicographic manner, in which each node indicates a base of the target motif. These algorithms guide the enumeration of candidate motifs by using the l -mer substrings of the input sequences. In other words, the input substrings determine the necessity for the prefix/suffix (of a candidate motif on the tree) to branch. A prefix/suffix branching with l nodes on the tree represents a target motif.

SPELLER [144] starts with a null word (i.e., root of the tree of motifs). Recursively, it adds an additional base to the tail of the word, and keeps the new word for further extension if at least q (a quorum) instances of it appear in at least q sequences. Note that each instance can mismatch up to d positions with the new word. For any l' -mer ($l' \leq l$), if it cannot satisfy the quorum q , there is no need to verify the candidate motifs that have the l' -mer as a prefix. In other words, there is no need to traverse all the subbranches following the l' -mer. This could reduce the number of candidates.

The algorithm MITRA uses a mismatch tree to split the space of all the possible 4^l candidates into disjoint subspaces, which respectively include a small number of candidates that start with the same prefix [48]. A mismatch tree is a rooted tree with each internal node having 4 branches, each labeled with a distinct base in Ω . The maximum depth of the tree is l , and a branch with l nodes can lead to a motif. Each node corresponds to a subspace of all the 4^l candidates with an x -mer prefix, which consists of x bases on the path from the root to the node. In addition, each node points to all the d -neighbors of the x -mer prefix. A d -neighbor of the x -mer prefix is an l -mer substrings (of the given sequences), of which the first x bases and the prefix have a Hamming distance not more than d .

MITRA achieves efficiency by pruning the search space of motif candidates in two different ways. One easy way is to count the number of d -neighbors at each node in the mismatch tree. The improved algorithm is referred to as MITRA-Count. If a node has less than q d -neighbors from the given sequences, there is no need to search the corresponding subspace any more. An alternative way adapted from WINNOWER is to construct graphs using the l -mer substrings (of the given sequences) associated with the node. This improved algorithm is referred to as MITRA-Graph. If there is no q -clique in the graph, there is no need to search the corresponding subspace any more.

MITRA-Count or MITRA-Graph is effective in discovering monad motifs. On that basis, MITRA-Dyad converts the dyad motif discovery problem into a monad motif discovery problem by creating virtual data from the real data. MITRA-Dyad is able to discover dyad motifs with $l_1+l_2>30$ (l_1 and l_2 are length of sub-motifs), on which other pattern-driven algorithms may fail. However, when the distance between the sub-motifs is large, MITRA-Dyad becomes inefficient as too many virtual strings have to be generated [48].

Apriori-Motif also uses a breadth-first search strategy over the tree of consensus motifs [77]. It iteratively constructs suffix trees of given sequences to control the growth of the consensus tree. After all construction, spelling along a specific path of the consensus tree leads to a possible consensus motif. This algorithm does not require an exact specification of the length of a motif. However, it requires more than 3 hours to handle the Motif Challenge Problem on an Intel computer with 2 GHz processor.

TrieAMD, with several modifications on the classic Apriori algorithm [4], records all biological sequences containing motifs in a trie structure and spells motifs along the branches [5]. After extracting motifs, it applies a normalized information content to distinguish motifs in real promoter sequences. One drawback of this algorithm is that it requires at least one exact occurrence of the targeted motif.

RISOTOO [132] utilizes the same idea as SPELLER but with an enhanced strategy for pruning the tree of candidate motifs. Specifically, if an l' -mer cannot branch any more to form any l -mer motif satisfying the quorum q , then besides the branches pruned above, RISOTOO also prunes all the other branches whose suffixes include the l' -mer. For example, if a 3-mer 'cgt' cannot find instances satisfying the quorum q , then it is true that a 4-mer 'acgt' cannot either. This is because when we can consider the former has 3 conditions ($c \& g \& t$) while the latter has 4 conditions ($a \& c \& g \& t$), the latter is a more stringent requirement than the former to satisfy the quorum q .

In another similar algorithm Census [49], the authors expect that the majority of the true motif instances share the prefix of the target motif. Therefore, Census examines the prefixes of candidates instead of the entire compositions. Moreover, it verifies a candidate consensus with a qualified prefix. This strategy favors the discovery of long motifs. To carry this out, Census arranges the l -mer substrings of each sequence (other than the 4^l candidate motifs) in a lexicographic tree. The l' -mer prefix of any valid motif must approximately match the nodes on an l' -mer branch of each tree, where the number of mismatches is less than or equal to d . On that basis, Census iteratively searches and verifies the prefix of the enumerated motif at increasing length.

A naive voting algorithm enumerates all 4^l candidate motifs, which determines a candidate as true if the candidate can receive at least one vote from one sequence [36]. A vote for a candidate from a sequence means that there is an l -mer substring of the sequence with up to d mismatches with the candidate. An improved voting algorithm successively hashes and verifies the l -mer candidates with the same l' -mer suffixes, where $l' < l$. This reduces the space requirement. In addition, the improved algorithm uses projection to map long l -mer candidates into short l'' -mer ones. It determines l'' that is sufficiently less than l , in which case it is able to enumerate and vote the l'' -mer subsequences. Meanwhile, l'' must be relatively large so that the projection preserves

the characteristics of the original l -mer candidates. The improved voting algorithm can achieve approximations for long (weak) motifs, such as a (40, 15)-motif [36].

2.3.1.2 Reference-based algorithms

The above algorithms enumerate candidate motifs without any guidance and verify the candidates with the aid of all the input sequences. Recent algorithms select a small number of the input sequences as references to guide the enumeration of candidate motifs and verify the candidate motifs by using the remaining sequences. Essentially, they reduce the candidate space to the neighborhoods of a number of l -mer substrings of a few input sequences. This has led to a significant improvement on the algorithmic efficiency. The following paragraphs present a few typical algorithms.

Let us define that two strings are d -neighbors if their Hamming distance is within d . The idea of PatternBranching is straightforward [134]. For each l -mer substring of a randomly selected reference sequence, PatternBranching generates its d -neighbors, evaluates the significance of each d -neighbor over the remaining input sequences, and branches to the most significant ones. With a new candidate, PatternBranching repeats generating, evaluating, and branching procedure until the significance of a candidate does not improve further. ProfileBranching follows the same idea but uses a profile to represent motifs. These algorithms can report results quickly. However, due to the probabilistic characteristic, they cannot guarantee to find the true motif all the time.

PMS1 [136] is based on the following idea. A common ancestor motif evolves to all its instances, thus it is a common d -neighbor of the instances. Specifically, PMS1 constructs the d -neighborhoods of the l -mer substrings of all the input sequences and finds out the common d -neighbors shared by a certain number of l -mer substrings. Algorithm 3 shows the procedure of PMS1.

The time complexity of PMS1 is $O(mnC_d^l \|\Omega\|^d l/w)$ (w is the word length of the computer). PMS1 can solve weak motif problems efficiently. For example, it derives a

Algorithm 3 Planted motif search 1 (PMS1)

input : a dataset of sequences $\{s_i\}_{i=1}^m$, motif length l , number d .
output: a consensus of a set of motif instances.
 initialize all l -mers from each s_i into C_i for all $\{i\}_{i=1}^m$
for each substring u from each $\{C_i\}_{i=1}^m$ **do**
 generate all l -mers v such that u and v are within d (and let C'_i be the set of v related to C_i)
end for
 sort all the l -mers in C'_i (and let L_{s_i} be the sorted list of C'_i)
 obtain the overlap $M = \cap_{i=1}^m L_{s_i}$
return the l -mers in M

(11, 3)-motif in about one minute, where $m=20$ and $n=600$. On the other hand, the space complexity of PMS1 is $O(nlN(l, d)/w)$, where $N(l, d) = \sum_{i=1}^d C_i^l (\|\Omega\| - 1)^d$. As l and d increase, PMS1 requires a large amount of space, which makes the algorithm infeasible.

One improved algorithm PMSi reduces the space requirement of PMS1 [43]. PMSi generates the initial set of candidate motifs as $M = L_{s_1} \cap L_{s_2}$. Then it generates other candidate sets $\{L_i = L_{s_{2i-1}} \cap L_{s_{2i}}\}_{i=2}^{m/2}$, i.e., $L_{s_3} \cap L_{s_4}$, $L_{s_5} \cap L_{s_6}$, and so on, to prune M (i.e., $M = M \cap L_i$). As $L_i \subset L_{s_{2i-1}}$ and $L_i \subset L_{s_{2i}}$, PMSi records fewer d -neighbors and thus requires a lower space than PMS1. PMSi can derive a (13, 4)-motif (in 20 sequences of length 600) in minutes on a Pentium 2.4GHz machine with 1 GB RAM.

Another improved algorithm PMSP [43] generates the d -neighbors for each l -mer substring u of a selected reference sequence. For each generated l -mer v , PMSP outputs it as a motif if it can find at least one d -neighbor from each of the remaining sequences. The d -neighbors of v must show up to $2d$ distance from u ; therefore for each v , PMSP only verifies the l -mer substrings of the remaining sequences that are $2d$ -neighbors of u . This helps reduce the execution time. Meanwhile, the space complexity of PMSP is $O(mn^2)$, which shows significant improvement compared with PMS1. PMSP can solve the (15, 5)-motif problem in 35 minutes (on the same Pentium machine).

PMSprune improves PMSP algorithm by introducing several strategies [44]:

- For each l -mer u of a reference sequence, PMSprune represents its d -neighbors as nodes of a tree with height up to d . It verifies the d -neighbors with a branch-and-bound strategy. Specifically, for each l -mer v in the tree with height h , it uses the value of $d'_H(v, s)$ and h to prune the descendants of v , where $d'_H(v, s) = \max_{i=1}^m \min_{r \in C_i} d_H(v, r)$ with $d_H(v, r)$ denoting the Hamming distance between l -mers (v, r) . This strategy reduces the number of motif candidates to confirm.
- It reduces the set of all the l -mer substrings of $\{s_i\}_{i=2}^m$ to one set consisting of $(2d-h)$ -neighbors of v for calculating $d'_H(v', s)$, where v' is a child node of v .
- It calculates $d'_H(v', s)$ based on $d'_H(v, s)$, where v' is a child node of v .

Compared to PMSP, PMSi and PMS1, PMSprune has a lower space complexity of $O(mn^2)$. It also shows improved efficiency in execution time. For instance, it solves the (19, 7)-motif problem within 10 hours (on the same Pentium machine mentioned earlier). However, the time complexity $O(mn^2N(l, d))$ of PMSprune indicates that it is not scalable on the combination of l and d . qPMSprune enhances PMSprune by verifying the d -neighbors of the l -mer substrings of $m-k+1$ sequences, which is capable of discovering motifs from noisy datasets.

The PMS series are also improved with a common technique [137]. The improved algorithms obtain initial candidate motifs from a small number of selected sequences and verify them by using the remaining sequences. Empirically, they demonstrate the maximum speedup by selecting 40 percent of the sequences.

iTriplet is a rule-based algorithm for weak motif discovery [66]. First, it selects two sequences randomly as references. Second, it constructs a number of triplets. Each triplet consists of three l -mer substrings respectively in the two reference sequences and one of the remaining sequences. Third, it generates candidate motifs from each triplet based on several rules and associates the candidates with the sequences. If a candidate

has associations with a certain number of sequences, it is a potential motif. iTriplet can detect highly degenerate motifs such as a $(18, 6)$ -motif. However, the number of candidate motifs becomes extremely large along with the increase of l and d , which could limit the application of iTriplet. The effect of varying l and d on the performance of motif discovery algorithms will be further discussed in the experimental comparisons.

2.3.1.3 Other pattern-driven algorithms

The algorithms introduced in the preceding sections reduce candidate motifs by handling the substrings of the input sequences. Meanwhile, several algorithms handle the subsequences of the input sequences. For example, TEIRESIAS [141] scans for short but significant subsequences of the input sequences and recombines the elementary subsequences into long ones by using a convolution process.

As discussed, a motif instance shows up to Hamming distance d from the target motif, which means there are errors in up to d positions for spelling the instance. In view of this fact, MULTIPROFILER recovers the ‘backbone’ of a motif by constructing multi-positional profiles based on all the possible instances [82]. Such profiles can alleviate the influence of wrong spellings in some positions of the instances, especially when they absorb a sufficient number of correct motif instances. MULTIPROFILER can detect a $(15, 4)$ -motif with background sequences being as long as 3000.

MobyDick [29] builds up a dictionary of words representing possible motifs. The enumerated words have high over-representativeness in the input sequences. A progressive concatenation of short words can lead to long ones. MobyDick reads the ‘texts’ (i.e., the input sequences) by using the words and eliminates words by checking whether their significance is due to the overlapping with real motifs.

GRISOTTO improves the enumeration algorithm RISOTTO on analyzing its outputs by using a greedy procedure [33, 132]. It applies prior information of motifs from several

different sources, such as orthologous conservation, DNA duplex stability, and nucleosome positioning, to supervise the optimization procedure of the greedy process.

Several improved algorithms also represent motifs as words in a dictionary. A stochastic probabilistic matrix is adopted to represent words, which form a stochastic dictionary [62]. WordSpy [185] consists of word counting and statistical modeling. A difference of WordSpy from MobyDick is that it assumes that a model of (*dictionary, grammar*) generates the motif instances and the background sequences. Based on word sampling, it optimizes the model of (*dictionary, grammar*) by using a Hidden Markov Model.

2.3.2 Sample-driven algorithms

Pattern-driven algorithms start from the generation of candidate motifs according to the parameters l and d . Increasing l and d can lead to an increasing number of candidate motifs, which results in algorithmic inefficiency. Sample-driven algorithms only deal with a limited number of l -mer sample substrings of input sequences. Most profile-based algorithms (in Section 2.2) recover motifs by using this strategy. In this regard, sample-driven algorithms are superior to pattern-driven ones.

In this section, we discuss a few string-based algorithms that are driven by sample substrings. Most of these algorithms use the strategy of finding cliques in graphs, where each clique is a possible motif.

2.3.2.1 Graph-based algorithms

WINNOWER is a graph-based algorithm to solve the weak (l, d) -motif problem [131]. As any true motif instance can show up to Hamming distance d from the consensus motif, any two true motif instances can show up to Hamming distance $2d$ from each other. On that basis, WINNOWER constructs a multi-partite graph by representing l -mer substrings of the sample sequences as the vertices and connecting any pair of vertices (of different sequences) with an edge if their Hamming distance is up to $2d$. The original

problem becomes one that finds cliques in graphs, where a clique is a set of vertices in which any pairs are within $2d$. By iteratively deleting inconsistent edges and retaining the extendable cliques, WINNOWER can give satisfactory performance as the length of sequences grows up to 1000 for the $(15, 4)$ -motif problem.

However, WINNOWER requires substantial time and space and becomes less effective when finding weak motifs in long background sequences. An improved cWINNOWER algorithm imposes an additional filtering constraint to WINNOWER to decide whether edges should be deleted or retained. The modification substantially improves the execution time compared with WINNOWER. However, it still takes hours to produce results for a $(15, 4)$ -motif problem [104].

DPCFG is also a graph-based algorithm for weak motif discovery, which guarantees to detect all the possible weak motifs [198]. This algorithm uses a dynamic programming technique to find cliques from initially constructed graphs, with each clique indicating one set of motif instances. DPCFG has a better performance than WINNOWER for sequences as long as 2000 for the $(15, 4)$ -motif problem. GWM2 extends the algorithm to handle noisy datasets [67].

Other graph-based algorithms have also been proposed. For example, a graph-tree algorithm [1] improves the efficiency in execution time based on converting the clique finding problem (in graphs) into the vertex-cover problem (in trees). The algorithm is implemented by using parallelizing strategies. Using 32 processors running at 500 MHz, the algorithm obtains cliques of motif instances in seconds. MCL-WMR is a heuristic that uses a Markov Cluster algorithm to search for cliques of motif instances in weighted graphs [23]. MCL-WMR has shown competitive accuracy and execution time. MotifCut formulates the motif discovery problem as finding maximum density subgraphs of a weighted graph considering base dependencies, where a graph is constructed by using l -mer substrings of all the input sequences [53].

These graph-based algorithms have a common two-step strategy. First, they try to construct an almost complete graph or a set of disconnected graphs by drawing pairwise edges between any two vertices that satisfy a similarity measure. Second, they delete the spurious edges (non-signal edges) of the graphs to obtain vertices that represent the motif instances; or they search maximum density subgraphs to recover the motif. The separating strategy may affect their performance. There could be as many as 20,000 false connections for each true connection in the (15, 4)-motif problem [131], thus connecting two vertices arbitrarily and later deleting this connection can incur a large amount of unnecessary computation. This could make an algorithm time-prohibitive when the number of vertices becomes extremely large.

2.3.2.2 Other sample-driven algorithms

Another algorithm SP-STAR [131] is a conventional local search algorithm. SP-STAR adopts a search strategy similar to the median string search, but all candidates for searching are from sample sequences. In addition, by modifying the objective function as sum-of-pairs scoring, SP-STAR achieves a better discrimination between the true motif instances and the spurious substrings.

The algorithm PROJECTION projects all the sample substrings of input sequences into different buckets according to several randomly selected positions, expecting that there is a higher probability to recover the consensus motif from the buckets with a larger number of substrings than the others [27]. That is, PROJECTION takes the profiles derived from these buckets as the starting points of EM with the expectation that they can be better than randomly selected profiles to reach true motifs. This random projection technique enhances the performance of weak motif discovery algorithms by generating excellent starting points for local search strategies.

PROJECTION shows higher performance than WINNOWER for various (l, d) -motif problems. However, the performance of PROJECTION is still sensitive to sequence

length, which begins to drop significantly above sequence length of 1400 for the (15, 4)-motif problem [27]. As random motif instances might occur when sequences are long enough, PROJECTION may be trapped in local optima as it is a local search method. A uniform projection technique is used to improve the efficiency in execution time by selecting only a part of projections from all the possible projections [138].

PosMotif allows arbitrarily ignored positions within the non-conserved portion of a string motif model [203]. It starts with length- l' subsequences ($l' < l$) of a length- l motif. For each combination of ignored positions, a Markov chain is used to model the background distribution of a length- l' string. This leads to a series of Markov chains that can model nucleotide dependencies in a long range of positions. Running time of this method can be doubled when increasing the length of a motif by 1. It takes up to hours for finding a motif of length 20 (in tens of sequences up to 2000 bp).

In summary, although existing graph-based algorithms are inefficient in graph construction and clique (of motif instances) finding, they have shown some superiority in practice, especially more efficient on handling long motifs than the pattern-driven algorithms. In addition, graph-based algorithms are capable of finding weak motifs in inexact expressions [187], on which pattern-driven algorithms might fail. In view of these facts, this thesis will present graph-based algorithms attempting to reduce redundancies in current algorithms so as to improve their efficiency and widen the applications.

2.4 Gapped motif discovery

A transcriptional factor is likely to act with other factors in higher organisms to realize specific regulatory behaviors. Their binding sites appear near each other with gaps in-between. Such a group of motifs can be assembled as one 'large' motif consisting of them as sub-motifs. In this case, the gaps can be represented as insertions or deletions occurring in the motif instances. This relates to the gapped data, i.e., data with the

Table 2.2: Example: dyad motifs in *P. horikoshii*

sequence name	motif information		
	left component	length of gap	right component
<i>purA</i>	attaacatagccctgtcaaa	(23)	ctttacttacccttttgtaa
<i>purB</i>	atttctacaaatgtcaaa	(23)	tttaccgtgaaaatggtgat
<i>purC</i>	tttgccagatatatgtctaa	(22)	ttttacataaacatggtgaa
<i>purD</i>	gttaacacgtttatgtaaac	(23)	tttgacttaaataatggtgat
<i>purE</i>	attagcacatatatgtagaa	(23)	attgacattaaattgctagg
<i>purF</i>	ttcaccatgtttatgtaaaa	(22)	ttagacatatatctggcaaa
<i>purL-I</i>	attaacataaaaatgttaaa	(22)	ttttacatctttttggtaat
<i>purL-II</i>	attgacattttcttgtaaa	(22)	ttttacatttttctggcaaa
<i>put</i>	ttaaacatatattatgttaaa	(22)	tttaacatttatacgtcaat
<i>purD paralog</i>	tttgcataagaatatgttaaa	(23)	tttaacatgaacatggtgat
<i>purine permease</i>	gttgacaaaattttgttaaa	(23)	tttgacaaaacaaaggcaat
<i>formate dehydrogenase</i>	ttggtcatatttatgtaaat	(23)	tttaacatttatatgctcaa

gapped motifs or composite motifs [48, 51, 79, 111, 176, 191]. For instance, *dyad motifs* are gapped motifs consisting of two sub-motifs. Table 2.2 shows dyad motifs [59].

A *gapped motif* is, for a coverage ratio $r < 1$, a string of length L that consists of characters in $\{A, C, G, T, x\}$, with at least $\lfloor rL \rfloor$ characters in $\{A, C, G, T\}$ [111, 192]. Each maximal substring of consecutive x represents a *spacer/gap*, and each maximal substring of other characters represents a *sub-motif*.

Mathematically, a gapped motif can be modeled as a pair (M, d) , where M is a p -tuple of single models $(M_1, M_2, \dots, M_p)$, and d is a $(p-1)$ -tuple of triplets $((d_{min_1}, d_{max_1}, \delta_1), (d_{min_2}, d_{max_2}, \delta_2), \dots, (d_{min_{p-1}}, d_{max_{p-1}}, \delta_{p-1}))$. Parameter p is a positive integer giving the number of sub-motifs (and thus the number of gaps is $p-1$), M_i is a string over Ω representing a sub-motif, and d_{min_j}, d_{max_j} ($d_{min_j} < d_{max_j}$) and δ_j are non-negative integers defining the distance between each pair of sub-motifs (as given below).

Given m sequences and an integer quorum $1 < q \leq m$, a gapped candidate (M, d) is valid if the following conditions hold. For each occurrence u_i of M_i ($1 \leq i \leq p$), there are occurrences $u_1, \dots, u_{i-1}, u_{i+1}, \dots, u_p$ of $M_1, \dots, M_{i-1}, M_{i+1}, \dots, M_p$ respectively, such that

- (1) the Hamming distance between u_k and M_k for all $1 \leq k \leq p$ is not more than an error parameter e (i.e., the number of mismatches/mutations);

- (2) $u_1, \dots, u_{i-1}, u_i, u_{i+1}, \dots, u_p$ belong to the same sequence from the dataset;
- (3) there is d_j with $d_{\min_j} + \delta_j \leq d_j \leq d_{\max_j} - \delta_j$ such that the distance between the end of u_i and the start position of u_{i+1} in the sequence is in $[d_j - \delta_j, d_j + \delta_j]$.

Given a set s of m sequences, a positive integer e , and a positive quorum $q \leq m$, the *gapped motif discovery problem* (GMD) is to find all possible motifs (M, d) . When $p=2$, the problem is to find all motifs of the form $((M_1, M_2), (d_{\min_1}, d_{\max_1}, \delta_1))$, the problem of discovering dyad motifs.

One difficulty in discovering gapped motifs is that one of the component motifs (sub-motifs) may be extremely weak, making it beyond discovery using the existing monad motif discovery tools. For example, two transcriptional factors *URS1* and *UASH* regulate a set of yeast *S.cerevisiae* genes, of which the binding sites appear quite near each other with one fixed gap. Although the *URS1* binding site conserves well and is easy to locate by monad motif discovery tools, the *UASH* binding site is too weak to discover [61].

A few algorithms based on profiles or strings are available for gapped motif discovery [32, 69, 91, 132, 139, 150–152, 175, 176, 192]. BioProspector can discover dyad motifs using Gibbs Sampling [106]. Other algorithms using GS also exist [51, 61, 142]. Besides, profile-based algorithms using EM [31, 159], Hidden Markov Models [12, 54, 55, 155] and Genetic Algorithms [2, 3, 100, 182, 202] have also been proposed for gapped motif discovery. While profile-based algorithms can produce results in a short time, they cannot guarantee that the motifs will always be discovered.

2.5 Summary

In this chapter, we have briefly introduced the state-of-the-art of motif discovery. We have also analyzed in detail a number of available motif discovery algorithms and pointed out the corresponding advantages and disadvantages. (The features of several typical algorithms are shown in Table 2.3, where ‘M’ is short for model, ‘S’ for searching strategy,

‘A’ for accuracy, and ‘T’ for type of a dataset. A model can be either string/word ‘s/w’ or profile ‘p’. A strategy can be either pattern ‘pa’ or sample ‘sa’. Accuracy can be either exact ‘ex’ or probabilistic ‘pr’. The type of a dataset includes either OOPS ‘O’, noisy ‘N’, ZOOPS ‘Z’, Dyad ‘D’ or gapped ‘G’.)

Most of the algorithms aim to tackle monad motif discovery, i.e., discovering motifs one by one. From the literature, we have seen that profile-based algorithms are likely to miss true motifs; meanwhile, although exact string-based algorithms can discover all the possible motifs, they require a large amount of execution time or space. This could limit their applications in practice, especially when the motifs are extremely weak. The existing algorithms need further improvements to efficiently provide reliable results for biologists.

Table 2.3: Typical motif discovery algorithms

Algorithm	M	S	A	T	Brief Description	Year	
Seeder [50]	p	sa	pa	ex	O	Others	2008
MULTIPROFILER [82]					O	Others	2002
EOMM [97]				ex	O	Suffix Tree	2005
GADEM [100]				pr	D	EM+GA	2009
Co-Bind [61]			GS			2001	
SeSiMCMC [51]			GS			2005	
Helicis [91]			GS			2007	
GA-DPAF [202]			GS+GA			2009	
Dyad analysis [176]			Others			2000	
MEME [9, 10]				pr	N	EM	1994
ProfileBranching [134]			EM+Branching			2003	
fdrMotif [101]			EM like			2008	
CONSENSUS [64, 65]			Others (Greedy)			1999	
Gibbs Motif Sampler [93]			GS			1993	
MotifSampler [170]			GS			2001	
Gibbs Recursive Sampler [172]			GS (Recursive)			2003	
W-AlignACE [34]			O			GS	2008
ANN-SPCE [195]				GS+ANN	2000		
CMfinder [199]			O/N	EM-like	2006		
GIMSAN [123]			O/Z	GS (Hybrid)	2008		
MERMAID [70]			O/D	Others	2003		
BIOPROSPECTOR [106]			N/D	GS	2001		
iTriplet [66]	s/w	pa	ex	O/N	Hashing	2009	
MITRA [48]				N/D	Mismatch Tree	2002	
IRE ^{1,2} [178]					Others (Sets Intersection)	2000	
MotifEnumerator [167]				O/G	Others (Enumeration)	2006	
PMS1 [136]			O/N	O/N	Others (Sets Intersection)	2005	
PMSi/PMSP [43]					Others (Sets Intersection)	2006	
PMSprune [44]						2007	
Apriori-Motif [77]					Pattern Tree	2010	
PMS4 [137]						2011	
MSS [102]			pr	O	Median string search	1999	
SP-SATR [131]				N/G	Median string search	2000	
FSoUL [127]				N	Suffix Tree	2001	
SPACE [192]			sa	G	Others	2007	
WINNOWER [131]				N	Graph	2000	
FootPrinter [17, 20]					Graph (PF)	2002	
DPCFG [198]				O	Graph (List)	2004	
MCL-WMR [23]					Graph (Weighted)	2007	
SuffixTree [111]				O/N	Suffix Tree	2000	
cWINNOWER [104]					Graph	2003	
SPADCP [1]					Tree	2003	
PROJECTION [27]			pr	N	EM+Random Projection	2002	
PatternBranching [134]					Branching	2003	
IWMGA [126]				O	GA	2006	
MotifCut [53]				O/N	Graph (Weighted)	2006	

Chapter 3

Tree-structured Algorithms for WMD in Exact Datasets

In this chapter, we first present an exact algorithm TreeMotif-BF for discovering weak motifs from exact datasets. TreeMotif-BF converts the graph representation of motif instances into a tree-structured representation. A final tree represents motif instances, if its branches consist of nodes from every sequence. TreeMotif-BF features a breadth-first construction of the tree in the process of motif discovery, while previous graph-based algorithms usually have two separate processes: graph construction and spurious edges deletion. This difference enables TreeMotif-BF to be more efficient and scalable in handling longer weak motifs than the existing algorithms.

TreeMotif-BF can consume a large amount of space. This limits its application in discovering very weak motifs. To overcome this, we present another algorithm TreeMotif-DF that constructs trees of motif instances in a depth-first manner. TreeMotif-DF is efficient in space, but it shows a little decreased efficiency in execution time.

3.1 Introduction

Taking DPCFG algorithm [198] as an example, we first explain why the current graph-based algorithms might not be efficient. On that basis, we introduce TreeMotif algorithms that improve the computational efficiency.

To begin with, assume a dataset $s = \{s_i\}_{i=1}^m$, where $s_i = (s_{i,j})_{j=1}^n$ denotes the i^{th} sequence in s and $s_{i,j} \in \Omega$. For simplicity, assume that the dataset s satisfies OOPS, and $\{s_i\}_{i=1}^m$ have the same length $n (\gg l)$. Let $v_i = \{v_{i,j}\}_{j=1}^{n-l+1}$ be the set of all l -mer substrings/nodes in sequence s_i , where $v_{i,j} = (s_{i,j'})_{j'=j}^{j+l-1}$ represents the substring starting at site j of the sequence. Let function D compute the Hamming distance between two nodes and $D_{\max}$ compute the maximum pairwise Hamming distance among several nodes.

3.1.1 Inefficiency of DPCFG algorithm

DPCFG detects (l, d) -motifs as follows. Starting with substrings of a reference sequence, it represents substrings of the other sequences as vertices of m -partite graphs (of which the construction given in the next paragraph). Thereafter, it constructs a list (of vertices) of size i ($i \in [2, m]$) for each vertex of a graph based on all the lists of size $i-1$. A list of size m can be constructed on some vertices, if a clique of size m (or m -clique) exists in this graph. This list represents a possible motif.

Suppose s_1 is the reference sequence having nodes $\{v_{1,t}\}_{t=1}^{n-l+1}$. For $\{i\}_{i=2}^m$, let P_i be the set of the nodes selected from $v_i = \{v_{i,j}\}_{j=1}^{n-l+1}$ according to a reference node $v_{1,t}$, and $\|P_i\|$ gives the number of nodes. Initially, for $\{i\}_{i=2}^m$, P_i is set to be null. For each $v_{i,j}$ from v_i , put it in P_i if $D(v_{1,t}, v_{i,j}) \leq 2d$. This is the graph construction of DPCFG.

Let $L_{v_{i,j}}$ denote the list for a vertex $v_{i,j}$ in the graph. DPCFG creates $L_{v_{i,j}}$ at each vertex $v_{i,j}$ in P_i by incorporating $L_{v_{i-1,j'}}$ at each vertex $v_{i-1,j'}$ in P_{i-1} , if $D(v_{i,j}, v_{i-1,j'}) \leq 2d$. The dynamic strategy is unnecessary to check the lists before P_{i-1} when constructing lists for P_i . However, the overall running time of the algorithm is not reduced in terms of the number of Hamming distance calculation. For example, for a single list in P_{i-1} , $i-1$ times of Hamming distance calculation must be executed to check if the list and a new vertex can form an i -clique. Moreover, duplication of lists in P_{i-1} is necessary for the construction of lists in P_i , which aims to retain the same information (on the existing

lists) for each further construction. However, such duplications can be redundant, as shown by the following example. Suppose there are nodes grouped by P_1 to P_3 , where $\{R\} \subseteq P_1$, $\{A, B\} \subseteq P_2$, $\{C, D, E, F\} \subseteq P_3$, as shown in Figure 3.1.

Figure 3.1: Example: nodes for list construction by DPCFG

Assume that, each node in sets $\{R, A, C\}$, $\{R, A, D\}$, $\{R, B, D\}$, $\{R, B, E\}$, $\{R, B, F\}$, has a Hamming distance equal to or less than $2d$ with the others in the same set. Besides, suppose there are new nodes: H , I and J in P_4 to construct new lists. Assume that H has a Hamming distances not more than $2d$ with nodes R, A, B, E, F and more than $2d$ with nodes C, D . Also, assume that, node I has a Hamming distance not more than $2d$ with nodes R, B, C, E and more than $2d$ with nodes A, D, F ; node J has a Hamming distances not more than $2d$ with nodes B, C, D, E, F and more than $2d$ with nodes A, R . To begin with, let $L_{v_{i,j}}^z$ denote the z^{th} duplication of $L_{v_{i,j}}$, where $z \geq 1$.

For each node, DPCFG will have the following lists: $L_R = NULL$; $L_A = \{R\}$; $L_B = \{R\}$; $L_C = \{\{A\} \cup L_A^1\}$; $L_D = \{\{A\} \cup L_A^2\} \cup \{\{B\} \cup L_B^1\}$; $L_E = \{\{B\} \cup L_B^2\}$; $L_F = \{\{B\} \cup L_B^3\}$.

According to assumptions on Hamming distance for node H , its initial list is: $L_H = \{\{E\} \cup L_E^1\} \cup \{\{F\} \cup L_F^1\}$. Further processing by extending all sub-lists shows that the final list is: $L_H = \{E, B, R\} \cup \{F, B, R\}$. For node I , its initial list is: $L_I = \{\{C\} \cup L_C^1\} \cup \{\{E\} \cup L_E^2\}$. Further processing finds that the element A in L_C^1 has a Hamming distance more than $2d$ with node I . Thus, the sub-list $\{C\} \cup L_C^1$ has to be deleted. It can be seen

that duplication and deletion on such sub-lists are redundant operations. The initial list for node J is duplicated from all the previous lists: $L_J = \{\{C\} \cup L_C^2\} \cup \{\{D\} \cup L_D^1\} \cup \{\{E\} \cup L_E^3\} \cup \{\{F\} \cup L_F^2\}$. According to the assumptions, in each of the sub-lists, there is at least one node that has a Hamming distance more than $2d$ with node J . Thus, all sub-lists have to be deleted after further processing.

In the example, duplications of intermediate lists might require a large amount of space. This results in the failure of DPCFG in discovering weak motifs. Furthermore, it becomes worst when there is at least one node in each extended sub-list such that its Hamming distance with the new node is larger than $2d$. Because in this case, all connections between nodes have to be duplicated, and all of them are deleted eventually, contributing nothing but execution time. However, DPCFG requires such duplications to keep the complete information in the constructed lists. This guarantees that deletions will not affect the construction of lists for other nodes.

Even though a strategy is designed to determine how to copy the necessary sub-lists selectively, it is still time consuming because the sub-lists are already there but have to be duplicated. For instance, L_E has to be duplicated 3 times to construct new lists for nodes H , I and J . In reality, when the size of cliques is small, random nodes will form a large number of lists [131]. Thus, these redundant duplications will lead to a large amount of computational time (and space). The new algorithm TreeMotif-BF will remove such redundancies to improve efficiency in execution time.

3.2 TreeMotif-BF: tree construction in a breadth-first manner

TreeMotif-BF is inspired by graph-based approaches to weak motif discovery. The novelty of TreeMotif-BF is that it is a tree-structured algorithm to find cliques of motif

instances in graphs. As a clique can represent motif instances having an inexact consensus, TreeMotif-BF is capable of detecting weak motifs with very degenerated sites. The algorithm consists of three steps: graph representation, tree construction, and motif refinement, as summarized in Algorithm 4. In what follows, we describe these steps in detail.

Algorithm 4 TreeMotif-BF

input: A set of m sequences $\{s_t\}_{t=1}^m$, l and d .
output: ranked m -cliques
begin:
 select s_1 and s_2 as the reference sequence;
 build reference node pairs with l -mers in s_1 and s_2 , where one reference node has Hamming distance less than or equal to $2d$ from the other;
 for each reference node pair
 build a graph G using l -mers of $s_3, s_4, \dots, s_m$, where any vertex of G has Hamming distance less than or equal to $2d$ from each reference node;
 for each vertex x of G , and $x \in s_3$
 initialize tree T rooted at x ;
 for $t \in [4, m]$
 append each vertex y of G ($y \in s_t$) to the leaf node(s) of T if y has a Hamming distance less than or equal to $2d$ from all the nodes on the path from the root to the leaf(s) of T ;
 prune tree T by removing the leaf(s) with depth less than $t-2$;
 break if T is empty;
 end for
 extract any m -clique from the path of root to a leaf if T is non-empty, merge and record cliques (ranked according to information content);
 end for
 end for
end

3.2.1 Graph representation

In order to build graphs representing motif instances in dataset $s = \{s_i\}_{i=1}^m$, a pair of reference sequences are chosen. Without loss of generality, let us choose s_1 and s_2 . Each pair of nodes $\{(v_{1,j'}, v_{2,j''})\}_{j', j''=1}^{n-l+1}$ in v_1 and v_2 forms a reference node pair if they have

Hamming distance up to $2d$. Thereafter, a graph is constructed by drawing edges between a reference node pair $(v_{1,j'}, v_{2,j''})$ and nodes in $\{v_i\}_{i=3}^m$ (of non-reference sequences). Specifically, an edge is drawn if the maximum Hamming distance of a node to the reference node pair $(v_{1,j'}, v_{2,j''})$ is equal to or less than $2d$ (as if the reference node pair were one node). Algorithm 5 describes the graph construction where P_i gives nodes selected from the sequence s_i according to a reference node pair $(v_{1,j'}, v_{2,j''})$. The selected nodes, as candidate motif instances, will then be used to construct trees.

Algorithm 5 Node selection

input : reference node pair $(v_{1,j'}, v_{2,j''})$; l -mers of sequences $\{s_i\}_{i=3}^m$: $\{\{v_{i,j}\}_{j=1}^{n-l+1}\}_{i=3}^m$.
output: vertices of a graph for the reference node pair: $\{P_i\}_{i=3}^m$.
 $\{P_i \leftarrow \emptyset\}_{i=3}^m$
for $\{i\}_{i=3}^m$ **do**
 for $\{j\}_{j=1}^{n-l+1}$ **do**
 if $D_{\max}(v_{1,j'}, v_{2,j''}, v_{i,j}) \leq 2d$ **then**
 $P_i = P_i \cup \{v_{i,j}\}$
 end if
 end for
end for
return $\{P_i\}_{i=3}^m$

3.2.2 Tree construction

To begin with, let $T = \{\text{root}(T)\} \cup_{k=1}^K T_k$ be a K -ary tree with a finite set of nodes where $\text{root}(T)$ denotes the root of the tree. Except for the root, other nodes are partitioned into k disjoint subsets $\{T_k\}_{k=1}^K$ where each T_k is a K -ary tree. Except for the root, each node in T has one unique parent node and can have up to K child nodes. Let $\text{parent}(X)$ denote the parent node of node X , $\text{child}(X)$ denote the set of child nodes of node X , and $\text{depth}(X)$ denote the depth of node X in the tree. Assume $\text{depth}(\text{root}(T)) = 1$. A path connecting the root node of a tree with a leaf node is referred to as a branch (of the tree). That is, a leaf node indicates a branch of the tree.

Now we introduce how to use the nodes given in $\{P_i\}_{i=3}^m$ to construct trees. First of all, another reference sequence (other than the reference pair for node selection, say s_3) is selected for tree construction. The l -mers of the reference (in P_3) will be initialized as root nodes of trees (to construct). Next, each root node is incrementally built into a tree (if possible) by using nodes in $\{P_i\}_{i=4}^m$. Specifically, we try to extend each branch of an intermediate tree by using new nodes (i.e., l -mers in the remaining sequences). A branch of an intermediate tree is said to be *extendable* by a node if the maximum Hamming distance between any one of the nodes on the branch and the node is equal to or less than $2d$.

Given a new node and an intermediate tree T , the *Extendable()* function looks for the extendable branches of tree T . This is carried out in a breadth-first manner by using a first-in-first-out queue Q , as given in Algorithm 6. The returned elements in Q are the leaf nodes indicating the corresponding branches can be extended.

Algorithm 6 *Extendable*($v_{i,j}, T$): check if tree T can be extended by a node $v_{i,j}$.

input : an intermediate tree T and a node $v_{i,j}$.
output: a set of leaf nodes of T that can be appended by $v_{i,j}$.
initialize $Q \leftarrow \emptyset$
if $D(v_{i,j}, \text{root}(T)) \leq 2d$ **then**
 $Q.\text{back}() \leftarrow \text{root}(T)$ /* insert the root to the end of Q */
 while $Q \neq \emptyset$ and $\text{depth}(Q.\text{front}()) < i - 3$ **do**
 $F \leftarrow Q.\text{front}()$ /* get the front node and delete it from Q */
 for each node $c \in \text{child}(F)$ **do**
 if $D(c, v_{i,j}) \leq 2d$ **then**
 $Q.\text{back}() \leftarrow c$
 end if
 end for
 end while
end if
return Q

Initially, Q is set as empty. For the root node R of T , if $D(v_{i,j}, R) > 2d$, traverse of the tree is stopped and Q is returned as empty; otherwise R is inserted into Q . If traverse

of the tree is continued, the front node in Q is firstly checked. If the node is a leaf node of T , traverse of the tree is stopped and Q is returned; otherwise, the node is removed from Q and its children (whose Hamming distances from $v_{i,j}$ are not larger than $2d$) are inserted into the end of Q . While Q is not empty, the above process is repeated. Note that a tree node $v_{i,j}$ corresponds to a node from s_i starting at position j .

If queue Q is returned as nonempty, all the kept nodes correspond to branches that are extendable by node $v_{i,j}$. Thus we directly append $v_{i,j}$ onto these leaf nodes. If Q is empty, it means that no branches in T can be extendable by $v_{i,j}$.

After checking for $v_{i,j}$, the same process is carried out for all the other nodes in P_i . After appending, there is a further step referred to as elimination of spurious branches. If a branch (the path from root node to a leaf node) cannot be extendable by any node in a specific sequence, a set of nodes can be deleted that are on a path by backtracking from the leaf node to a node whose parent has more than one child.

Specifically, when appending has been completed for all new nodes $v_{i,j}$ from P_i , a leaf node in the current tree should have a depth of $i-2$ if the clique indicated by its parent node (a previous leaf node) had been extendable by any new nodes at least once. Conversely, a deletable branch for sequence s_i has been encountered if a leaf node has a depth less than $i-2$. Therefore, to find out and prune away all deletable branches for sequence s_i in the new tree, we simply traverse all current leaves and check the depth of each leaf node. If it is less than $i-2$, the corresponding branch is pruned away.

If a tree has at least one branch of depth $i-2$, the tree is retained to append new nodes in P_{i+1} (if $i < m$); otherwise, the tree construction process for the current root can be terminated. If there is any other root node initialized, the same construction process is carried out. The tree-construction algorithm is given in Algorithm 7.

Algorithm 7 Tree construction

```

input : a reference node pair  $(v_{1,j'}, v_{2,j''})$  and  $\{P_i\}_{i=3}^{m-1}$ .
output: a set of trees relevant to  $(v_{1,j'}, v_{2,j''})$ .
for each node  $v_{3,r} \in P_3$  do
    initialize a tree  $T_r$  at  $v_{3,r}$ 
    for  $\{i\}_{i=4}^m$  do
        for each node  $v_{i,j} \in P_i$  do
             $Q \leftarrow \text{Extendable}(v_{i,j}, T_r)$ 
            if  $Q$  is not null then
                append  $v_{i,j}$  to  $T_r$  according to  $Q$ 
            end if
        end for
        prune  $T_r$ 
        if  $T_r$  is empty then
            break
        end if
    end for
    Report  $T_r$  (if not empty) with  $(v_{1,j'}, v_{2,j''})$  /* from which cliques are extracted and possibly merged*/
end for

```

3.2.3 Motif refinement

Cliques extracted from a tree can be composed of a limited number of true motif instances and several random l -mer string. This is because these random l -mer strings have Hamming distance less than or equal to $2d$ to the true instances. In this case, if only m -cliques are reported, the recall rate could be low. Therefore, we can merge the cliques to increase the number of true instances in a report. The refinement is performed as follows:

- *Merging redundant tree branches after tree construction:* Once a tree is obtained, each branch together with the related reference node pair form a clique. Such cliques can be merged which aims to guarantee more possible target instances included in the same clique (if the clique is formed by motif instances).

False motifs (or cliques involving with random l -mers) cannot be excluded by only merging of cliques. Suppose there are X cliques formed by using nodes of a tree, and at most x cliques are to be reported as motifs, where $X \geq x$. To get rid of false motifs, each clique is ranked according to information content (Eq. 3.1), and the top x highly ranked cliques are selected as the possible motifs [9, 148]. Then, the algorithm tries to merge the remaining $X - x$ cliques into one of the x cliques. Specifically, if any pair of nodes respectively from the two cliques are within $2d$, the two cliques are merged.

After another tree is constructed, a newly formed clique can have higher information content than the current x cliques to report. Then, the lowest ranked clique is removed from the report list, and the new clique will be added (to report). How merging of cliques influences discovery accuracy will be discussed in the experimental section.

Moreover, consensus(es) of the cliques can be aligned from reported clique(s). The algorithm also implements re-scanning using the consensus(es) which aims to increase true positives, although false positives might also be increased [198].

$$I = \sum_{i=1}^l (2 + \sum_{a \in \Omega} (f(a, i) * \log_2 f(a, i))) \quad (\text{Eq. 3.1})$$

where $f(a, i)$ is the frequency of base a appearing on position i .

3.2.4 TreeMotif-BF: an example

Based on initial graph representation of candidate motif instances, TreeMotif-BF dynamically appends interesting candidates onto a tree and eliminate false candidates. The processes of tree construction and clique refinement in TreeMotif-BF are explained using Figure 3.2 (a)-(d).

Figure 3.2(a) shows an intermediate tree of depth 3 and nodes to append. Suppose the tree is built of nodes in P_3 to P_5 , where $\{R\} \subseteq P_3$, $\{A, B\} \subseteq P_4$, $\{C, D, E, F\} \subseteq P_5$; and new candidate nodes $\{H, I, J\} \subseteq P_6$ are to be appended to the tree. Suppose each

Figure 3.2: TreeMotif-BF: illustration of tree construction and clique refinement

node in sets $\{R, A, C\}$, $\{R, A, D\}$, $\{R, B, D\}$, $\{R, B, E\}$, $\{R, B, F\}$ has a Hamming distances less than or equal to $2d$ with the other nodes in the same set. Assume H has a Hamming distances less than or equal to $2d$ with nodes R, A, B, E, F and more than $2d$ with nodes C, D . Also assume that node I has a Hamming distances less than or equal to $2d$ with nodes R, B, C, E and more than $2d$ with nodes A, D, F . Assume J has a Hamming distances less than or equal to $2d$ with nodes B, C, D, E, F and more than $2d$ with nodes A, R .

Figure 3.2(b) shows the tree construction with nodes H, I and J . The tree is filtered using the function `Extendable()` (Algorithm 6 of Section 3.2.2) and appended according to queue Q . Specifically, Q should keep several leaf nodes with depth 3 (if any) in the tree as the retained elements after filtering. Firstly, node H is checked. According to the assumptions, Q will change as follows: Q is initialized as $Q=\{R\}$ as $D(R, H) \leq 2d$; then remove R from Q (now $Q=\emptyset$) and check its children A, B : insert A into Q as $D(A, H) \leq 2d$ (now $Q=\{A\}$); insert B into Q as $D(B, H) \leq 2d$ (now $Q=\{A, B\}$); then

remove A from Q (now $Q=\{B\}$) and check its children C, D : as $D(C, H) > 2d$ and $D(D, H) > 2d$, none of the children of A is inserted into Q ; then remove B from Q (now $Q=\emptyset$) and check its children E, D, F : insert E into Q as $D(E, H) \leq 2d$ (now $Q=\{E\}$); skip D as $D(D, H) > 2d$ (still $Q=\{E\}$); insert F into Q as $D(F, H) \leq 2d$ (now $Q=\{E, F\}$). The filtering process will terminate as the depth of E is equal to 3, which means leaf nodes of the tree before any appending have been reached. Final elements in Q , i.e. leaf nodes E and F , correspond to branches that are extendable by the new node H , thus append H to these two leaf nodes respectively. A similar process is carried out for node I . Queue Q changes as follows: $\emptyset, \{R\}, \emptyset, \{B\}, \emptyset, \{E\}$ (note that all children of A are skipped as $D(A, I) > 2d$, which eliminates redundant traversing). So node I can only be appended to leaf node E , as shown in Figure 3.2(c). For node J , as $D(R, J) > 2d$, there is no need to traverse the tree. Now as there are no new nodes from P_6 , the filtering and appending process have been completed for s_6 .

Figure 3.2(c) shows the elimination of spurious branches after tree construction with nodes H, I and J . That is, this is an intermediate tree which needs to be checked further if there is any deletable branch for s_6 to be pruned. This related to the first motif refinement strategy (in Section 3.2.3). In detail, leaf nodes C, D (the child of A), D (the child of B) have depth less than 4, therefore three deletable branches (sub-branches) are found in the new tree and needed to be pruned away. The paths of backtracking for pruning have been shown in the dashed ellipses.

Figure 3.2(d) shows the merging of redundant tree branches after tree construction. This is a new tree after processing for all new nodes from P_6 . If $m=6$ (two reference sequences for node selection included), it is a final tree with three branches. In this case, branches of the tree stand for cliques of motif instances. For example, $\{R, B, E, H\}$ and $\{R, B, E, I\}$ are two branches, i.e., they can form two cliques of size 6 (with the reference node pair). Consensus motif is obtained using such cliques, supposing the base

at each position is deterministic. Furthermore, note that the two cliques share not only the reference nodes (for node selection) but also nodes $\{R, B, E\}$ and might refer to the same motif. Therefore, such cliques are merged as one clique if all the nodes are within $2d$ (making the clique size larger than m); otherwise, they are deemed as different cliques. For the preceding example, a clique of $\{R, B, E, H, I\}$ is formed if $D(H, I) \leq 2d$. In this way, more random substrings (that preserve as good as true motif instances) are included in the final cliques. At the same time, this can guarantee more target instances included in the same clique. This relates to the second clique refinement strategy (in Section 3.2.3).

On the other hand, if $m > 6$, it is a tree that can be retained to be further processed. That is, the new nodes from $P_7, P_8, \dots$ will be tested whether they can be appended onto the new tree. If $m > 6$ and suppose none of the leaf nodes in the tree has a child appended from P_6 , there is no need to build the tree anymore for the other nodes from $P_i, i > 6$. As with OOPS data, if there are no motif instances in P_6 appended onto the tree, OOPS condition is not satisfied. This relates to a termination condition of tree construction (see Algorithm 7 in Section 3.2.2).

3.2.5 Time and space complexity

3.2.5.1 Time complexity

Profiling analysis of an implementation of a tree-structured algorithm shows that Hamming distance calculation takes up most of execution time. Thus, the running time of the TreeMotif-BF algorithm is analyzed in terms of the number of Hamming distance calculations. Suppose there are m sequences of length n . Given reference sequences s_1 and s_2 , TreeMotif-BF needs $(n - l + 1)^2$ times of calculations to find out all the reference node pairs.

Suppose p is the probability of two random strings of length l having a Hamming distance less than or equal to $2d$. Then, an intuitive but inexact measure of weakness of

the (l, d) -motifs is given by

$$p = \sum_{i=0}^{2d} \binom{l}{i} (3/4)^i (1/4)^{l-i} \quad (\text{Eq. 3.2})$$

The larger the value p , the weaker the motif tends to become [27, 66, 198]. The expected number of node pairs is $(n - l + 1)^2 p$, while in the worst case, it is $(n - l + 1)^2$. For each reference node pair, the number of l -mer substrings in each remaining sequence s_i ($3 \leq i \leq m$) for tree construction is $(n - l + 1)$ in the worst case.

There are up to $(n - l + 1)$ root nodes in s_3 . Each tree is a $(n - l + 1)$ -nary tree during construction, and it can have a maximum depth of $m - 2$. The number of nodes in an intermediate tree of depth x ($1 \leq x < m - 2$) is bounded by $O((n - l + 1)^{x-1})$. Note that for depth 1 of the tree, we append nodes in s_4 . For depth $x > 1$, we continue appending nodes in s_{x+3} , and the number of Hamming distance calculations required is bounded by $O((n - l + 1)^{x-1} \times (n - l + 1))$, or $O((n - l + 1)^x)$. Thus for each tree, the number of Hamming distance calculations is bounded by $O((n - l + 1)^{m-3})$.

As mentioned, there are up to $(n - l + 1)$ such trees for each reference node pair and there are $(n - l + 1)^2$ reference node pairs. Thus, the total number of Hamming distance calculations is bounded by $O((n - l + 1)^{m-3} \times (n - l + 1) \times (n - l + 1)^2)$, or $O(n^m)$.

Although the theoretical time complexity is exponential, the number of nodes that can be appended onto an intermediate tree decreases with the increased depth of the tree. This property leads to a reliable algorithm in practice. In addition, TreeMotif-BF does not have to duplicate the previous tree for new nodes. When sequences containing motif instances are long enough, TreeMotif-BF can reduce its execution time by avoiding the redundant duplications required by DPCFG. For example, it can be seen in Figure 3.2(b, c, d) that the TreeMotif-BF algorithm does not have to duplicate the previous tree for the new nodes in s_6 . It only needs to append new nodes onto the previous tree after filtration. The pruning operations only happen when all nodes in s_6 have been checked for extendability, i.e., first appending and then deleting will keep all the information derived

from the previous sequences without duplications. By effectively avoiding duplications, TreeMotif-BF can run much faster in practice while producing the same (OOPS) results as DPCFG [198].

3.2.5.2 Space complexity

As mentioned, there are up to $(n - l + 1)$ root nodes in s_3 . For each root we expect to construct a tree of l -mers. The trees are constructed sequentially. For each tree, the number of nodes is $\sum_{x=1}^{m-2} (n - l + 1)^{x-1}$. Thus the space complexity is $O(n^{m-3})$ (in the worst case). TreeMotif-BF can consume a large amount of space when applied to discover extremely weak motifs. This can limit its application in finding very weak motifs.

3.3 TreeMotif-DF: tree construction in a depth-first manner

To reduce the space requirement by TreeMotif-BF, TreeMotif-DF constructs trees (or simply lists) in a depth-first manner. A general description of the algorithm is given in Algorithm 8. In the following sections, we introduce the details of the algorithm.

3.3.1 List construction

TreeMotif-DF builds trees by using the vertices of a graph in $\{P_i\}_{i=3}^m$ constructed according to a reference node pair $(v_{1,j'}, v_{2,j''})$. Specifically, for the selected reference node pair, a triplet is first formed with a vertex in P_3 . Then it checks if there is a vertex in P_4 that has a Hamming distances less than or equal to $2d$ to all the elements of the triplet. If true, a quartet will be formed; otherwise, it checks the next vertex in P_4 (if any) in the same way. For the latter case, if there are no more vertices to be checked in P_4 , another triplet will be formed using the reference node pair and the next vertex in P_3 (if any). Similarly, if there are no more vertices in P_3 to form triplets with the reference node pair,

another reference node pair (if any) will be checked with the same process. After all the reference node pairs have been tested, the list construction ends.

Generally, after an $(i-1)$ -multiplet ($i \in [4, m]$) has been formed by the vertices in $\{s_k\}_{k=1}^{i-1}$, TreeMotif-DF goes deeper to sequence s_i , gets a vertex from P_i and checks if the vertex can form an i -multiplet with the current $(i-1)$ -multiplet. If the check returns false, it gets a next vertex in P_i (if any) and checks if an i -multiplet can be formed; otherwise, it adds the vertex from P_i and goes further deeper to s_{i+1} (if $i+1 \leq m$) and proceeds in the same way as that for s_i . For the former case, if there are no more vertices to be checked in P_i , the algorithm will backtrack to a previous sequence s_k ($k \geq 4$) for which there is still a next vertex in P_k to be checked. For the latter case, if $i=m$ and there is an m -multiplet formed, the m -multiplet represents a clique of motif instances.

Let $\{L[i]\}_{i=1}^m$ be a variable where the intermediate substrings (deemed as the motif instances) are stored. An operation $L[i] \leftarrow v_{i,j_i}$ will overwrite its previous content. Let $CheckVertices(v_{i,j}, L)$ return TRUE if $D(v_{i,j}, L[k]) \leq 2d$ for all $\{k\}_{k=3}^{i-1}$; otherwise it returns FALSE. The list construction procedure is shown in Algorithm 9.

3.3.2 TreeMotif-DF: an example

An example about the list construction is shown in Figure 3.3. Suppose with a reference node pair $(v_{1,j'}, v_{2,j''})$, a graph is constructed and the vertices of the graph are recorded in $\{P_i\}_{i=3}^m$. The assumptions on nodes and their pair-wise Hamming distance are the same as those in Section 3.2.4.

The vertex R in P_3 is inserted into $L[3]$ to form a triplet with the reference node pair $(v_{1,j'}, v_{2,j''})$. Then TreeMotif-DF goes deeper into s_4 and inserts vertex A in P_4 into $L[4]$ as $D(R, A) \leq 2d$ (note that there is no need to check the Hamming distance of vertex A to each vertex forming the reference node pair as we have already made this check; this also holds for the other vertices to be added into L). Similarly, it inserts vertex C into

$L[5]$ as $D(A, C) \leq 2d$ and $D(R, C) \leq 2d$ and goes deeper into s_6 , as shown in Figure 3.3(a). For P_6 , however, as $D(C, H) > 2d$, $D(I, A) > 2d$ and $D(J, A) > 2d$, none of the vertices can be added into the current L to form a larger clique. Therefore, TreeMotif-DF backtracks to s_5 , replaces the current vertex C in $L[4]$ with a next vertex D and goes deeper into s_6 again, as shown in in Figure 3.3(b).

Figure 3.3: TreeMotif-DF: illustration of tree construction

Similar to vertex C , the new clique $\{R, A, D\}$ can also be extended by none of the vertices in P_6 . Therefore, TreeMotif-DF backtracks to s_5 again, as the Hamming distances between A and E or F are larger than $2d$, $L[4]$ will be reset as B in P_4 and goes deeper into s_5 again. C and D will be skipped as their Hamming distances to B are larger than $2d$. $L[5]$ will be set as E according to the assumptions on the Hamming distances and the process goes to s_6 . As all the Hamming distances of vertex H to the vertices in $\{R, B, E\}$ are less than or equal to $2d$, thus vertex H can be inserted into $L[6]$. Then a clique is found as $\{R, B, E, H\}$, as shown in in Figure 3.3(c). After recording the clique, next vertices I, J in P_6 will also be checked sequentially (note that another clique

Algorithm 8 TreeMotif-DF

```

input: A set of  $m$  sequences  $\{s_t\}_{t=1}^m$ ,  $l$  and  $d$ .
output: ranked  $m$ -cliques
begin:
  select  $s_1$  and  $s_2$  as the reference sequence;
  build reference node pairs with any two  $l$ -mers in  $s_1$  and  $s_2$ ;
  for each reference node pair
    build a graph  $G$  using  $l$ -mers of  $s_t$ ,  $t \in [3, m]$ , and  $P_t$  collects the vertices
      (of  $G$ ) from  $s_t$ ;
    for each vertex  $y$  in  $P_3$ 
      initialize List  $L$  with  $y$ ;
      initialize  $i$  as 4 and get the first vertex  $x$  from  $P_i$ ;
      while  $i$  is not 4 or vertex  $x$  is not null /* 'null' indicates empty  $P_i$  */
        if vertex  $x$  has a Hamming distance less than or equal to  $2d$  from all
          existing elements in  $L$ 
          replace the element  $z$  in  $L$  with  $x$ , where  $x, z \in P_i$ ;
          if  $i$  is equal to  $m$ 
            record  $l$ -mers in  $L$  (ranked with information content);
            get the next vertex  $x$  from  $P_i$ ;
          else /* continuation */
            increase  $i$  to  $i + 1$ ;
            get the first vertex  $x$  from  $P_i$ ;
          end if
        else
          get one next vertex  $x$  from  $P_i$ ;
        end if
        while  $i$  is larger than 4 and  $x$  is null /* backtracking */
          decrease  $i$  to  $i - 1$ ;
          get one next vertex  $x$  from  $P_i$ ;
        end while
      end while
    end for
  end for
end

```

is found as $\{R, B, E, I\}$ by extending $\{R, B, E\}$, as shown in in Figure 3.3(d)). After all vertices in P_6 is finished, TreeMotif-DF backtracks to s_5 again, replaces vertex E with a next vertex F and checks the construction in the same way. When the check for vertex F is finished, TreeMotif-DF Backtracks to s_4 as F is the last vertex in P_5 . Meanwhile, as vertex B is also the last vertex to check in P_4 , TreeMotif-DF further backtracks to s_3 . As there is no more vertex in P_3 to form a triplet with $(v_{1,j'}, v_{2,j''})$, TreeMotif-DF terminates the list construction process for the current reference node pair. If there is any more reference node pair, TreeMotif-DF carries out a similar process for it.

3.3.3 Time and space complexity

3.3.3.1 Preliminary analysis

Suppose there are two vertices $v_{i,1}, v_{i,2}$ located in different positions of P_i . One of them can extend an $(i-1)$ -multiplet to a $\{v_{i,1}\} \cup (i-1)$ -multiplet, and the other can extend the same clique to a $\{v_{i,2}\} \cup (i-1)$ -multiplet. Moreover, suppose there is another vertex $v_{i+1,1}$ in P_{i+1} that can extend the two i -multiplets. Then, we can see that when checking if $v_{i+1,1}$ can be added into $\{v_{i,1}\} \cup (i-1)$ -multiplet or $\{v_{i,2}\} \cup (i-1)$ -multiplet, the part of checking with vertices in the $(i-1)$ -multiplet is common. That is, there is time wasted in calculating the same Hamming distances.

To avoid the redundancy mentioned in the preceding paragraph, we can pre-compute a lookup table T of Hamming distances among vertices. For sequence i ($i \in [2, m]$), let Q_i denote the set of vertices of s_i that have a Hamming distance less than or equal to $2d$ to a given reference node (say from s_1), and $|Q_i|$ denote the number of vertices in Q_i . With the vertices in Q , the element $T(v_{i,j}, v_{i',j'})$ is inserted with the Hamming distance between vertex $v_{i,j}$ and vertex $v_{i',j'}$, where $\{i\}_{i=2}^{N_p-1}, \{j\}_{j=1}^{|Q_i|}, \{i'\}_{i'=i+1}^{N_p}, \{j'\}_{j'=1}^{|Q_{i'}|}$ and $N_p = \min\{m, 20\}$. It is not necessary to set $N_p = m$ when m is large. This will be supported by the experiments for testing algorithmic scalability on the number of sequences.

Algorithm 9 TreeMotif-DF: list construction

input : a reference node pair $(v_{1,j'}, v_{2,j''})$ and candidate instances $\{P_i\}_{i=3}^m$.
output: list L containing motif instances relevant to $(v_{1,j'}, v_{2,j''})$.
 initialize $L[1] \leftarrow v_{1,j'}$
 initialize $L[2] \leftarrow v_{2,j''}$
/ start depth-first traversal from s_3 */*
for each $v_{3,j'''} \in P_3$ **do**
 $L[3] \leftarrow v_{3,j'''}$
 $i \leftarrow 4$
 $v_{i,j} \leftarrow$ first vertex in P_i
 / continue and backtrack */*
 while $(i \neq 4 \text{ or } v_{i,j} \neq \text{NULL})$ **do**
 if $(\text{CheckVertices}(v_{i,j}, L) \text{ is TRUE})$ **then**
 $L[i] \leftarrow v_{i,j}$
 if $(i = m)$ **then**
 return L
 $v_{i,j} \leftarrow$ next vertex in P_i */* if no next, $v_{i,j}$ is set as NULL */*
 else
 $i \leftarrow i + 1$ */* continue */*
 $v_{i,j} \leftarrow$ first vertex in P_i
 end if
 else
 $v_{i,j} \leftarrow$ next vertex in P_i */* if no next, $v_{i,j}$ is set as NULL */*
 end if
 while $(i > 4 \text{ and } v_{i,j} = \text{NULL})$ **do**
 $i \leftarrow i - 1$ */* backtrack */*
 $v_{i,j} \leftarrow$ next vertex in P_i */* if no next, $v_{i,j}$ is set as NULL */*
 end while
 end while
end for
/ Function $\text{CheckVertices}(v_{i,j}, L)$ return TRUE if $D(v_{i,j}, L[k]) \leq 2d$ for all $\{k\}_{k=3}^{i-1}$ */*

3.3.3.2 Complexity

First of all, to find $\{Q_i\}_{i=1}^m$ by using one vertex in s_1 , it requires $(m-1)(n-l+1)$ Hamming distance calculations. Therefore in total, it approximately requires $(m-1)n^2$ Hamming distance calculations.

For graph construction, the Hamming distance calculation can refer to the Hamming distance table $\{\{T(v_{i,j}, v_{i',j'})\}_{i=1}^{N_p-1}\}_{i'=i+1}^{N_p}$. To prepare the table, it requires $C_2^{N_p-1} C_1^{(n-l+1)}$ Hamming distance calculations (where $C_x^y = y!/(x!(y-x)!)$ and x, y are integers and $y \geq x$). By setting $N_p=m$, an upper bound of the number of Hamming distance calculations can be obtained as: $C_2^{m-1} C_1^{(n-l+1)} C_1^{(n-l+1)} < m^2 n^2$. As there are $n-l+1$ such tables to prepare, the time complexity of TreeMotif-DF is bounded by $O(m^2 n^3)$, better than TreeMotif-BF theoretically. However, if taking the depth-first search of cliques into account, as there could be $(n-l+1)^m$ cliques in a dataset of m length- n sequences in the worst case, the algorithm has an exponential time complexity of $O(n^m)$ in the worst case.

The main space requirement of TreeMotif-DF is in recording substrings, L and the Hamming distance table, which is $O((mn)^2)$ in the worst case. It is much better than that of TreeMotif-BF. Table 3.1 summaries complexities of several motif discovery algorithms (in the worst case).

Table 3.1: Comparison: time and space complexities

Algorithms	Time Complexity	Space Complexity
DPCFG	$O(n^m)$	$O(n^{m-2})$
PMSprune	$O(mn^2 N(l, d))$	$O(mn^2)$
iTriplet	$O(mn^3 l^3 d^2)$	$O(N(l, d))$
TreeMotif-BF	$O(n^m)$	$O(n^{m-3})$
TreeMotif-DF	$O(n^m)$	$O((mn)^2)$

Note* $N(l, d) = \sum_{i=0}^d C_i^l 3^i$.

3.4 Experimental results

In this section, TreeMotif algorithms are evaluated and compared with the other existing algorithms on synthetic datasets and biological datasets. For synthetic comparison, we test the performance of TreeMotif algorithms and other known algorithms by tuning parameters m , n , l , d and biasing base distributions of background sequences. All the algorithms are tested on the same workstation with 2.66 GHz CPU and 3.00 GB RAM. The memory consumption of several algorithms when handling a few (l, d) -motif problems is given in Table 3.2. In the following comparisons, if an algorithm does not encounter memory shortage during running, we deem its memory requirement as satisfactory. All the statistics are averaged over 10 datasets (which also applies for the experiments on synthetic datasets of Chapters 4 and 5 if not specified).

Table 3.2: Comparison: memory consumption

m, n	20, 600	20, 2000	20, 600	20, 600	100, 600
l, d	15, 4	15, 4	16, 5	18, 6	14, 4
iTriplet	14	17	20	44	17
PMSprune	2	2	2	2	2
DGCFG	30	296	m.s.	m.s.	>1000
TreeMotif-DF	9	25	20	32	17
TreeMotif-BF	8	10	10	m.s.	130

Unit: Megabyte; m.s.: memory shortage

3.4.1 Synthetic data

Synthetic datasets are generated by using the model of fixed number of mutations [131]. First, m i.i.d. sequences are generated. Second, an instance of a target motif of length l is planted at a random site of each of k sequences. Third, for each motif instance, up to d randomly selected sites are mutated. Notes: (1) the base distribution in motif instances also follows i.i.d.; (2) when parameter $k=m$, a dataset corresponds to an exact one (OOPS) while when parameter $k < m$, it corresponds to a noisy one (ZOOPS). In this chapter, all the datasets are generated as exact ones.

Let TP denote the number of predicted motif positions that are known to be true motif positions; FP denote the number of predicted motif positions that are known to be non-motif positions; FN denote the number of predicted non-motif positions that are known to be true motif positions; TN denote the number of predicted non-motif positions that are known to be true non-motif positions. The nucleotide level statistics, including $nSn = TP/(TP+FN)$, $nPPV = TP/(TP+FP)$; $nSP = TN/(TN+FP)$; $nPC = TP/(TP+FN+FP)$; $nASP = (nSn+nPPV)/2$; $nCC = (TP*TN - FP*FN)/[(TP+FP)(FP + TN)(TN+FN)(FN+TP)]^{0.5}$, are used to compare different algorithms. For more details about the statistics, please refer to the original introduction [174].

Moreover, besides p as given by Eq. 3.2, another measure E [27] could be used to indicate the weakness of a motif, which is given by Eq. 3.3. Value E is the logarithmic-scaled number of (l, d) -motifs that randomly appear in a dataset of m length- n sequences. As E -value applies an operator $-\log_{10}()$ on the number of random motifs, a smaller E -value (including negative numbers) suggests greater difficulty to discover a motif (theoretically).

$$E(l, d, m, n) = -\log_{10}(4^l(1 - (1 - \sum_{i=0}^d C_i^l (3/4)^i (1/4)^{l-i})^{n-l+1})^m) \quad (\text{Eq. 3.3})$$

3.4.1.1 Motif challenge problem

It has been shown by Pevzner *et al.* that MCP (where $l=15$, $d=4$, $m=20$, and $n=600$) is beyond reach to many existing probabilistic algorithms [131], thus we use this basic (l, d) -motif problem as a natural test case for our algorithm.

Algorithms are first compared on the Motif Challenge Problem. For this problem, each dataset is generated with $m=20$, $n=600$, $l=15$ and $d=4$ (and thus $p=0.054$ and $E=15$). Although E indicates that there is not any random motif in such datasets, it will be shown that probabilistic algorithms cannot perform as satisfactorily as deterministic algorithms.

All algorithms including MEME, BioProspector, MDscan, PatternBranching, GibbsMotifSampler, PROJECTION, DPCFG, PMSprune, iTriplet and TreeMotif algorithms, are allowed to report three groups of motif instances they ranked as the best, and the one with the highest performance coefficient (nPC) is selected for comparison. Note that TreeMotif algorithms rank candidate motifs according to the information content of cliques as given in Section 3.2.3.

Table 3.3 shows the comparison. GWM2, iTriplet and PMSprune can achieve nSn of 1.00 (with standard error as 0.00) which means that they find out all the target motif instances. Meanwhile, these algorithms give nPC of less than 1.0. This is because there are random l -mer substrings of sequences that are as (more) conserved to the consensus motif as (than) the target instances. Random strings can be included in the reported cliques due to the re-scanning strategy by using consensus motifs in GWM2, the alignment strategy of candidate motifs in PMSprune and the sequence-instance association strategy in iTriplet.

TreeMotif-DF and DPCFG produce nPC and nSn less than 1.00. They are set to produce m -cliques among which the target one (of motif instances) must be included. Cliques highly ranked according to information content are reported as candidate motifs. However, there is a possibility that a random string is more conserved than a target motif instance. As a result, the information content of a ‘quasi-random’ clique (where a subset of the elements are random l -mers) might be larger than the target one (composed of all the true motif instances), thus highly ranked and reported. As m is the number of predicted positives, if FP is x , then TP is $m-x$, and FN is x . Therefore, $nSn=(m-x)/m$ and $nPC=(m-x)/(m+x)$. If $x>0$, nPC and nSn must be less than 1. This means that although the target clique must have been formed, it is possible that it is not reported. (Note that DPCFG does not have a ranking strategy; but as it finds the same cliques as TreeMotif-DF, we consider it delivers the same accuracy. This is not repeated in the following sections.)

TreeMotif-BF shows different nSn and nPC from the above algorithms. This is because TreeMotif-BF uses a strategy for refining motifs by merging cliques. The merging process leads to cliques of size larger than m , but smaller than or equal to that of GMW2, iTriplet and PMSprune. That is, more target instances can be included in the same clique. This is why TreeMotif-BF shows a little higher nSn than TreeMotif-DF and DPCFG. Meanwhile, as random strings can be included in the same clique, nPC of TreeMotif is decreased a little.

Table 3.3: Comparison: motif challenge problem

Algorithm	nPC	nSn	Execution Time
GibbsMotifSampler	0.01±0.01	0.02±0.02	3.5±0.4
MDscan	0.06±0.05	0.13±0.11	1.3±0.0
MEME	0.25±0.21	0.36±0.26	3.0±0.1
BioProspector	0.23±0.14	0.36±0.20	3.4±0.5
PROJECTION	0.34±0.26	0.45±0.27	75.4±3.1
PatternBranching	0.94±0.10	0.97±0.06	4.2±0.0
GWM2	0.93±0.07	1.00±0.00	3.9±0.1
PMSprune	0.93±0.07	1.00±0.00	5.2±0.1
iTriplet	0.93±0.07	1.00±0.00	186.0±5.3
DPCFG	0.96±0.05	0.98±0.03	0.7±0.0
TreeMotif-DF	0.96±0.05	0.98±0.03	3.4±0.0
TreeMotif-BF	0.93±0.07	1.00±0.02	0.5±0.0

Note* time unit: second.

Overall, the string-based algorithms could be more robust for finding weak motifs than the profile-based algorithms, because the former can enumerate and validate all the possible motifs. Although PatternBranching shows comparable performance to the latter, Section 3.4.1.2 will show that it cannot perform consistently when the length of background sequences is increased. From the perspective of execution time, TreeMotif-DF requires more time than TreeMotif-BF and DPCFG on the MCP while those of the later two are comparable.

3.4.1.2 Impacts of increasing length of sequences

Although genome-wide ChIP-seq analysis has been increasingly used to characterize transcription factor binding, such methods usually generate peak regions of hundreds of base pairs where motifs are included [90, 201]. Therefore, additional motif finding tools are required to further extract binding motifs from such data. However, one limitation of such tools is that they are not scalable for increasing the ChIP-seq peak regions where the motifs are hidden [90]. This is because the longer the sequences are, the more false instances (that are similar to the true instances) are present in the dataset, and thus the more difficult it is to discover the true instances, especially for weak motifs.

In this section, we test if TreeMotif algorithms can discover motifs with increased length of background sequences. Moreover, their computational efficiency will also be compared to the other existing algorithms.

Datasets are generated with the following settings [27, 166]. The length of sequences n is increased from 700 to 2000 with a step of 100. Each dataset contains twenty sequences with instances of a (15, 4)-motif. Table 3.4 shows several decreased E values with increased sequence length.

Table 3.4: Comparison: weakness indicated by value E

n	700	1000	1300	1600	2000
E	13	10	8	7	5

As seen from Figure 3.4 (note: the corresponding nSn is shown in Figure 3.5; MCP also included), probabilistic algorithms, including GibbsMotifSampler, Bioproscpector, MDscan, and MEME, cannot guarantee satisfactory discovery accuracy. PROJECTION can produce slightly better results than these algorithms but its nPC is still below 0.5. PatternBranching shows higher discovery accuracy with the length of sequences increased to 1300. However, its nPC (as well as nSn) is inconsistent and drops off quickly when sequences become even longer.

Figure 3.4: Comparison: effect of increasing n on nPC

TreeMotif-DF and DPCFG are shown by the same line. They deliver the best nPC when the length of sequences is less than 1500. However, nPC decreases fast when sequences become long (or the weakness of the motif is increased). This is because many random l -mer substrings that are as conserved to the consensus motif as the target instances present in the dataset. These random l -mers lead to random or quasi-random cliques which are highly ranked and reported as motifs. Meanwhile, Figure 3.5 shows that nSn of TreeMotif-DF and DPCFG is less than 1.0; iTriplet, PMSprune and GWM2 guarantee the discovery of all the target instances, i.e., nSn=1.0. The reasons for these performances are similar to discussions in Section 3.4.1.1. These algorithms perform almost consistently with the increase of the length of sequences. For example, they can still produce nPC above 0.8 for $n=2000$.

Overall, accuracy of all the algorithms decrease but due to different reasons. Probabilistic algorithms are likely to miss true motif instances; TreeMotif-DF and DPCFG can find all true motif instances but cannot guarantee to report them thoroughly (due to the selection of exactly m instances); iTriplet, PMSprune, and GWM2 introduce false positive instances (by reporting more than m instances). TreeMotif-DF and DPCFG can deliver the same nSn as iTriplet, PMSprune, and GWM2 as long as they are allowed to report cliques of size larger than m by using a similar re-scanning strategy to GWM2. Instead of re-scanning the dataset with a consensus motif, TreeMotif-BF merges cliques if all instances of these cliques show Hamming distance up to $2d$ from each other. This results in that more target instances are collected in the same clique than m -cliques, and possibly fewer false instances than cliques refined by re-scanning. Therefore, it can be viewed as an eclectic algorithm of those with and without re-scanning.

The execution time is also compared among TreeMotif algorithms, PMSprune and DPCFG, as given in Figure 3.6. Notes: (1) the execution time of PatternBranching and a part of the execution time of iTriplet are also included; (2) for each point, the time is

Figure 3.5: Comparison: effect of increasing n on nSn

an average of 10 datasets; (3) the standard errors are extremely small compared to the corresponding means and thus not shown.

Figure 3.6 shows that TreeMotif-BF requires less execution time than TreeMotif-DF when the length of sequences is increased (or when a motif becomes weaker). This indicates that the strategy on reducing space requirement in TreeMotif-DF leads to an increase of execution time. When the sequence length is below 1700, TreeMotif-BF required the lowest execution time among all exact algorithms. TreeMotif algorithms consume about one third to a half the execution time of DPCFG when sequences are longer than 1800 bp. As more random false instances present in long sequences, DPCFG requires a large number of duplication or redundant operations for list construction, which incurs more execution time. TreeMotif algorithms do not have such overheads and thus they executes much faster.

By increasing n (with $m=20$, $l=15$, and $d=4$), PMSprune exhibits approximate linear increase in execution time. When sequence length is larger than 1700, PMSprune shows the lowest execution time. This indicates that PMSprune has better scalability on n than all the other algorithms. PMSprune performs better than the other algorithms under these settings because the value of l and d are relatively small. That is, the neighborhood $N(l, d)$ it examines is small. When $N(l, d)$ increases, its execution time will increase rapidly. This will be further shown in Section 3.4.1.4.

3.4.1.3 Impacts of increasing number of sequences

With fixed l , d and n , increasing the number of sequences in datasets will provide increased number of motif instances, which theoretically leads to a more exact consensus motif (as seen from the increase of value E). However, in this case, more sequences also lead to more random l -mers which are similar to target instances. Thus, the computational performance of the algorithms can also be affected. The scalability of the

Figure 3.6: Comparison: effect of increasing n on execution time

algorithms must also concern how their performance changes with the increase of the number of sequences.

In this section, experiments are designed to examine how the performance of TreeMotif algorithms varies with regard to the number of sequences in the datasets.

Datasets are generated with m as 20, 25, 30, 35, 40, 50, 100, with $n=600$, $l=14$, and $d=4$ [76]. It is reasonable to set the length of sequences as 600 bp as TFBSs motifs usually appear in such upstream regions of transcription start sites [30]. The results of all relevant algorithms are shown in Table 3.5, where TreeMotif-DF* corresponds to results of TreeMotif-DF without using lookup tables of Hamming distances. The corresponding E values show that the motif becomes stronger.

Table 3.5: Comparison: effects of increasing m on execution time

Algorithms	m (with E below)						
	20	25	30	35	40	50	100
	6	10	14	17	21	29	66
DPCFG	0.7±0.1	0.6±0.1	0.6±0.0	0.7±0.1	0.7±0.1	1.1±0.2	1.2±0.4
GWM2	4.5±0.3	4.3±0.3	4.1±0.3	4.4±0.3	4.6±0.3	4.5±0.3	6.3±2.2
iTriplet	10.6±0.4	10.9±0.3	10.7±0.4	10.7±0.8	10.6±0.3	9.9±0.4	9.9±0.3
PMSprune	0.2±0.0	0.2±0.0	0.4±0.0	0.4±0.0	0.5±0.0	0.7±0.0	1.4±0.0
TreeMotif-BF	0.3±0.0	0.3±0.0	0.3±0.0	0.3±0.0	0.3±0.0	0.3±0.2	4.4±4.8
TreeMotif-DF	0.3±0.0	0.3±0.0	0.3±0.0	0.3±0.0	0.3±0.0	0.4±0.0	1.0±0.6
TreeMotif-DF*	0.6±0.1	0.5±0.0	0.6±0.0	0.5±0.0	0.5±0.0	0.6±0.1	1.1±0.6

Note* time unit: $\times 10^2$ second.

Table 3.5 shows that when the number of sequences is not large (say $m \leq 40$), the execution time of TreeMotif-DF does not increase with pre-computed Hamming distance tables (note that it does not decrease neither although the weakness of the motif is decreased). This is in accordance with the previous analysis that, as more sequences are involved, less random lists are left and thus less execution time is required on Hamming distance calculations. In addition, TreeMotif-DF shows improved execution time compared with TreeMotif-DF*. Taking DPCFG as a basis, it can be derived that less than

a half the reduction on execution time by TreeMotif-DF is contributed by the algorithm itself while the remaining is contributed by Hamming distance lookup tables.

TreeMotif-BF consumes slightly less execution time than TreeMotif-DF. This further indicates that tree construction in the breadth-first manner is more efficient than that in the depth-first manner, although the latter reduces the space requirement. DPCFG, GWM2 and iTriplet are not affected by increasing the number of the sequences. However, it can be seen that TreeMotif algorithms require less time than these algorithms.

When $m \leq 40$, TreeMotif-BF almost requires the lowest execution time among all algorithms. However, when the number of sequences is larger (say $m \geq 50$), sample-driven algorithms begin to perform inconsistently which can be seen from the larger deviation on execution time. This is because, the more sequences included in a dataset, the more l -mer substrings are to be handled. In other words, the possibility becomes high for random cliques to be checked if they can be extended.

For example, supposing there are x substrings of a sequence that show up to $2d$ distance from the same number of substrings of each of the other sequences, the maximum number of cliques formed can be x^m . However, the practical number of cliques are dataset-dependent. That is, some datasets contain a large number of random cliques while others contain few. As a result, the performance of the sample-driven algorithms is also dataset-dependent, showing large standard deviations. For example, DPCFG has difficulty in dealing with 6 (of 10) datasets with $m=100$ (encountering errors on memory allocation); TreeMotif-BF involves a large amount of overhead to traverse the intermediate tree for appending and deleting nodes, which makes the performance of TreeMotif-BF incomparable to that of TreeMotif-DF.

Meanwhile, pattern-driven algorithms, such as PMSprune, check Hamming distances of a candidate motif to a maximum number of $n \times m$ l -mer substrings in a dataset. The number of check increases linearly with m . This is the reason why pattern-driven

algorithms show better consistency (or lower standard deviation) than sample-driven ones when increasing m .

Overall, it can be seen from the table that TreeMotif-DF is slightly better (showing lower average execution time) than PMSprune and other algorithms even for $m=100$. In practice, if a dataset consists of a large number of sequences and it is sure that each sequence contains at least one motif instance, the impact of the number of sequences can be decreased by dividing the dataset into several sub-datasets. Each sub-dataset contains a small number of sequences [166]. The size of the sub-datasets should be determined carefully. Because if the size is too small (then E would be small), many random cliques might be produced and difficult to be eliminated.

3.4.1.4 Impacts of increasing l and d with fixed value p

One of the most critical parameters for motif discovery algorithms is the expected length of motifs. Hu *et al.* showed that the performance of most existing probabilistic algorithms is acceptable for limited length of motifs ranging between 10 and 25 [68]. When increasing the length l (and d), the performance of combinatorial algorithms will also be influenced. Theoretically, for pattern-driven algorithms, increasing l and d will result in exponentially increased number of candidate motifs even if p is fixed; for sample-driven algorithms, increasing l and d will result in exponentially increased number of combinations of candidate instances if p is increased. All these indicate increased requirements on computational time and space.

In this section, experiments are designed to check how the performance of the algorithms is affected by increasing l and d but with fixed p . Section 3.4.1.5 checks how the performance of the algorithms is affected by increasing l and d with increased p .

This section checks whether an algorithm is scalable for finding motifs of different lengths. Experiments are carried out with (l, d) increased from (12, 3) to (50, 16) with

$m=20$ and $n=600$, while the value p is fixed at around 0.05 (and the corresponding value E is also given). The value p is approximately the same as that of a $(15, 4)$ -motif. The value E is extremely large for $l>30$, as given in Table 3.6, which means that there are few random motifs. However, we would like to keep comparison on such motifs, because the value E can be decreased if we increase n and/or decrease m . If an algorithm can detect such motifs in short background sequences (here, $n=600$), there is a possibility for it to detect in long sequences. The corresponding experimental results are collected in Table 3.6.

The performance of PMSprune indicates that it is very sensitive to the combination of the length of motif l and the number of mutation d , i.e., the neighborhood $N(l, d)$ of an l -mer string. Specifically, when l and d are increased, the execution time of PMSprune increase exponentially. The performance of the other pattern-driven algorithm iTriplet also shows the similar trend to that of PMSprune. In addition, iTriplet can encounter errors on memory allocation when handling long motifs such as $(44, 14)$. They are not useful in detecting long motifs (even when the motif is not weak).

DPCFG delivers consistent execution time, which is little affected by increasing the motif length l and number of mutation d as long as p is fixed. For several cases, it requires slightly lower execution time than TreeMotif algorithms.

TreeMotif algorithms also exhibit consistent execution time, which are significantly less than those of PMSprune and iTriplet and comparable to DPCFG. TreeMotif-DF requires more time than TreeMotif-BF, which indicates tree construction in the depth-first manner is inefficient. In addition, because we empirically set N_p as 20, additional execution time could be resulted from unnecessary calculations of Hamming distance.

In summary, execution time of pattern-driven algorithms grows fast due to the exponential increase of the size of the neighborhood $N(l, d)$ of an l -mer string, although p is fixed. Specifically, PMSprune has to verify many more candidate motifs, requiring more

Table 3.6: Comparison: effects of varying (l, d) with fixed p on execution time

$(l, d), p, E$	DPCFG	PMSprune	iTriplet	TreeMotif-BF	TreeMotif-DF
(12, 3), 0.05, 6	0.8	1.6	173.6	0.6	3.0
(18, 5), 0.06, 23	0.6	16.9	230.4	0.5	4.2
(21, 6), 0.06, 32	0.5	46.5	250.0	0.4	4.5
(24, 7), 0.06, 41	0.5	80.2	291.5	0.4	3.9
(27, 8), 0.05, 49	0.4	137.1	354.4	0.4	4.3
(30, 9), 0.05, 58	0.4	242.9	553.9	0.4	4.3
(33,10), 0.05, 66	0.4	405.2	1050.2	0.4	4.4
(36,11), 0.05, 75	0.3	651.8	1418.9	0.4	3.6
(39,12), 0.04, 83	0.3	1056.0	2779.2	0.4	3.8
(42,13), 0.04, 92	0.3	1841.8	2895.5	0.4	3.6
(44,14), 0.06, 92	0.7	19213.5	-e	0.6	7.1
(47,15), 0.06, 100	0.6	28896.2	-e	0.7	7.0
(50,16), 0.06, 109	0.5	-o	-e	0.8	6.5

Note* '-o': more than 10 hours; '-e': error on memory; time unit: second.

execution time; iTriplet has to maintain many more candidate motifs in the hash table, requiring more execution time and space. Meanwhile, sample-driven algorithms such as TreeMotif algorithms can perform more consistently than pattern-driven algorithms under the same settings, because the expected number of l -mer substrings of each sequence for them to handle is a constant of $n \times p$ if p and n are fixed.

3.4.1.5 Impacts of increasing weakness p

In this section, we check if an algorithm is scalable with increased weakness of motifs. The value of p is increased from 0.03 to 0.29 by varying l and d as listed in Table 3.7 (where $m=20$ and $n=600$; the corresponding value E for each motif is also given). Experimental results given by different algorithms are shown in Figure 3.7.

Table 3.7: Comparison: (l, d) -motifs for increasing p

ID	p, E	(l, d)	ID	p, E	(l, d)	ID	p, E	(l, d)	ID	p, E	(l, d)
1	0.03, 58	(28, 8)	8	0.11, 66	(37,12)	15	0.16, 93	(50,17)	22	0.21, 33	(27, 9)
2	0.09, 49	(29, 9)	9	0.12, 58	(34,11)	16	0.17, 59	(36,12)	23	0.22, 59	(38,13)
3	0.10, 32	(23, 7)	10	0.13, 50	(31,10)	17	0.18, 15	(19, 6)	24	0.23, 24	(24, 8)
4	0.10, 23	(20, 6)	11	0.14, 41	(28, 9)	18	0.18, 50	(33,11)	25	0.24, 51	(35,12)
5	0.10, 75	(40,13)	12	0.15, 32	(25, 8)	19	0.19, 7	(16, 5)	26	0.26, 95	(54,19)
6	0.11, 15	(17, 5)	13	0.16, 67	(39,13)	20	0.20, 42	(30,10)	27	0.28, 7	(18, 6)
7	0.11, 6	(14, 4)	14	0.16, 24	(22, 7)	21	0.21, 68	(41,14)	28	0.29, 60	(40,14)

Firstly, we examine how l and d (with p larger than 0.05) influence the performance of the algorithms. iTriplet requires much more execution time than the other algorithms, and it could also be limited in applications as it consumes a large amount of space.

For PMSprune, if the difference in the values of p of two motifs is small, the execution time lies on l and d . However, if the difference in the values of p is large, the execution time is mainly influenced by p . For example, for a $(14, 4)$ -motif and a $(37, 12)$ -motif with $p=0.112$, PMSprune consumes more than 10 hours on the latter compared to 20 seconds on the former. This is the same as discussions in the last section, i.e., PMSprune is sensitive to l and d . For a $(24, 8)$ -motif and a $(25, 8)$ -motif with values of p showing a large difference, PMSprune requires about 4.7 hours on the weaker one compared to 0.9 hours on the other. In addition, PMSprune generally requires over 10 hours for handling motifs with $l \geq 30$ and $p > 0.1$, showing limited scalability.

For TreeMotif algorithms, different from PMSprune, the longer the motif the less time is required if the values of p for two motifs are large but the difference between them is small. Taking a $(22, 7)$ -motif and a $(50, 17)$ -motif as examples, TreeMotif-BF requires 121 seconds for the former and 25 seconds for the latter.

This feature on the execution time of TreeMotif algorithms is due to that they are sample-driven, i.e., their performance depends on how many l -mer strings will be handled. If the length of sequences is n and the length of a motif is l , then the number of l -mer

substrings of each sequence is $n-l+1$. Therefore, for a random l -mer string, it is expected that there are $(n-l+1)p$ substrings of each sequence that show Hamming distance up to $2d$ to the random string. With approximately the same n and p , the larger the length l , the smaller the value $n-l+1$, i.e., the fewer l -mer strings to handle, which indicates that the execution time can be reduced.

Moreover, two l -mer strings have a lower chance to be $2d$ -neighbors if l is large. Thus, the $2d$ -neighbors of such a string have a lower chance to present in the same dataset of size $n \times m$. As a result, there are less random cliques of l -mer strings, which indicates less calculation on Hamming distances. This character also contributes to the lower execution time required by TreeMotif algorithms on discovering long motifs.

Generally, TreeMotif algorithms are more efficient for handling long weak motifs than other algorithms. For a (24, 8)-motif, TreeMotif algorithms requires up to 3 hours while PMSprune requires up to 5 hours. Note that DPCFG encounters errors on memory allocation and iTriplet requires more than 10 hours to produce the results. For a (40, 14)-motif, TreeMotif algorithms can produce results in up to 8 hours while all the other algorithms failed because of error on memory allocation or failing to produce results within a limit of 10 hours on the executive time.

Now we examine how the value p influences the performance of algorithms. As p increases, the execution time of TreeMotif algorithms and DPCFG increases with relatively smooth trends. Meanwhile, the execution time of PMSprune and iTriplet becomes extremely long, sensitive to the combination of l and d .

When p is below 0.167 (corresponding to column 16 in Figure 3.7), DPCFG requires comparable execution time to TreeMotif algorithms. However, when $p > 0.167$, DPCFG begins to perform significantly slower than TreeMotif algorithms. In addition, when $p > 0.23$, DPCFG is likely to fail to produce motifs due to shortage of memory. TreeMotif algorithms perform well on most cases but fail to find (18, 6)-motifs in which PMSprune shows superiority. TreeMotif-DF is slower than TreeMotif-BF.

Figure 3.7: Comparison: effect of increasing p on execution time

All the above comparisons show that pattern-driven algorithms have better scalability to handle extremely weak motifs such as a (18, 6)-motif with $p=0.28$ and $E=7$, while sample-driven algorithms like TreeMotif algorithms are more scalable to handle long weak motifs. TreeMotif algorithms cannot solve problems listed in Table 3.8. There are too many false cliques with such settings. As a result, TreeMotif-DF consumes a large amount of execution time to check the extendability of cliques. The reason why TreeMotif-BF fails is the same as DPCFG. They consume an unreasonable amount of space to maintain cliques. These problems indicate that the capability of TreeMotif algorithms still needs to be improved for handling short weak motifs.

Table 3.8: Comparison: cases that TreeMotif algorithms cannot handle (for $n=600$ and $m=20$)

(l, d)	(9, 2)	(11, 3)	(13, 4)	(15, 5)	(17, 6)	(18, 6)	(19, 7)
p	0.05	0.12	0.21	0.31	0.43	0.28	0.54
E	-0.2	-0.7	-0.7	-0.5	0.1	7	0.8

3.4.1.6 Impacts of tuning background base distribution

The tests presented in the preceding sections assume that datasets and motif instances follow i.i.d., i.e., bases (A , C , G , and T) appear with an approximately equal frequency. However, real sequences of different biological sources have different base compositions, as shown in Table 3.9 [27, 179]. In *Plasmodium falciparum*, for instance, GC-content can be as low as 20 percent (thus AT-rich), although such cases are very rare [133].

As the background base distribution is biased, sequences become more similar to each other than in i.i.d.-cases [27]. In other words, l -mer substrings of sequences are likely to show small distances to each other. This makes motif discovery become even more difficult. We therefore tested the performance of TreeMotif algorithms on synthetic motif-finding problems with background sequences of varying composition.

Table 3.9: Percentage of bases in different samples of DNA

No.	Source	A	G	C	T
1	<i>Beef sperm</i>	28.7	22.2	22.0	27.2
2	<i>Human thymus</i>	30.9	19.9	19.8	29.4
3	<i>Human liver</i>	30.3	19.5	19.9	30.3
4	<i>Human sperm</i>	30.7	19.3	18.8	31.2
5	<i>Hen red cells</i>	28.8	20.5	21.5	29.2
6	<i>Rat bone marrow</i>	28.6	21.4	21.5	28.4
7	<i>Herring sperm</i>	27.8	22.2	22.6	27.5
8	<i>Paracentrotus lividus (sea urchin) sperm</i>	32.8	17.7	18.4	32.1
9	<i>Salmon</i>	29.7	20.8	20.4	29.1
10	<i>Wheat germ</i>	26.5	23.5	23.0	27.0
11	<i>Yeast</i>	31.3	18.7	17.1	32.9
12	<i>Diplococcus pneumoniae</i>	29.8	20.5	18.0	31.6
13	<i>K-12 Escherichia coli</i>	26.0	24.9	25.2	23.9
14	<i>Mycobacterium tuberculosis</i>	15.1	34.9	35.4	14.6
15	<i>Bacteriophage T2</i>	32.5	18.2	16.7	32.6

In this section, datasets are generated by tuning $(G+C)$ -content as 30, 35, 40 to 45 percent with $l=15$, $d=4$, $n=600$, and $m=20$. Notes: (1) the content of G or C is a half the $(G+C)$ -content, and the content of A or T is a half the $100-((G+C)\text{-content})$; (2) base distribution in motif instances still follows i.i.d.. Experimental comparison is given by Table 3.10.

Deterministic sample-driven algorithms are sensitive to base compositions. When the $(G+C)$ -content is 40 to 45 percent, TreeMotif algorithms and DPCFG deliver consistent nPC. When the $(G+C)$ -content is 35 percent, they produce nPC near zero (and DPCFG begins to encounter memory shortage). This decrease in accuracy is due to that there can be hundreds of thousands of m -cliques formed by l -mer strings which are not target instances. A few random m -cliques are more conserved than the target one. Table 3.11 compares two cliques found in the same dataset. Because the information content of the random clique is higher than that of the target, TreeMotif algorithms will report the random one. Moreover, when the $(G+C)$ -content is 30 percent, TreeMotif-BF and DPCFG encounters memory shortage; TreeMotif-DF cannot produce results within days.

The aim of finding exact consensus motifs facilitates pattern-driven algorithms to deliver slightly better performance than sample-driven ones. For example, PMSprune and iTriplet deliver consistent accuracy when $(G+C)$ -content ranges from 35 to 45 percent. This is because they ignore random cliques that have no exact consensus, which would be discovered by TreeMotif algorithms and DPCFG. The random cliques might be more conserved than the target clique, and hence they may be reported. The new algorithms require further strategies to eliminate false cliques.

Table 3.12 shows the base counts along sites for cliques in Table 3.11. Most sites of the consensus of the random clique are more conserved than those of the target one. However, if the base on each site of the consensus is determined as one with more than 50 percent appearance, no exact consensus exists for the random clique because of no dominant bases for sites 11 and 14. Hence, if cliques are ranked according to the information

Table 3.10: Comparison: effect of varying $(G+C)$ -content on discovery accuracy

Algorithm	Percentage of $G+C$ (%)			
	30	35	40	45
GibbsMotifSampler	0.23±0.34	0.08±0.14	0.03±0.07	0.05±0.12
	0.26±0.37	0.10±0.18	0.04±0.09	0.06±0.14
MDscan	0.08±0.07	0.03±0.02	0.14±0.18	0.05±0.05
	0.15±0.10	0.07±0.04	0.25±0.29	0.10±0.09
MEME	0.44±0.36	0.31±0.41	0.16±0.20	0.08±0.10
	0.55±0.32	0.37±0.41	0.23±0.26	0.14±0.16
BioProspector	0.22±0.24	0.18±0.27	0.03±0.02	0.10±0.09
	0.32±0.29	0.25±0.28	0.05±0.03	0.17±0.15
PatternBranching	0.01±0.01	0.00±0.01	0.79±0.33	0.82±0.34
	0.01±0.02	0.01±0.01	0.83±0.33	0.84±0.34
PMSprune	0.12±0.32	0.91±0.05	0.92±0.05	0.93±0.06
	0.14±0.35	1.00±0.00	1.00±0.00	1.00±0.00
iTriplet	0.12±0.33	0.91±0.05	0.92±0.05	0.92±0.07
	0.15±0.34	1.00±0.00	1.00±0.00	1.00±0.00
DPCFG	-	-	0.95±0.07	0.93±0.10
	-	-	0.98±0.04	0.96±0.06
TreeMotif-BF	-	0.00±0.01	0.92±0.05	0.93±0.06
	-	0.01±0.01	1.00±0.00	1.00±0.00
TreeMotif-DF	-	0.00±0.01	0.95±0.07	0.93±0.10
	-	0.01±0.01	0.98±0.04	0.96±0.06

Note* for each algorithm, the first line: nPC, the second line: nSn; '-': fail.

Table 3.11: Example: discovered cliques

Sequence of length 600	Target Clique		Random Clique	
	Position	Instance	Position	Instance
seq-01	336	ctagactccgccgac	198	ataattctattgctt
seq-02	122	ctacactgcatggtg	5	ttaatgctatatattt
seq-03	20	ttatccgccccggtc	117	atgatggtattgttt
seq-04	355	ccatgctgcacgggc	98	ataattgtttccct
seq-05	401	ctttaccccgaggtc	39	ctactgtatagtct
seq-06	356	ctatactccatgcca	258	ataattgtttttct
seq-07	462	ctattctccgcgacc	377	atatgttatatgtt
seq-08	480	cttacctccacggcc	582	ctaattgtaatttt
seq-09	111	atattcttcacggtc	355	ttaagtgtatgcat
seq-10	14	ccaagctccccggtc	344	ataaatctaattgt
seq-11	28	cgacactccccggtc	127	ataattgtaatttca
seq-12	363	ctctactccatgata	245	ataattgttatctat
seq-13	158	ctctacgcgacgttc	460	ataattctatcatat
seq-14	578	ctatgtcaagagtc	524	ttaaattgtaatttat
seq-15	159	atatactggacgctc	467	ataattgtataggaa
seq-16	315	ctccactccaccgac	436	agaagtgtatgtttt
seq-17	496	ctatacttcagtttc	466	atagttttatatttt
seq-18	391	ccattctctccggtc	317	atacgtctatttctt
seq-19	542	ctatactctaagttt	442	atatttctaacttct
seq-20	278	ctagactacatggac	427	ataaatgtttgttct
Information constant	-	12.8084 bits	-	15.9742 bits
Consensus	-	ctatactccacggtc	-	ataattgtatxtxt

Note* 'x': each base appears with percentage less than 50 percent.

content, the random one will be reported; if they are ranked according to whether an exact consensus exists, the target one will be selected.

In spite of this superiority, when the $(G+C)$ -content is 30 percent, PMSprune and iTriplet also encounter a similar problem of finding a large number of random motifs. In addition, a limitation of pattern-driven algorithms can be seen from the preceding analysis. That is, if the target motif contained in a dataset has inexact consensus, they will fail to find it, which will be further discussed in Section 3.4.2. This is why we prefer to find instances of a consensus motif instead itself.

Besides deterministic algorithms, probabilistic ones should have also been unable to produce satisfactory results due to the increased similarity of background sequences. However, Table 3.10 shows that probabilistic algorithms produce increased accuracy nPC

Table 3.12: Base counts of cliques (given in Table 3.11)

Clique	Base	Position along a Consensus														
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Target	a	2	0	15	2	12	0	0	1	1	13	2	1	2	3	3
	c	17	3	3	3	2	20	1	14	16	3	12	4	2	3	15
	g	0	1	0	2	3	0	2	3	2	3	2	14	13	1	1
	t	1	16	2	13	3	0	17	2	1	1	4	1	3	13	1
Random	a	15	0	19	15	3	0	0	0	16	6	5	2	0	5	2
	c	2	0	0	2	0	0	6	0	0	0	3	2	4	7	0
	g	0	1	1	1	3	2	13	0	0	0	3	4	2	1	0
	t	3	19	0	2	14	18	1	20	4	14	9	12	14	7	18

and nSn. For example, MEME shows the best results when $(G+C)$ -content is 30 percent. This is possibly resulted from the scheme on the generation of the datasets. The datasets are generated with the biased base distribution while the motif instances are generated with the uniform base distribution. Due to the difference, statistic-based algorithms have a better capability of distinguishing the motif instances from the background sequences (than combinatorial algorithms).

Additional tests show that when the base distribution in motif instances approaches that in background sequences, probabilistic algorithms deliver decreased accuracy. For example, if the $(G+C)$ -content is 30 percent for both background sequences and motif instances, MEME delivers $nPC=0.21\pm0.23$ and $nSn=0.30\pm0.29$. The large deviations show that MEME either cannot perform consistently in such cases.

In summary, if the base distribution is heavily biased and is the same in both background sequences and motif instances, both probabilistic algorithms and deterministic combinatorial algorithms need further improvement to deliver satisfactory results. If the base distribution in motif instances shows large difference from that in background sequences, probabilistic algorithms show better capability of detecting the true signals than deterministic combinatorial ones. However, there is still much room for improving them to be consistent in accuracy.

3.4.1.7 Discussions on TreeMotif algorithms

The increase of n or p (or decrease of E) can result in many candidate l -mer substrings of sequences, which indicates that the number of random cliques of small sizes becomes large. From the preceding sections, it can be seen that the execution time of TreeMotif algorithms increases exponentially as n or p increases (or E decreases). This indicates that the depth-first search of cliques in TreeMotif-DF actually involves much overhead, resulting in a non-polynomial running time in practice. In addition, a large amount of space can be required by TreeMotif-BF. TreeMotif-DF does not suffer from memory shortage, but the overhead for checking the extendability of random cliques makes it time-consuming.

Candidate l -mer substrings to derive motifs can also be introduced by increasing the number of sequences in the dataset or biasing base distributions. In these cases, TreeMotif algorithms may become dataset-sensitive when the motif is very weak and can not deliver results within a consistent execution time.

Another limitation of TreeMotif algorithms is that they have difficulty in dealing with short weak motif problems, such as those with (13, 4)-, (15, 5)- or (18, 6)-motifs (with very small E values). Thus, future work is required to improve the algorithms to tackle such problems.

3.4.2 Biological data

In this section, TreeMotif algorithms are tested on real TFBS datasets, including *E.coli* CRP [162], *preproinsulin*, *DHFR*, *c-fos* [82], and *LexA* [64]. Assume these datasets satisfy the OOPS constraint. In practice, as not any information is known about the (l, d) of true motifs in the data, l is tested by starting from 6. For each l , different values of d ($d < l/2$) are tested to check if the target motif can be discovered.

Datasets *c-fos*, *DHFR* and *Preproinsulin* contain relatively strong motifs, which can be discovered by TreeMotif algorithms accurately. Table 3.13 includes the results.

Table 3.13: Comparison: performance of TreeMotif algorithms on real datasets

Data	Discovered	l, d	Published
<i>c-fos</i> [82]	<i>ccatattaggacatct</i>	16, 3	<i>ccatattaggacatct</i>
<i>DHFR</i> [82]	<i>ttcgcgccaact</i>	13, 2	<i>ttcgcgccaact</i>
<i>E.coli CRP</i> [162]	<i>xatgtgaxxxgxtcaca</i>	18, 6	<i>tgtagxxxgxtcaca</i>
<i>LexA</i> [64]	<i>tactgtatatatacagta</i>	20, 5	<i>tactgtatatatacagta</i>
	<i>xtactgtatatatacagtaxat</i>	24, 8	
<i>Pre-proinsulin</i> [82]	<i>cctcagcccc</i>	10, 2	<i>cctcagcccc</i> ,
	<i>agaccagca</i>	10, 2	<i>agaccagca</i> ,
	<i>ccctaattgggcca</i>	13, 2	<i>ccctaattgggcca</i>

Note* 'x': each base appears with frequency less than 50 percent.

3.4.2.1 *E.coli CRP* dataset

E.coli CRP dataset is composed of eighteen 105-mer sequences extracted from GenBank Release 55 [162]. This datasets satisfies the settings of the (l, d) -motif problem. There is at least one *CRP*-binding site (motif) contained in each sequence (OOPS). The motif *tgtagxxxgxtcaca* verified by biological experiments shows high weakness (that five positions on it cannot determine dominant bases).

By setting $l=18$ and $d=6$, TreeMotif algorithms detect *xatgtgaxxxgxtcaca* (determined by the instances reported) as the best motif. Note that the motifs in Table 3.13 discovered by the algorithms are inexact, i.e., the consensus motifs are not made up of unique bases at all positions. Such motifs make the discovery task more difficult.

By setting (l, d) as $(18, 6)$, TreeMotif algorithms produce results within 3 seconds. While with the same settings, DPCFG requires about 35 seconds, iTriplet requires about 4 minutes and PMSprune requires about 10 seconds (note that iTriplet and PMSprune yielded no motifs). The time reduction is small for a single dataset; however, when the number of datasets to be tested becomes large, more time can be reduced by the newly proposed algorithms. On the other hand, PROJECTION can give the consensus as *ttgtgatggagttcacattt* with settings $l=20, d=4$, yet it produces no motifs with the same weak setting as the proposed algorithms. All the discussions prove that TreeMotif algorithms can perform better than these known algorithms.

3.4.2.2 *LexA* dataset

The *LexA* dataset is composed of 16 sequences [64]. Each of the sequences contains more than one instance except for two, namely *himA* and *uvrC*. These two sequences are corrupted with no motif instances. That is, *LexA* is a typical noisy dataset.

By excluding the two sequences without motif instances (to obtain the OOPS data), all algorithms can discover the motif *tactgtatatatacagta* by a strong setting of $l=20$ and $d=5$. In addition, without exclusions, with the weak setting of $l=24$ and $d=8$, TreeMotif algorithms could discover the best matching motif as *xtactgtatatataaaacagtaxat*. That is, although the datasets might be noisy, there is also a possibility for TreeMotif algorithms to discover a weak motif. Note that along with the best matching motif, the shifted versions of this motif and other meaningful motifs are also discovered.

Generally, for inexact weak motifs, sample-driven algorithms exhibit better discovering capability than pattern-driven algorithms. For example, PMSprune fails to find the motifs in *E.coli* *CRP* and *LexA* under the same weak settings of l and d as TreeMotif algorithms. This is because signals from the motif instances are not strong enough to derive an exact pattern, whereas such a pattern is the prerequisite for pattern-driven algorithms [44].

3.4.2.3 A real benchmark dataset

Recently, using binding sites extracted from TRANSFAC data-base version 9.4, benchmark datasets are constructed for evaluating motif discovery algorithms [145]. The real benchmark data consist of 50 datasets. Each dataset contains 5 to 18 sequences with the maximum length of 2000 bp. In addition, each sequence in a dataset contains an instance of a non-gapped motif.

The performances of TreeMotif algorithms on the real benchmark data is compared with that of Weeder [128]. Weeder detects motifs using suffix tree, and performs better

Table 3.14: Comparison: performance of algorithms on another real benchmark dataset

Algorithm	(l, d)	nSn	nPPV	nSP	nPC	nCC	nASP
MEME	-	0.10±0.20	0.09±0.20	0.98±0.01	0.06±0.14	0.08±0.20	0.10±0.20
Weeder	-	0.20±0.24	0.07±0.10	0.96±0.03	0.06±0.07	0.10±0.14	0.14±0.16
TreeMotif	(10, 2)	0.19±0.26	0.08±0.13	0.98±0.01	0.07±0.10	0.11±0.17	0.13±0.19
	(11, 2)	0.18±0.29	0.08±0.14	0.98±0.01	0.06±0.12	0.10±0.19	0.13±0.21
	(12, 2)	0.16±0.28	0.08±0.16	0.99±0.02	0.06±0.12	0.10±0.19	0.12±0.21
	(12, 3)	0.23±0.29	0.10±0.16	0.98±0.02	0.08±0.13	0.13±0.20	0.16±0.22

Note* '-': not provided [145].

than the other tools according to the comparisons in [174]. The nucleotide level metrics nSn, nPPV, nSP, nPC, nCC and nASP are used to compare different algorithms. The result of all algorithms are evaluated using the web benchmark tool provided by [145]. Also note that the performance of Weeder is obtained from the website provided by [145].

As shown in Table 3.14, by roughly setting (l, d) for all datasets, e.g., (10, 2), TreeMotif algorithms delivers comparable performances to Weeder (note that TreeMotif algorithms are set to output ten OOPS cliques of motif instances for all tests). With a weaker setting of (12, 3), TreeMotif algorithms show slightly better performances than Weeder. Furthermore, as seen from Figure 3.9, by fine-tuning l and d for each dataset empirically, TreeMotif algorithms achieves much higher nSn compared to Weeder even with lower false positive rate (i.e., 1-nPPV). This also indicates the nPC, nCC and nASP of TreeMotif algorithms are higher than Weeder.

The standard error of Weeder for each statistic is smaller than TreeMotif algorithms. This is because, taking nCC as an example, it produces nCC larger than 0.4 for only one dataset; while TreeMotif algorithms produce eight nCCs larger than 0.4 (the corresponding information can also be seen in Figure 3.8, which shows the QQ-plot of nCC of TreeMotif-BF and Weeder). The large difference in all nCCs of TreeMotif algorithms results in the larger standard error.

Also, it is observed that TreeMotif algorithms produce lower nCC than Weeder for 16 datasets. All sequences in these 16 datasets have an average length of 1203 and the

Figure 3.8: QQ-plot of nCC of TreeMotif-BF and Weeder

Figure 3.9: Comparison: performance of algorithms on a real benchmark dataset

GC contents are around 0.45 or 0.55. That is, the bases A, C, G, T are not uniformly distributed in such long sequences. Under this circumstance, many random cliques of spurious instances appear [27]. Random cliques can disguise the targets as they might possess higher information content. These facts indicate that better strategies are required for separating real motifs from the spurious ones.

Figure 3.9 also exhibits the performances of several other algorithms with the same settings. BioProspector achieves comparable nSn to TreeMotif algorithms. However, from its lower nPPV it can be inferred that many false positives are also reported. This makes its performance statistics, including, nPC, nCC and nASP, lower than TreeMotif algorithms. iTriplet does not have any motif ranking strategies, the first reported ten motifs are selected as the final results and evaluated by using the web tool respectively (with the highest performance reported here). This arbitrary selection of final motifs may result in its lower statistics. MDscan reports motif instances on both reverse and forward strands. After excluding the instances reported on the reverse strand (note that

the benchmark data are guaranteed with motifs only on the forward strand [145]), it still shows lower performance statistics than the other algorithms. This might be due to that MDscan requires a certain number of sequences (in each dataset) with higher probability to contain motif instances arranged as the top sequences. Meanwhile, as the number of sequences is usually small in the benchmark data (e.g., thirty-five of fifty datasets contain only five to eight sequences), a dataset might not provide enough strong motif signals for MDscan to detect as the seeds for further refinement. PMSprune cannot find motifs for eight of fifty datasets with the same parameter settings (while TreeMotif algorithms can give true positives for all these eight datasets), because exact consensus motifs do not exist. Moreover, it does have strategies for ranking candidate motifs. These factors lead to the low statistics of PMSprune.

3.5 Summary

The existing graph-based algorithms for WMD can be time consuming due to redundant duplications and deletions on intermediate information. Much computational time is wasted when the number of (sub-)cliques becomes large.

In this chapter, algorithms TreeMotif-BF and TreeMotif-DF have been introduced for discovering motifs from exact datasets. TreeMotif-BF builds up initialized trees of motif instances in a breadth-first manner. The appending operations happen when cliques indicated by the leaf nodes are extendable to certain new nodes. The novel strategy of clique construction can avoid trivial operations on intermediate results. By doing motif refinement, the sensitivity of TreeMotif-BF can be higher than those producing exact OOPS motif instances. TreeMotif-DF constructs trees of motif instances in a depth-first manner. This reduces the space requirement compared to TreeMotif-BF. TreeMotif-DF prepares a Hamming distance table to reduce redundant calculations. It is more

consistent in handling a large number of sequences than TreeMotif-BF, while the latter is more efficient for handling very weak motifs and long sequences.

The performance of TreeMotif algorithms have also been compared with several deterministic or probabilistic algorithms. It has been demonstrated that TreeMotif algorithms are excellent in handling MCP and several extended (l, d) -motif problems, although they are not applicable in finding very weak motifs such as a $(18, 6)$ -motif or motifs in sequences with biased base distribution. However, TreeMotif algorithms have shown better performance on handling real biological datasets.

In Chapter 4, a space-efficient algorithm RecMotif will be proposed without preparing Hamming distance tables, which will also show improved efficiency in execution time.

Chapter 4

Graph-based Recursive Algorithm for WMD in Exact Datasets

In Chapter 3, we have presented two tree-structured algorithms that associate motif instances as nodes of trees based on initialized graphs. TreeMotif algorithms reduce the need to maintain redundant information resulting in efficiency in execution time and space. However, the space requirement of TreeMotif-BF and many other known algorithms still increase exponentially when the weakness of motifs increases. TreeMotif-DF does not have limitations in space. Nevertheless, TreeMotif-DF uses Hamming distance tables to reduce redundant calculations, which incur additional execution time.

In this chapter, we will present a space-efficient algorithm, RecMotif, which finds cliques of motif instances with recursively constructed graphs in a depth-first manner. We will provide experiments to show the efficiency of RecMotif.

4.1 Introduction

The existing exact algorithms for weak motif discovery still need improvement on efficiency in time or space. When an algorithm is not space-efficient, it might fail to produce results. We therefore propose an exact graph-based algorithm RecMotif. RecMotif constructs cliques of motif instances one after another, requiring significantly reduced space. RecMotif is inspired by the following ideas.

For weak (l, d) -motif discovery, the notion of a “reference sequence” (as well as “reference word/vertex/substring”) has been used in a number of algorithms. A reference sequence is expected to contain at least one true motif instance. A reference word/vertex/substring decreases candidate motif instances. If it is a true instance, the other true instances in the remaining sequences could be easily elicited by search algorithms.

Given an exact dataset, the existing pattern- or sample-driven algorithms firstly select one or two sequences as the reference. Thereafter, pattern-driven algorithms generate new candidate motifs according to the vertices of the reference sequences, and they verify the candidates by using the remaining non-reference sequences. Differently, sample-driven algorithms find cliques of motif instances in graphs. Examples include MULTIPROFILER [82], the PMS series [43, 44, 136, 137], iTriplet [66], DPCFG [198] and TreeMotif algorithms. These algorithms have shown efficiency in time or space in solving problems such as MCP.

MULTIPROFILER uses each substring of a reference sequence to select substrings of the other sequences, and it recovers the backbone of a motif based on multi-positional matrices converted from selected substrings. PMSprune arranges d -neighbors of each substring of a reference sequence in a tree structure, and it verifies them effectively with a branch-and-bound strategy by using the remaining sequences. iTriplet firstly uses each pair of substrings of two reference sequences to construct triplets, any of which consists of the pair of reference substrings and a selected substring; after that, it generates and hashes candidate motifs according to each triplet; finally, it reports any motif that is associated with a minimum quorum of the sequences.

DPCFG uses each substring of a reference sequence to initialize m -partite graphs, and it constructs lists of motif instances at each vertex of them. Referring to reference node pairs of two reference sequences, TreeMotif algorithms initialize graphs of candidate motif instances and convert them into tree representations with false instances eliminated.

These algorithms use a general method that combines a reference process and a strategy of motif construction. This method has shown success in motif discovery. However, there is still much room for further improvement. For example, PMSprune is efficient in both time and space for short motifs while it misses target motif instances with an inexact consensus; TreeMotif-BF improves efficiency in execution time for long motifs compared with other graph-based algorithms, but it can suffer from a large space requirement.

The process of reference is effective in reducing the number of candidate motif instances. Intuitively, we can apply a reference process again over the motif candidates that are resulted from the first reference process. This is expected to further reduce the number of candidate motif instances (if the two reference vertices used are true motif instances). These findings lead to an exact and efficient recursive algorithm, RecMotif, for weak motif discovery.

4.2 RecMotif algorithm

RecMotif finds cliques of motif instances in graphs, which are recursively constructed by using the l -mer substrings of the input sequences. Substrings belong to two classes in each recursion of graph construction. The first class is referred to as reference vertex, and the second class is candidate vertex selected according to reference vertex. A general description of the algorithm is given in Algorithm 10.

All the l -mer substrings of the input sequences are represented as the vertices of an initial graph, which will be used as the first ‘last graph’. Thereafter, a reference sequence is selected from the ‘last graph’; according to each reference vertex, a part of vertices related to non-reference sequences are selected from the ‘last graph’. A ‘new graph’ is built up by these selected vertices and the reference vertex. If a ‘new graph’ is always built with at least one vertex from each of the non-reference sequences in the ‘last graph’, the

Algorithm 10 RecMotif (general description)

input: A set of m sequences $\{s_t\}_{t=1}^m$, l and d .
output: ranked m -cliques
begin:
 initialize a graph G_0 with all the l -mers of $\{s_t\}_{t=1}^m$, and vertices are collected in $\{P_{0,t}\}_{t=1}^m$;
 initialize r as 1 and a null list L ;
 sub-function RecMotif($r, \{P_{r-1,t}\}_{t=r}^m$):
 if r is less than or equal to m
 for each vertex x in $P_{r-1,r}$
 build a graph G_r with vertices in $\{P_{r-1,t}\}_{t=r+1}^m$, vertices collected in $\{P_{r,t}\}_{t=r+1}^m$;
 if all sets $\{P_{r,t}\}_{t=r+1}^m$ are non-empty
 inset x into list L ;
 RecMotif($r+1, \{P_{r,t}\}_{t=r+1}^m$);
 end if
 end for
 else
 record l -mers in L (ranked according to information content);
 end if
 end sub-function
end

graph construction can be continued recursively. Finally, if no non-reference sequences are left, all the reference vertices forms an m -clique of motif instances.

4.2.1 Reference selection

As RecMotif aims to handle OOPS datasets, how the reference sequences are selected will not influence the results. Therefore, it is feasible to use all the sequences as reference sequentially as ordered in a given dataset.

4.2.2 Graph construction with one reference sequence

Related notations are the same as those in Chapter 3. Similarly, a sliding window of length l is used to obtain all the l -mer vertices (or substrings) of sequences $\{s_k\}_{k=1}^m$. For $k \in [1, m]$, all the vertices from s_k are inserted into the set $P_{0,k} = \{v_{k,j}\}_{j=1}^{n-l+1}$, where number 0 indicates the sets are initial. Let $\|P_{0,k}\|$ denote the number of vertices in $P_{0,k}$.

Suppose the position of the true motif instance in $P_{0,1}$ is known, and it is selected as the reference vertex. If we build up a graph with vertices in $\{P_{0,k}\}_{k=2}^m$ that show up to $2d$ distance from the reference vertex, all the other motif instances in the remaining sequences must have been included in the graph. That is, if let $\{P_{1,k}\}_{k=2}^m$ denote the sets of vertices of the graph constructed according to vertices in $\{P_{0,k}\}_{k=2}^m$, all the true motif instances must appear in $\{P_{1,k}\}_{k=2}^m$. In reality, as the position of the true motif instance in $P_{0,1}$ is not known, all the vertices in $P_{0,1}$ have to be tested to guarantee that a graph containing all the remaining true motif instances is constructed.

4.2.3 Recursive graph construction

RecMotif extends the above idea of constructing graphs of motif instances with reference vertex of reference sequence. That is, it carries out the second time reference process for each vertex in $P_{1,2}$ on vertices in $\{P_{1,k}\}_{k=3}^m$. Specifically, for each reference vertex in $P_{1,2}$, it selects vertices from $\{P_{1,k}\}_{k=3}^m$ to build a new graph with vertices collected in

$\{P_{2,k}\}_{k=3}^m$. Similarly, if the selected reference vertex in $P_{1,2}$ is a true motif instance, all the remaining true motif instances must appear in $\{P_{2,k}\}_{k=3}^m$. Further, it conducts the third time reference process for each vertex in $P_{2,3}$ on vertices in $\{P_{2,k}\}_{k=4}^m$ to construct a graph with vertices collected in $\{P_{3,k}\}_{k=4}^m$. The algorithm continues in this way until all the sequences have been used as the reference.

During the recursive graph construction process, all currently used reference vertices will form a path. The path continuation condition is: if and only if all the remaining sequences contribute at least one vertex that is within $2d$ with all the current reference vertices. Otherwise, the algorithm substitutes the last reference vertex with another one from the same set as it to form a new path. If there are no more new vertices in the corresponding set, the algorithm deletes the last reference vertex and backtracks to the last second reference vertex and finds a substitute for it. The deletion, backtracking, and substitution process is repeated until the path continuation condition is satisfied.

Details about RecMotif are as follows. RecMotif takes all the l -mer substrings from s as input. Suppose G_1 is an initial graph containing all vertices in $\{P_{0,k}\}_{k=1}^m$. Let $\{G_i\}_{i=2}^m$ denote graphs constructed in the recursive process. For $\{i\}_{i=1}^{m-1}$, let $\{P_{i,k}\}_{k=i+1}^m$ denote the set of vertices of graph G_{i+1} that is built by vertices in $\{P_{i-1,k}\}_{k=i}^m$ of G_i according to a reference vertex in $P_{i-1,i}$. As RecMotif deals with each reference vertex sequentially, $\{P_{i,k}\}_{k=i+1}^m$ can be used repeatedly, which reduces the space requirement.

Algorithm 11 shows pseudo-code of RecMotif. For each $v_{i,r}$ in $P_{i-1,i}$, $\{P_{i,k}\}_{k=i+1}^m$ are initialized as null. For each $v_{k,j}$ from $P_{i-1,k}$, insert it in $P_{i,k}$ if $D(v_{i,r}, v_{k,j}) \leq 2d$. FLAG indicates if the recursive path can be continued. RecMotif inserts the reference vertices into set L . If $i=m+1$, the path formed by elements in L indicates an m -clique. A consensus motif can possibly be derived from L , with which the motif can be refined by local alignments with given sequences. Specifically, RecMotif starts from $i=1$ with initial sets $\{P_{0,k}\}_{k=1}^m$, i.e., the first reference sequence is s_1 . When RecMotif($m+1$) is reached, it indicates that a clique of size m have been found.

Figure 4.1: RecMotif: an example

4.2.4 RecMotif: an example

An example about how RecMotif works is shown in Figure 4.1 with $m=4$. Assume $P_{0,1} = \{A, B, C\}$, $P_{0,2} = \{Z, E, F, G\}$, $P_{0,3} = \{H, I, J, K, L\}$, $P_{0,4} = \{M, N, O, Q, R, T\}$. Each capital letter stands for a vertex (substring) of length l .

Moreover, assume the following relationships on pair-wise Hamming distances of vertices involved in the explanation of the example. Let $A_{2d} = \{Z, E, G, H, J, M, O, R, T\}$ indicate that the set of vertices which have Hamming distances not more than $2d$ with vertex A . Note that the other vertices not included in A_{2d} have Hamming distances more than $2d$ with vertex A . Similarly, we assume $Z_{2d} = \{A, H, J, M, T\}$; $E_{2d} = \{A\}$; $G_{2d} = \{A, T\}$; $H_{2d} = \{A, Z\}$; $J_{2d} = \{A, Z, M, T\}$; $M_{2d} = \{A, Z, J\}$; $O_{2d} = \{A\}$; $R_{2d} = \{A, H\}$; $T_{2d} = \{A, Z, G, J\}$.

With the above assumptions, $\text{RecMotif}(i)$ starts with $i = 1$. Firstly, vertex A from $P_{0,1}$ is selected as the reference vertex. $P_{1,2}$ is set as empty. All the vertices in $P_{0,2}$ as shown in Figure 4.1 will be checked if they are within $2d$ with A . According to the assumptions, vertex A is within $2d$ with Z, E, G from $P_{0,2}$ (connected by dotted lines in Figure 4.1). Thus $P_{1,2}$ is obtained as $\{Z, E, G\}$. Similarly, $P_{1,3} = \{H, J\}$ and $P_{1,4} = \{M, O, R, T\}$, as has been shown in Figure 4.2(a).

Algorithm 11 RecMotif(i): recursive motif discovery

```

1: input : vertices  $\{P_{i-1,k}\}_{k=i}^m$  of  $G_i$ .
2: output: cliques of motif instances.
3: if  $i \leq m$  then
4:   for each  $v_{i,r} \in P_{i-1,i}$  do
5:     FLAG  $\leftarrow$  TRUE
6:     for  $\{P_{i,k}\}_{k=i+1}^m$  do
7:        $P_{i,k} \leftarrow \emptyset$ 
8:       for each  $v_{k,j} \in P_{i-1,k}$  do
9:         if  $D(v_{i,r}, v_{k,j}) \leq 2d$  then
10:           $P_{i,k} \leftarrow P_{i,k} \cup \{v_{k,j}\}$ 
11:        end if
12:      end for
13:      if  $\|P_{i,k}\| = 0$  then
14:        FLAG  $\leftarrow$  FALSE
15:        break
16:      end if
17:    end for
18:    if FLAG = TRUE then
19:       $L_i \leftarrow v_{i,r}$ 
20:      RecMotif( $i+1$ ) with  $\{P_{i,k}\}_{k=i+1}^m$ 
21:    end if
22:  end for
23: else
24:   return  $L$ 
25: end if

```

Figure 4.2: RecMotif: graph construction for vertex A

As can be seen, for vertex A , vertices that have Hamming distance not more than $2d$ with it can be found from all other sets: $P_{0,2}$, $P_{0,3}$ and $P_{0,4}$. Thus there is possibility for vertex A to be a motif instance. Therefore, L_1 is set as A and RecMotif(i) with i being 2 is to be proceeded. (RecMotif(1) is suspended temporarily. Moreover, note that $\{\{P_{i,k}\}_{i=1}^{m-1}\}_{k=i+1}^m$ must be set as null each time before using them. In the following explanation, the description on such operations is not repeated.)

Next, vertex Z from $P_{1,2}$ is selected as the reference vertex. All vertices in $P_{1,k}$, $k = 3, 4$, will be checked. Vertex Z is within $2d$ with H, J from $P_{1,3}$ and M, T from $P_{1,4}$, as shown by the links in Figure 4.2(a). That is, we can obtain $P_{2,3} = \{H, J\}$ and $P_{2,4} = \{M, T\}$, as shown in Figure 4.2(b). Also, there is possibility for vertex Z to be a motif instance. L_2 is set as Z and RecMotif(i) with i being 3 is to be proceeded. At the same time, RecMotif(2) is suspended temporarily.

Next, vertex H from $P_{2,3}$ is selected as the reference vertex. All vertices in $P_{2,k}$, $k = 4$, will be checked. As H has no vertex from $P_{2,4}$ that has a Hamming distance not more than $2d$ with it, current $P_{3,4}$ is null. Thus L_3 will not be set as H . In other words, there

will be no path to form an m -clique involving vertex H with the current elements in L , as shown in Figure 4.2(c). The process will have to be continued in RecMotif(3) with another substitute vertex from $P_{2,3}$.

Next, vertex J , which is the next vertex to be checked in the same set $P_{2,3}$ with H , is selected as the reference vertex. All vertices in $P_{2,k}$, $k = 4$ will be checked. As $D(J, M) \leq 2d$ and $D(J, T) \leq 2d$, $P_{3,4}$ is obtained as $\{M, T\}$. As there is possibility for vertex J to be a motif instance, L_3 is set as J . Then RecMotif(3) is suspended temporarily and RecMotif(i) with i being 4 is to be proceeded.

Next, vertex M from $P_{3,4}$ is selected as the reference vertex. As $i=m$, indicating vertex M is a vertex in the last sequence thus there are no more vertices to be checked for vertex M , L_4 is directly set as M . RecMotif(4) is suspended temporarily and RecMotif(i) with i being 5 is to be proceeded.

Now, i is larger than m . This means that the recursive path of an m -clique of motif instances is formed, as shown in Figure 4.2(d). The elements forming the clique have been stored in L as: A, Z, J, M .

At this point, the process for vertex M has been completed, i.e. the current RecMotif(5) process is completed. Then RecMotif returns to the process of RecMotif(4). The process that was carried out for vertex M is repeated for vertex T from $P_{3,4}$. Using this process another clique is obtained as: A, Z, J, T from L according to the assumptions.

As vertex T is the last vertex in $P_{3,4}$, the RecMotif(4) process is completed. Thus RecMotif returns to the process of RecMotif(3). As vertex J is the last vertex in $P_{2,3}$, the RecMotif(3) process is completed. Thus RecMotif returns to the process of RecMotif(2). Hitherto, the whole process for vertex Z from $P_{1,2}$ has been completed.

As there are still vertices E, G in $P_{1,2}$, the above process for vertex Z is repeated similarly on E and G sequentially. When the process for vertex G is completed, it means that the process of RecMotif(2) has been completed. Thus RecMotif finally returns to the process of RecMotif(1).

At this point, the process for vertex A has been completed. As there are still vertices B and C in $P_{0,1}$, the above process for vertex A is repeated similarly on B and C respectively. When the process for vertex C is completed, it means that the RecMotif(1) process has been completed. That is, the whole algorithm is completed with all possible cliques found (if any).

4.2.5 Time and space complexity

4.2.5.1 Time complexity

Assume that the sequence length is n and there are m *i.i.d.* sequences in total. The time complexity of RecMotif is now analyzed by estimating the number of Hamming distance calculations in the worst case.

Let $f(n-l+1, m)$ denote the number of Hamming distance calculation for a node in the first reference sequence. According to the algorithm, we have $f(n-l+1, m) = (m-1)(n-l+1) + (n-l+1)f(n-l+1, m-1)$. In general, for $i \in [2, m]$, we have $f(n-l+1, i) = (i-1)(n-l+1) + (n-l+1)f(n-l+1, i-1)$. For the base case of recursion, we do not need to calculate any Hamming distance, that is, $f(n-l+1, 1) = 0$. As there are $n-l+1$ reference nodes in the first reference sequence, the overall number of Hamming distance calculation is $(n-l+1)f(n-l+1, m)$. Expanding the formula, we have $(n-l+1)f(n-l+1, m) = (m-1)(n-l+1)^2 + (n-l+1)^2 f(n-l+1, m-1) = (m-1)(n-l+1)^2 + (n-l+1)^2 ((m-2)(n-l+1) + (n-l+1)f(n-l+1, m-2))$. Then, the total number of Hamming distance calculation is $\sum_{i=1}^{m-1} (m-i) \times (n-l+1)^{i+1}$. As $n \gg m$ the time complexity is $O(n^m)$ in the worst case.

The time complexity in the worst case grows exponentially. However, it is expected that along with deeper recursion, the number of nodes for subsequent recursions would be quickly decreased. For example, for each reference node in the first sequence, the number of candidate nodes to check is $(m-1)(n-l+1)$. The expected number of nodes

that have Hamming distance less than or equal to $2d$ from the first reference node in each of the remaining sequences is $(n - l + 1)p$ (p is discussed in Section 3.2.5 of Chapter 3). As a result, for each node related to the second reference sequence, the number of candidate nodes to check is $(m - 2)(n - l + 1)p$. Although the probability that a new candidate node has Hamming distance less than or equal to $2d$ to any collected reference node will increase, the candidate nodes to be checked in the subsequent recursions will be expected to be further reduced. This allows the algorithm to become practical.

Table 4.1: Comparison: time and space complexities (RecMotif included)

Algorithms	Time Complexity	Space Complexity
PMSprune	$O(mn^2N(l, d))$	$O(mn^2)$
iTriplet	$O(mn^3l^3d^2)$	$O(N(l, d))$
DPCFG	$O(n^m)$	$O(n^{m-2})$
TreeMotif-BF	$O(n^m)$	$O(n^{m-3})$
TreeMotif-DF	$O(n^m)$	$O((mn)^2)$
RecMotif	$O(n^m)$	$O(m^2n)$

$$\text{Note* } N(l, d) = \sum_{i=0}^d C_i^l 3^i.$$

4.2.5.2 Space complexity

Before the algorithm starts, the number of vertices to record is $m(n - l + 1)$. For the first recursion, the number of vertices to record is $(m - 1)(n - l + 1)$ in the worst case. In general, for the i^{th} ($i \in [1, m]$) recursion, the number of vertices to record is $(m - i)(n - l + 1)$ in the worst case. Then, the total number of vertices is $\sum_{i=1}^m (m - i)(n - l + 1)$. The space complexity is $O(m^2n)$ in the worst case.

From Table 4.1 it can be seen that RecMotif consumes much less space than the other algorithms. The space complexity of iTriplet increases exponentially with respect to l or d . For DPCFG and TreeMotif-BF consume exponentially increased space with respect to m . This results in their difficulty in dealing with weak motifs. While PMSprune requires much less space than DPCFG and TreeMotif algorithms, the space required is still greater than RecMotif because $n \gg m$.

4.3 Experimental results

4.3.1 Synthetic data

For experiments in Section 3.4.1 of Chapter 3, RecMotif can achieve comparable accuracy to TreeMotif algorithms if all of them report m -cliques. This is because these algorithms can discover all the possible m -cliques, which are ranked according to the same statistic called information content. Thus the reported m -cliques are also the same. The accuracy of RecMotif can thus be seen from, for example, Figures 3.6 and 3.7 (the same curve as TreeMotif-DF).

In this section, we only compare the execution time of different algorithms on finding weak motifs in synthetic datasets. The parameters tuned include m , n , l , d , and biasing base distributions of the input sequences. The results for all algorithms are collected from tests on a workstation with 2.66 GHz CPU and 3.00 GB RAM. For the problems given in Table 3.2, memory consumption of RecMotif is less than 33 mygabyte. In the following discussions, if an algorithm does not encounter memory shortage during running, we deem its memory requirement as satisfactory. All the statistics are averaged over 10 datasets. Datasets are the same as those used in Chapter 3.

4.3.1.1 Motif challenge problem

Table 4.2 shows the comparisons of RecMotif with the other existing algorithms on the $(15, 4)$ -motif problem. It can be seen that RecMotif requires an execution time comparable to TreeMotif-BF and DPCFG and lower than other methods. In the following sections, it will be shown that RecMotif is capable of discovering weaker motifs with less execution time than algorithms such as TreeMotif algorithms.

4.3.1.2 Impacts of increasing length of sequences

In this section, RecMotif is compared with other algorithms by increasing the length of background sequences n with $m=20$ and $(l, d)=(15, 4)$. Figure 4.3 shows the execution

Table 4.2: Comparison: motif challenge problem (RecMotif included)

Algorithm	Execution Time
GWM2	3.9±0.1
PMSprune	5.2±0.1
iTriplet	186.0±5.3
DPCFG	0.7±0.0
TreeMotif-BF	0.5±0.0
TreeMotif-DF	3.4±0.0
RecMotif	0.7±0.0

Note* time unit: second.

time of TreeMotif algorithms, RecMotif, PMSprune and DPCFG.

As seen, with a better theoretical space complexity than TreeMotif algorithms, RecMotif also requires a lower execution time than them in practice. When the sequence length is below 1800, RecMotif requires the lowest execution time among all the algorithms. It consumes less than a half the execution time of DPCFG for datasets with sequences longer than 1300. For example, for sequences of length as 2000, the execution time of RecMotif is about one quarter that of DPCFG; it is slightly lower than that of TreeMotif-BF and about three fifths that of TreeMotif-DF. On the other hand, RecMotif requires more time than PMSprune for handling sequences longer than 1800 bp.

4.3.1.3 Impacts of increasing number of sequences

In this section, we examine if the efficiency of RecMotif in execution time is sensitive to the quantity of sequences. The test results are shown in Table 4.3.

When the number of sequences is not large (say $m \leq 40$), it can be seen that the execution time required by RecMotif is much less than other algorithms, although it also increases with the increase of the number of sequences. When the number of sequences is large enough (say $m=100$), RecMotif begins to perform inconsistently showing large deviations on execution time, for which the reasons are similar to discussions in Section 3.4.1.3 of Chapter 3. Nevertheless, RecMotif is more consistent than other sample-driven algorithms. The strategy that RecMotif handles substrings of sequences in breadth and

Figure 4.3: Comparison: effect of increasing n on execution time. (RecMotif included)

Table 4.3: Comparison: effects of increasing m on execution time (RecMotif included)

Algorithms	m						
	20	25	30	35	40	50	100
DPCFG	0.7±0.1	0.6±0.1	0.6±0.0	0.7±0.1	0.7±0.1	1.1±0.2	1.2±0.4
GWM2	4.5±0.3	4.3±0.3	4.1±0.3	4.4±0.3	4.6±0.3	4.5±0.3	6.3±2.2
iTriplet	10.6±0.4	10.9±0.3	10.7±0.4	10.7±0.8	10.6±0.3	9.9±0.4	9.9±0.3
PMSprune	0.2±0.0	0.2±0.0	0.4±0.0	0.4±0.0	0.5±0.0	0.7±0.0	1.4±0.0
TreeMotif-BF	0.3±0.0	0.3±0.0	0.3±0.0	0.3±0.0	0.3±0.0	0.3±0.2	4.4±4.8
TreeMotif-DF	0.3±0.0	0.3±0.0	0.3±0.0	0.3±0.0	0.3±0.0	0.4±0.0	1.0±0.6
TreeMotif-DF*	0.6±0.1	0.5±0.0	0.6±0.0	0.5±0.0	0.5±0.0	0.6±0.1	1.1±0.6
RecMotif	0.2±0.0	0.2±0.0	0.2±0.0	0.2±0.0	0.2±0.0	0.3±0.0	0.6±0.2

 Note* time unit: $\times 10^2$ second.

depth simultaneously possibly contributes to this performance. Overall, RecMotif is better than all the other algorithms even for $m=100$.

4.3.1.4 Impacts of increasing l and d with fixed weakness p

In this section, experiments are carried out with the value of p fixed at around 0.05. Table 4.4 shows the experimental results. RecMotif shows a consistent execution time, which is the same as TreeMotif algorithms. It also can be seen that, without preparing Hamming distance tables, RecMotif can proceed more efficiently than TreeMotif-DF.

4.3.1.5 Impacts of increasing weakness p

In this section, the value of p is increased from 0.03 to 0.29 by varying the values of l and d as listed in Table 3.7 of Chapter 3. Figure 4.4 shows the results.

RecMotif shows a similar trend to TreeMotif algorithms when increasing p . It is more capable of handling long weak motifs than the other algorithms. For a (24, 8)-motif, RecMotif requires a half the execution time of TreeMotif algorithms and only about one-quarter the execution time of PMSprune. For a (40, 14)-motif, RecMotif can produce results within 5 hours while TreeMotif algorithms requires up to 8 hours and the other algorithms are not comparable due to shortage of memory or failing to produce results within 10 hours.

Table 4.4: Comparison: effects of varying (l, d) with fixed p on execution time (RecMotif included)

$(l, d), p$	DPCFG	PMSprune	iTriplet	TreeMotif-BF	TreeMotif-DF	RecMotif
(12, 3), 0.05	0.8	1.6	173.6	0.6	3.0	0.6
(18, 5), 0.06	0.6	16.9	230.4	0.5	4.2	0.7
(21, 6), 0.06	0.5	46.5	250.0	0.4	4.5	0.7
(24, 7), 0.06	0.5	80.2	291.5	0.4	3.9	0.6
(27, 8), 0.05	0.4	137.1	354.4	0.4	4.3	0.6
(30, 9), 0.05	0.4	242.9	553.9	0.4	4.3	0.6
(33,10), 0.05	0.4	405.2	1050.2	0.4	4.4	0.6
(36,11), 0.05	0.3	651.8	1418.9	0.4	3.6	0.5
(39,12), 0.04	0.3	1056.0	2779.2	0.4	3.8	0.5
(42,13), 0.04	0.3	1841.8	2895.5	0.4	3.6	0.5
(44,14), 0.06	0.7	19213.5	-e	0.6	7.1	1.0
(47,15), 0.06	0.6	28896.2	-e	0.7	7.0	1.0
(50,16), 0.06	0.5	-o	-e	0.8	6.5	1.0

Note* '-o': more than 10 hours; '-e': error on memory; time unit: second.

However, RecMotif still cannot produce results for problems in Table 3.8 within a reasonable execution time. Overall, RecMotif is more scalable on both the value p and the motif length l compared to the other algorithms.

4.3.1.6 Impacts of tuning background base distribution

In this section, we examine RecMotif by tuning base compositions of sequences. Table 4.5 shows the results. With a low $(G+C)$ -content of 30 percent, RecMotif still cannot produce results within a reasonable execution time; with a $(G+C)$ -content larger than 35 percent, RecMotif performs more consistently than TreeMotif algorithms. However, all sample-driven algorithms in these cases require improvement to deliver satisfactory performance.

4.3.2 Biological data

RecMotif is tested on *E.coli CRP* [162], *preproinsolin*, *DHFR* and *c-fos* [82], and *LexA* datasets, with the same assumptions as discussed in Section 3.4.2 of Chapter 3.

Figure 4.4: Comparison: effect of increasing p (RecMotif included) on execution time
 Note* '-o': more than 10 hours; '-e': error on memory.

Table 4.5: Comparison: effect of varying ($G+C$)-content on execution time

Algorithm	Percentage of $G+C$ (%)			
	30	35	40	45
PMSprune	21.4±2.3	17.3±1.2	11.1±0.8	7.4±0.3
DPCFG	-	-	6.8±3.7	1.2±0.1
TreeMotif-DF	-	203.1±352.0	10.3±1.3	7.2±0.4
TreeMotif-BF	-	174.1±313.0	2.5±0.8	1.0±0.1
RecMotif	-	30.3±37.2	1.8±0.4	0.9±0.1

Note* '-': fail; time unit: second.

RecMotif can effectively and efficiently discover motifs in datasets of *c-fos*, *DHFR* and *Preproinsulin*. For the (18, 6)-motif in *E.coli CRP* dataset, RecMotif can discover the motifs within 1 second; for the (24, 8)-motif in *LexA* dataset, it requires about 3 seconds (correspondingly, TreeMotif-BF requires about 4 seconds, and PMSprune requires more than 10 minutes without reporting any motifs).

For the real benchmark dataset [145], RecMotif can deliver comparable accuracy to TreeMotif algorithms by reporting m -cliques of (l, d) -motifs. Importantly, it shows improved efficiency in execution time. For example, for 39 of the 50 datasets, RecMotif requires about 21 hours in total to find (12, 3)-motifs, while TreeMotif-BF requires around 92 hours. For each of the remaining 11 datasets, both RecMotif and TreeMotif-DF require more than 10 hours to report results while TreeMotif-BF encounters memory shortage. The results indicate that although the algorithms have shown better overall accuracy than the other existing ones, there is still much room left for further improvement.

4.4 Summary

In this chapter, an algorithm RecMotif has been proposed. RecMotif is a recursive graph construction process. It requires a space complexity of $O(m^2n)$. As m and n are fixed for a certain dataset, RecMotif consumes almost constant space. Not only can this reduce execution time but also prevent memory shortage for discovering very weak motifs.

The efficiency of RecMotif in execution time has been compared against the other existing algorithms on synthetic and real datasets. RecMotif has shown better performance in handling MCP and several extended (l, d) -motif problems. As n or p increases, RecMotif consumes an execution time less than that of DPCFG and TreeMotif algorithms.

RecMotif and TreeMotif algorithms can discover all the possible motifs from an exact dataset, where each sequence contains at least one motif instance. When noisy sequences containing no motif instances are involved, these algorithms are not applicable. In the next chapter, extended algorithms, nRecMotif and nTreeMotif, will be introduced with the capability of discovering motifs from noisy datasets.

Chapter 5

Improved Graph-based Algorithms for WMD in Noisy Datasets

In Chapters 3 and 4, we have presented novel algorithms for discovering weak motifs from exact datasets, where each sequence contains at least one motif instance. However, a more realistic scenario is that a subset of sequences in a dataset contains no motif instances. That is, datasets may be corrupted with noisy sequences. In this case, the existing algorithms deliver decreased discovery accuracy or increased execution time.

In this chapter, we will present improved algorithms, nTreeMotif and nRecMotif, for discovering weak motifs from noisy datasets. nTreeMotif relies on the tree representation of motif instances, and nRecMotif relies on the graph representation. Experiments will show that they are more effective than the other existing algorithms. For instance, nRecMotif can find (60, 21)-motifs in *ArcA* datasets, which has never been reported previously.

5.1 Introduction

Through gene expression analysis or comparative genomics, biologists can assemble genes that show similar expression patterns or functions. Genes of the same set can share a regulatory mechanism if they have high similarity [6]. However, if a subset of genes

exhibit low similarity to the rest, they may have little in common with the latter [164]. That is, sequences regulating these genes may contain different motifs. Even possibly, several of the collected regulatory sequences may not contain motifs at all [11]. Under these circumstances, we refer to the dataset (of regulatory sequences) as noisy.

Given a noisy dataset, the basic weak (l, d) -motif discovery is extended to the (l, d) - k -motif discovery. The new extension aims to find motif instances from k of m sequences (with *quorum* $k < m$) [149]. It is more realistic than the basic problem which assumes the presence of motifs in all sequences [165]. Solving the extension can reduce the heavy burden of eliminating noisy sequences by using time-consuming and expensive wet-lab experiments. However, complex computational strategies are required to cope with noisy sequences. The extension is even more challenging than the basic problem.

Researchers have proposed several probabilistic algorithms to tackle the extension [13, 74, 98, 154, 165, 172]. However, these algorithms, as well as MEME [9] and Gibbs Motif Sampler [93], are incapable of guaranteeing the completeness of discovery due to lack of exhaustiveness [164]. In this regard, exact algorithms are superior.

One sample-driven algorithm [164] first clusters l' -mer ($l' \leq l$) substrings of the input sequences as a number of cliques of minimum size k . In each clique, any pair of substrings must have Hamming distance not greater than $2d$. Then, cliques of l' -mers are merged into cliques of l -mers, which denote the final motifs. Tests show that it can find all the cliques of instances of a $(15, 4)$ -15-motif in 20 sequences, but it requires long execution time of around 13 hours [164]. As with the other graph-based algorithms, the efficiency of this algorithm is sensitive to the edge density.

Algorithm GWM2 [67] improves algorithm DPCFG [198] for discovering weak motifs from noisy datasets. Both of them find cliques of motif instances in initialized graphs. DPCFG constructs cliques of a certain size (say x , $x < m$) at each vertex of a graph. More generally, GWM2 constructs cliques of sizes 2 to x at each vertex of the graph. As a

result, GWM2 requires much more space than DPCFG. Meanwhile, the time complexity of GWM2 is that of DPCFG multiplied by a $\binom{m}{k}$ factor in the worst case. As shown in Chapter 3, DPCFG has limitations in discovering very weak motifs. The degraded efficiency of GWM2 may affect its application.

Algorithm exVote is improved from algorithm Voting [36, 96]. It considers each of the 4^l candidate patterns as a true motif if the candidate has at least k instances in a (noisy) dataset. ExVote associates all the candidates in a table according to their l' -mer prefixes. This leads to a theoretical space complexity $O(4^{l-l'})$, which is reduced but still unacceptable for long motifs. In addition, although exVote aimed to handle noisy datasets, the authors did not provide any relevant experimental comparisons.

Pattern-driven algorithms such as the PMS-series generate d -neighbors as candidate motifs for each l -mer substring of the reference sequence(s), and they examine each d -neighbor by using the remaining sequences [43, 44, 136]. PMSprune uses a tree structure for exploring d -neighbors and a branch-and-bound strategy for pruning, leading to the best efficiency in time and space. By referring to $m-k+1$ sequences, qPMSprune enhances PMSprune for discovering motifs from noisy datasets. Algorithm iTriplet handles noisy datasets by a similar generating-and-validating process, but the candidate motifs are generated by using triple l -mers from different sequences [66].

In general, pattern-driven algorithms are efficient for dealing with short motifs, and they are easily adaptable to noisy datasets. However, they require exponentially increased execution time for detecting long motifs. In addition, they can be limited in applications because they require the exact expression of a motif [134]. In reality, it might be unreasonable (or even impossible) to determine an exact consensus when the instances of a potential motif are extremely weak or insufficient [97, 148].

In summary, the existing algorithms for weak motif discovery in noisy datasets either deliver low discovery accuracy or incur long execution time, requiring improvements.

This chapter extends TreeMotif-BF/RecMotif to nTreeMotif/nRecMotif for (l, d) - k -motif discovery. The main feature of them is a novel strategy for selecting reference sequences and another strategy for determining noisy sequences. The former guarantees that at least one sequence containing motif instances is selected as the reference to extend the cliques. The latter accelerates the selection of reference sequences and the recursive construction of trees/graphs. nTreeMotif and nRecMotif preserve the efficiency of TreeMotif-BF and RecMotif in execution time or space for exact datasets. Moreover, they can efficiently discover all the (l, d) - k -motifs from noisy datasets.

5.2 Strategies for handling noisy datasets

Suppose $s = \{s_t\}_{t=1}^m$ is a set of m sequences, where $s_t = (s_{t,j})_{j=1}^n$ denotes the t^{th} sequences of length n with each symbol $s_{t,j} \in \Omega$. Let $v_t = \{v_{t,j}\}_{j=1}^{n-l+1}$ be the set of l -mer substrings of sequences s_t , where $v_{t,j} = (s_{t,j'})_{j'=j}^{j+l-1}$ denotes an l -mer substring starting at site j of s_t . Assuming s satisfies the ZOOPS constraint, we are interested in detecting (l, d) - k motifs.

5.2.1 Selection of reference sequences

A reference sequence is believed to contain at least one true instance of a motif, which provides specific information in finding other instances [134]. For a noisy dataset, $m-k$ sequences do not contain any true instances. That is, there is at least one of $m-k+1$ different sequences containing a true instance. Therefore, nTreeMotif and nRecMotif use $m-k+1$ sequences as the reference in turn for each depth of tree construction or each recursion of graph construction. This differs from the existing algorithms in the use of reference sequences [66, 67, 134, 198].

Figure 5.1 illustrates an $(m-k+1)$ -ary tree of reference, of which each node represents a candidate reference sequence. The figure also shows the relationship between the depth

of the tree of reference (ref-depth in short), the depth of trees of motif instances (for nTreeMotif), and the recursion of graphs of motif instances (for nRecMotif).

nTreeMotif/nRecMotif explores the tree of reference in a depth-first manner. A sequence that is being used as the reference in ref-depth 1 to $r-1$ ($r \leq k$) will be excluded for testing as the reference in ref-depth r . This guarantees that all sequences are non-repeated for tree/graph construction. For example, if s_{m-k+1} is being used as the reference in ref-depth 2, node s_{m-k+1} (NOTE 1) in ref-depth 3 (and the subtree rooted at it) will be pruned.

Figure 5.1: Tree of reference sequences

Note* annotations on the right are about the tree of reference; annotations on the left are correspondingly about trees or graphs of motif instances, for the convenience of descriptions on algorithms nTreeMotif and nRecMotif.

Moreover, any subtree rooted at node s_r , whose parent node is s_i , can be excluded for testing as the reference for the tree/graph construction if $r < i$. For example, for

the reference node s_{m-k+1} in ref-depth 2, s_3 to s_{m-k+3} are the corresponding candidate references in ref-depth 3. In this case, it is only necessary to test s_{m-k+2} to s_{m-k+3} . This strategy can prune the tree of reference effectively. It is feasible because the order of sequences will not influence the formation of all the target cliques, if all the sequences contain true motif instances.

For example, suppose that s_1 , s_3 and s_{m-k+1} in Figure 5.1 are the sequences containing true motif instances. The path of ' $s_1, s_3, s_{m-k+1}, \dots$ ' must appear before the path of ' $s_1, s_{m-k+1}, s_3, \dots$ ' in the tree of reference, as the tree is traversed in a depth-first manner. After testing the former, nTreeMotif/nRecMotif can thoroughly discover the cliques of motif instances concerning s_1 , s_3 and s_{m-k+1} . Therefore, there is no need to construct cliques according to the latter path. In general, for s_1 , $\{s_j\}_{j=4}^{m-k}$ and s_{m-k+1} , as nTreeMotif/nRecMotif tests ' $s_1, s_j, s_{m-k+1}, \dots$ ' before ' $s_1, s_{m-k+1}, s_j, \dots$ ', there is no need to test the reference paths of ' $\{s_1, s_{m-k+1}, s_j, \dots\}_{j=4}^{m-k}$ '.

Note that the reference paths of ' $s_1, s_{m-k+1}, \dots$ ' might fail to form any k -clique with sequence s_3 excluded. However, as discussed, since all the possible cliques involving s_1 , s_3 and s_{m-k+1} have been formed, this failure will not miss any target cliques. On the other hand, if there is any additional clique formed without s_3 , then the proposition that s_3 is a sequence containing a true motif instance becomes questionable. In this case, all the additional cliques must also be considered and reported.

The preceding cases are under the assumption that sequence s_3 contains a true motif instance. Now, assuming s_3 is a noisy sequence, it is obvious that the exclusion of it from the tree/graph construction will not influence the discovery of the target cliques of true motif instances.

5.2.2 Determination of noisy sequences

The time for traversing the tree of reference is $O((m-k+1)^k)$ (in the worst case). Pruning the tree can reduce the traversal time by using the strategies discussed in the preced-

ing section. Moreover, the number of reference sequences can be further decreased by determining noisy sequences.

The sequences in $\{s_r\}_{r=i}^{m-k+i}$ are tested as the reference one by one in each recursion i of tree/graph construction. Before changing the reference sequence from s_r to s_{r+1} ($r \in [i, m-k+i-1]$), nTreeMotif/nRecMotif checks if any k -clique has ever been formed by using any reference vertex in s_r . Specifically, if at least one k -clique has ever been formed, then whether s_r is a noisy sequence or not is indeterminate (*Case 1*: the sequence is pending). Otherwise, s_r is determined as a noisy sequence for the reference sequences in recursions 1 to $i-1$ (*Case 2*: s_r is excluded from the future recursions).

For Case 1, all the possible cliques concerning s_r must have been formed, thus s_r can also be excluded from the further clique construction (although it cannot be determined as a noisy or a true-signal sequence). When the total number of sequences excluded (as pending and noisy) is larger than the maximum number of noisy sequences ($m-k$), the tree/graph construction process can be terminated immediately.

For convenience, let $S_{i,t}$ denote the status of sequence s_t in ref-depth i of the tree of reference, where $t \in [1, m]$ and $i \in [1, k]$. Specifically, $S_{i,t}$ can be *REFERRED*, *NOISY*, *PENDING* or *FREE*. *REFERRED* sequences for ref-depth i are the ones that are being used (as the reference) for the tree/graph construction in ref-depths 1 to i . *NOISY* sequences for ref-depth i are the ones that have been used (as the reference) for the tree/graph construction in ref-depths 1 to i while not any k -clique is formed by using any vertex in such sequences as the reference vertex. *PENDING* sequences for ref-depth i are the ones that have been used (as the reference) for the tree/graph construction in ref-depths 1 to i and at least one k -clique is formed by using a certain vertex in such sequences as the reference vertex. *FREE* sequences for ref-depth i are the ones that are not labeled as *REFERRED*, *PENDING* or *NOISY* in ref-depths 1 to i .

Let N_i denote the number of *NOISY* and *PENDING* sequences corresponding to ref-depths 1 to i . Initially, $N_0=0$. Before constructing trees/graphs in ref-depth i , nTreeMotif

and nRecMotif set $N_i = N_{i-1}$. Let $F_i = \text{TRUE}$ denote that at least one k -clique has ever been formed by using s_r as the reference in ref-depth i . Initially, $\{F_i = \text{FALSE}\}_{i=1}^k$.

Algorithm 12 iNoisyFreeSet(i, r): update status of a sequence s_r in ref-depth i

input : current status of reference $S_{i,r}$
current number of noisy sequences N_i
current clique indicator F_i .
output: updated $S_{i,r}$, N_i and F_i .
if $F_i = \text{TRUE}$ **then**
 $F_i \leftarrow \text{FALSE}$; $S_{i,r} \leftarrow \text{PENDING}$; $N_i \leftarrow N_i + 1$
else
 $S_{i,r} \leftarrow \text{NOISY}$; $N_i \leftarrow N_i + 1$
end if
return $S_{i,r}$, N_i and F_i

The status $S_{i,r}$ is updated by using *Algorithm 12*. Since at least k sequences contain instances of a motif, $m - k$ is the maximum number of sequences that can be excluded from the tree/graph construction. If $N_i > m - k$ after node s_r of ref-depth i is determined as NOISY/PENDING, there is no need to traverse the subtrees rooted at the other nodes with the same parent node as s_r .

In summary, the tree of reference is explored according to a simple branch-and-bound strategy. That is, $\forall i \in [1, k]$,

- if a node contains a free sequence and currently $N_i \leq m - k$, the sequence will be selected as a new reference, and there is a possibility to explore the subtree rooted at the node (*branch condition 1*);
- if a node contains a sequence being referenced or having been determined as noisy or pending in ref-depths 1 to $i - 1$, the sequence will not be selected as a new reference, and there is no need to explore the subtree rooted at the node. In this case, a sibling node of the current node will be tested (*branch condition 2*);

- if $N_i > m - k$ holds due to the determination of a sequence as noisy or pending, the exploring on reference sequences in ref-depth i stops (*bound condition*). In this case, the reference sequence is to be changed at ref-depth $i-1$.

This traverse of the tree of reference gives all C_k^m combinations of all the sequences. Thus, the time complexity for traversing tree has been decreased from $O((m - k + 1)^k)$ to $O(C_k^m)$. Combining with a recursive traversal of the tree of reference, nTreeMotif/n-RecMotif enhances the tree/graph construction in TreeMotif-BF/RecMotif. They can efficiently find all k -cliques of motif instances out of m sequences in a noisy dataset.

5.3 nTreeMotif algorithm

As with TreeMotif algorithms, nTreeMotif consists of graph representation and tree construction. The main difference: nTreeMotif selects reference sequences, determines noisy sequences, and constructs trees in a recursive manner. nTreeMotif also uses two reference sequences to obtain substrings of the other sequences for tree construction, referred to as graph representation. Thereafter, it initializes one-node trees according to a third reference sequence. Then, nTreeMotif recursively tests whether branches of a tree can be extended by a new node from another selected reference sequences, and it prunes the branches that are not appended by any new reference node. This will be referred to as its tree construction. A general description of the algorithm is given in Algorithm 13. Details about nTreeMotif are given in the following sections.

5.3.1 Graph representation

Suppose the first and the second reference sequences are $s_{r'}$ and $s_{r''}$, where $r' \in [1, m - k + 1]$ and $r'' \in [r' + 1, m - k + 2]$. The sequences for the further graph construction are $\{s_t\}_{t=r''+1}^m$. Suppose $\{P_{0,t=v_t}\}_{t=1}^m$ represents all the initial l -mer substrings of $\{s_t\}_{t=1}^m$. Algorithm 14

Algorithm 13 nTreeMotif (general description)

input: A set of m sequences $\{s_t\}_{t=1}^m$, quorum k , l and d .
output: ranked k -cliques
begin:
 build pairs of reference sequences (s_{r1}, s_{r2}) , where $r1$ and $r2$ are identifiers of reference sequences, s_{r1} in $[1, m-k+1]$, s_{r2} in $[2, m-k+2]$, and $r2 > r1$;
 for each pair of reference sequences (s_{r1}, s_{r2})
 build reference nodes pairs with l -mers of s_{r1} and s_{r2} ;
 for each reference node pair
 build a graph G using l -mers of $s_{r2+1}, s_{r2+2}, \dots, s_m$;
 for each vertex x (of G) in s_{r3} , where $r3$ in $[r2+1, m-k+3]$
 initialize a tree T rooted at vertex x ;
 initialize r as $r3$ and d' as 2;
 sub-function Append(d', T, r):
 if depth of T is equal to $k-2$
 record tree T with the reference node pair (where cliques are extracted and ranked);
 else
 for each $t \in [r+1, m-k+d'+2]$
 if s_t is not used in the current tree construction
 append vertices (of G) in s_t onto leafs of T ;
 end if
 If tree T is extended
 prune leafs (of tree T) whose depth is less than d' ;
 append($d'+1, T, r+1$):
 end if
 end for
 end if
 end sub-function
 end for
 end for
 end for
end

shows the graph construction for each reference node pair of $s_{r'}$ and $s_{r''}$. Nodes returned in $\{P_{1,t}\}_{t=r''+1}^m$ are candidates for constructing trees.

Algorithm 14 Node selection II

input : reference node pair $(v_{r',j'}, v_{r'',j''})$
 initial vertices $\{P_{0,t}\}_{t=r'}^m$
output: vertices $\{P_{1,t}\}_{t=r''+1}^m$ of graphs.
 $\{P_{1,t} \leftarrow \emptyset\}_{t=r''+1}^m$
for $\{t\}_{t=r''+1}^m$ **do**
 / suppose j is the starting site of a vertex to join the graph */*
 for $\{j\}_{j=1}^{n-l+1}$ **do**
 if $D_{\max}(v_{r',j'}, v_{r'',j''}, v_{t,j}) \leq 2d$ **then**
 $P_{1,t} \leftarrow P_{1,t} \cup \{v_{t,j}\}$
 end if
 end for
end for
return $\{P_{1,t}\}_{t=r''+1}^m$

5.3.2 Recursive tree construction

nTreeMotif initializes trees with nodes in $\{P_{1,t}\}_{t=r''+1}^{m-k+3}$, of which each is built up with nodes in $\{P_{1,t'}\}_{t'=t+1}^m$. Assuming the depth of the root of a tree (of motif instances) is 1, the depth d' of any node of the tree corresponds to the ref-depth $d'+2$ of the tree of reference. Once the depth of the tree grows to $k-2$, k -cliques (of motif instances) are formed (counting in the reference node pair of $s_{r'}$ and $s_{r''}$). Algorithm 15 shows the tree construction process.

The kernel of tree construction is a recursive function of *Append(depth d' , tree T_x , status $\{S_{d'+1,t}\}_{t=r+1}^m$, excluded $N_{d'+1}$)*, as shown in Algorithm 16. Supposing s_r is the reference sequence in depth $d'-1$ (≥ 1) of the tree of motif instances, the reference sequences for depth d' include $\{s_t\}_{t=r+1}^{m-k+d'+2}$. That is, this function checks if the tree can be further appended with nodes from sequences $\{s_t\}_{t=r+1}^{m-k+d'+2}$. If at least one of them can contribute at least one node to the tree, the function recurs to the next depth with the appended

tree. Once having backtracked from a sub-recursion, it updates the status of the current reference sequence and checks if the maximum number of sequences excluded from tree construction is greater than $m-k$. If true, the current sub-recursion will terminate and backtrack to its parent recursion. When the recursion terminates completely, for the current reference node pair, another reference sequence will be selected at the root level, and the tree construction process will be repeated.

Algorithm 15 Tree construction II

```

input : reference node pair  $(v_{r',j'}, v_{r'',j''})$ 
        vertices of a graph  $\{P_{1,t}\}_{t=r''+1}^m$ 
        status of sequences  $\{S_{2,t}\}_{t=r''+1}^m$ 
        number of noisy sequence  $N_2$ .
output: tree of motif instances. /* note: trees returned in function Append() */
 $\{S_{3,t} \leftarrow S_{2,t}\}_{t=r''+1}^m, N_3 \leftarrow N_2$ 
/* suppose  $r$  is the reference sequence in this depth of tree of motif instances */
for each  $\{s_r\}_{r=r''+1}^{m-k+3}$  do
    if  $S_{3,r}$  is FREE then
         $S_{3,r} \leftarrow$  REFERRED
        for each node  $v_{r,x} \in P_{1,r}$  do
             $v_{r,x} = \text{root}(T_x)$ 
            Append(depth 2, tree  $T_x$ , status  $\{S_{3,t}\}_{t=r+1}^m$ , excluded  $N_3$ )
        end for
        iNoisyFreeSet(3,  $r$ )
        if  $N_3 > m-k$  then
            break
        end if
    end if
end for

```

Once a tree of motif instances is formed, the motif refinement can be carried out as discussed in Section 3.2.3 of Chapter 3. Before the graph/tree construction starts, the nodes for graph/tree construction, the statuses of all the sequences, and the number of noisy sequences are initialized as: $\{P_{0,t}=v_t, S_{0,t}=\text{FREE}\}_{t=1}^m$, and $N_0=0$. The complete tree construction algorithm is shown in Algorithm 17.

Algorithm 16 Append(depth d' , tree T_x , status $\{S_{d'+1,t}\}_{t=r+1}^m$, excluded $N_{d'+1}$)

/ note: suppose the reference sequence is s_r for depth $d'-1$ of the tree of reference */*
input : an intermediate tree T_x
 status of sequence $\{S_{d'+1,t}\}_{t=r+1}^m$
 number of noisy sequences $N_{d'+1}$.
output: tree of motif instances.
 $\{S_{d'+2,t} \leftarrow S_{d'+1,t}\}_{t=r+1}^m, N_{d'+2} \leftarrow N_{d'+1}$
if depth(T_x) is $k-2$ **then**
 $\{F_i \leftarrow TURE\}_{i=1}^k$
 return T_x with $(v_{r',j'}, v_{r'',j''})$
else
 for $\{t\}_{t=r+1}^{m-k+d'+2}$ **do**
 if $S_{d'+2,t}$ is FREE **then**
 $S_{d'+2,t} \leftarrow$ REFERRED
 Flag $\leftarrow$ FALSE
 for each node $v_{t,j} \in P_{1,t}$ **do**
 / please refer to Chapter 3 for function extendable() */*
 if extendable($v_{t,j}, T_x$) **then**
 Append $v_{t,j}$ to T_x according to Q
 Flag $\leftarrow$ TRUE
 end if
 end for
 if Flag is TRUE **then**
 prune branches that are not extended
 Append(depth $d'+1$, tree T_x , status $\{S_{d'+2,t'}\}_{t'=t+1}^m$, excluded $N_{d'+2}$)
 end if
 iNoisyFreeSet($d' + 1, t$)
 if $N_{d'+2} > m-k$ **then**
 break
 end if
 end for
end if

Algorithm 17 nTreeMotif: finding motifs in noisy datasets

input : $\{P_{0,t}\}_{t=1}^m, \{S_{0,t}\}_{t=1}^m, N_0=0, l, d,$ and k .

output: trees of motif instances

$\{S_{1,t} \leftarrow S_{0,t}\}_{t=1}^m, N_1 \leftarrow N_0$.

```

for  $\{r'\}_{r'=1}^{m-k+1}$  do
  if  $S_{1,r'}$  is FREE then
     $S_{1,r'} \leftarrow$  REFERRED
     $\{S_{2,t} \leftarrow S_{1,t}\}_{t=r'+1}^m, N_2 \leftarrow N_1$ 
    for  $\{r''\}_{r''=r'+1}^{m-k+2}$  do
      if  $S_{2,r''}$  is FREE then
         $S_{2,r''} \leftarrow$  REFERRED
        Node selection II
        Tree Construction II
        iNoisyFreeSet(2,  $r''$ )
        if  $N_2 > m-k$  then
          break
        end if
      end if
    end for
    iNoisyFreeSet(1,  $r'$ )
    if  $N_1 > m-k$  then
      break
    end if
  end if
end for

```

5.4 nRecMotif algorithm

nRecMotif constructs a series of graphs recursively according to reference sequences, where the reference vertices on the recursive path represent a clique of motif instances. As both the graph construction and the traversal of the tree of reference are in a recursive manner, they can be easily combined. In each sub-recursion, nRecMotif selects a reference sequence, determines noisy sequences and constructs a graph for each vertex of the reference sequence. As the recursion goes deeper, the vertices remaining for the further graph construction decreases exponentially, which reduces execution time. A general description of the algorithm is given by Algorithm 18.

5.4.1 Recursive graph construction

5.4.1.1 Graph construction in recursion i

Let $G_i=(P_i, E_i)$ denote the graph in the i th round of recursion (or recursion i in short, $1 \leq i \leq k$), where $P_i=\{P_{i,t}\}_{t=1}^m$ is the set of sets of vertices and E_i is the set of edges (not necessarily to record). G_i is built by using the vertices of G_{i-1} according to a reference vertex in $P_{i-1,r}$. Therefore, $P_i \subseteq P_{i-1}$ always holds. Specifically, suppose in recursion i :

- s_r is the current reference sequence, i.e., the reference vertex $v_{r,j}$ is in $P_{i-1,r}$ and $S_{i,r}$ is REFERRED;
- $\{S_{i,t}=S_{i-1,t}, t \neq r\}_{t=1}^m$, i.e., the statuses of sequences have been passed from the parent recursion $i-1$;
- $\{P_{i,t}=\emptyset\}_{t=1}^m$, i.e., the vertex sets for G_i have been initialized as null.

For each reference vertex $v_{r,j}$, and for each $\{t\}_{t=r+1}^m$, if $S_{i,t}=\text{FREE}$, nRecMotif draws an edge between $v_{r,j}$ and each $v_{t,j'} \in P_{i-1,t}$ if $D(v_{r,j}, v_{t,j'}) \leq 2d$. Correspondingly, vertex $v_{t,j'}$ is collected in $P_{i,t}$. This leads to a graph G_i for each $v_{r,j}$.

Algorithm 18 nRecMotif (general description)

input: A set of m sequences $\{s_t\}_{t=1}^m$, quorum k , l and d .
output: ranked k -cliques
begin:
 initialize a graph G_0 with l -mers of s_t and insert vertices in sets $P_{0,t}$,
 where $t \in [1, m]$;
 initialize a null list L (for recording a clique);
 initialize reference identifier r' as 1 and recursion identifier i as 1;
 sub-function nRecMotif($\{P_{i-1,t}\}_{t=r'}^m, r'$):
 if recursion identifier i is less than k
 for each r in $[r', i+m-k]$
 for each vertex x of $P_{i-1,r}$
 set r'' as $r'+1$;
 build a graph G_i with the elements in $\{P_{i-1,t}\}_{t=r''}^m$ and
 insert vertices of G_i in sets $P_{i,t}$, where $t \in [r'', m]$;
 if the number of vertices in L plus the number of non-empty $P_{i,t}$
 is larger than or equal to $k-1$
 insert vertex x into L ;
 nRecMotif($\{P_{(i+1)-1,t}\}_{t=r''}^m, r''$);
 end if
 end for
 break if more than $m-k$ sequences cannot form k -cliques;
 end for
 else
 record l -mers in L (with rank according to information content);
 end if
 end sub-function
end

Now let us discuss the continuation condition for constructing graph G_i with a reference vertex $v_{r,j}$, which would be more general than that of RecMotif. For convenience, we say s_t is a *valid* sequence for $v_{r,j}$ if the corresponding $P_{i-1,t}$ contributes at least one vertex to $P_{i,t}$. Suppose V is the total number of valid sequences for $v_{r,j}$ after $t-r$ sequences (from s_{r+1} to s_t) are checked for constructing G_i . Meanwhile, $m-t$ is the maximum number of the remaining FREE sequences (including sequences s_{t+1} to s_m).

If $V+i+m-t < k$, which indicates a k -clique cannot be formed, the construction on G_i for $v_{r,j}$ can be stopped. Otherwise, the construction of G_i can be continued with vertices in $P_{i-1,t+1}$ (if $\|P_{i-1,t+1}\| \neq 0$ and $S_{i,t+1} = \text{FREE}$; otherwise, check vertices in $P_{i-1,t+2}$). The construction of G_i is shown in *Algorithm 19*.

Algorithm 19 iGraphConstruction

```

input : graph  $G_{i-1}$  with  $\{P_{i-1,t}, S_{i,t}\}_{t=r}^m$ 
         reference vertex  $v_{r,j}$ .
output:  $V$  and graph  $G_i$  with  $\{P_{i,t}, S_{i,t}\}_{t=r+1}^m$ .
 $\{P_{i,t} \leftarrow \emptyset\}_{t=r+1}^m$ 
 $V \leftarrow 0$ 
for  $\{P_{i,t}\}_{t=r+1}^m$  do
    if  $S_{i,t}$  is FREE then
        for each  $v_{t,j'} \in P_{i-1,t}$  do
            if  $D(v_{r,j}, v_{t,j'}) \leq 2d$  then
                 $P_{i,t} \leftarrow P_{i,t} \cup \{v_{t,j'}\}$ 
            end if
        end for
        if  $\|P_{i,t}\| > 0$  then
             $V \leftarrow V + 1$ 
        end if
        if  $V + i + m - t < k$  then
            break
        end if
    end if
end for
return  $V$  and  $G_i$  with  $\{P_{i,t}, S_{i,t}\}_{t=r+1}^m$ 
    
```

5.4.1.2 Continuation and backtracking from recursion i

After constructing graph G_i in recursion i ($1 \leq i \leq k$), nRecMotif can continue to recursion $i+1$ or has to backtrack to recursion $i-1$ due to the following conditions.

First, let us discuss the continuation of a recursion. nRecMotif continues from recursion i to recursion $i+1$ under the condition that there are at least $V=k-i$ valid sequences for reference vertex $v_{r,j}$ (in $P_{i-1,r}$). In this case, there is a possibility to form k -cliques. Otherwise, nRecMotif replaces the reference vertex in recursion i with another one in $P_{i-1,r}$ (if any), and it checks if the recursion can be continued.

Specifically, if $V \geq k-i$, nRecMotif inserts $v_{r,j}$ into L and steps into recursion $i+1$. The reference sequence in recursion $i+1$ is selected from the tree of reference according to the branch-and-bound condition. If the recursion continues until reaches recursion $k+1$, the reference vertices on the recursive path are output as motif instances. Also, flags $\{F_i\}_{i=1}^k$ are set as TRUE. They indicate that the current reference sequences in all the (sub-)recursions can offer at least one reference vertex to form a recursive path of depth k .

On the other hand, if $V+i < k$, nRecMotif cannot step into recursion $i+1$ with reference vertex $v_{r,j}$. Under this condition, two cases are considered. Firstly, if there are any more reference vertices in $P_{i-1,r}$, each of them is tested with the same recursion continuation condition. Secondly, if there are no more reference vertices in $P_{i-1,r}$, nRecMotif changes $S_{i,r}$ from REFERRED to NOISY or PENDING according to Algorithm 12. Note that each time after Algorithm 12 is called, nRecMotif will check the branch-and-bound condition (in Section 5.2.2) for further selection of a reference sequence. If branch condition 1 is satisfied, nRecMotif begins testing the sequence as the reference by setting its status as REFERRED, and it checks the recursion continuation condition.

Now let us discuss two backtracking conditions of a recursion that is resulted from the change of reference sequences. Firstly, If Algorithm 12 returns with the fact that

more than $m-k$ sequences have been excluded, nRecMotif stops the current recursion and backtracks to recursion $i-1$. Meanwhile, as mentioned in Section 5.2.1, $\{s_r\}_{r=i}^{m-k+i}$ are the candidate references in recursion i . Therefore, secondly, after all of them are tested in turn, nRecMotif also backtracks to recursion $i-1$.

5.4.2 Initialization and motif refinement

Before recursion 1 starts, the vertices of the first graph, the first reference sequence, the statuses of all the sequences, and the number of noisy sequences are initialized as: $\{P_{0,t}=v_t, S_{0,t}=\text{FREE}\}_{t=1}^m$, $r'=1$ (referring s_1), and $N_0=0$. In addition, the flags $\{F_i=\text{FALSE}\}_{i=1}^k$. nRecMotif is shown in *Algorithm 20*.

Consensus motifs can be aligned from the vertices in L . Further scanning by using the consensus motif can be applied to incorporate all possible motif instances. This refinement can increase the sensitivity of discovery, although spurious instances might be included in the results [67]. However, a consensus motif may not always be determined by a given set of motif instances. In this case, the clique of k instances are recorded as a motif. This makes nRecMotif capable of handling datasets with inexact consensus motifs. An option of reporting exactly k motif instances or re-scanned motif instances has been incorporated in the implementation of nRecMotif.

5.4.3 nRecMotif: an example

nRecMotif and nTreeMotif discover weak motifs from noisy datasets by manipulating reference sequences. An example about how nRecMotif selects reference sequences is given below. Suppose there are sequences $\{s_i\}_{i=1}^5$ in the dataset s . One sequence s_2 is noisy containing no motif instances, i.e., $k=4$.

In the following, each capital letter represents an l -mer vertex. Assume $s_1: P_{0,1} = \{A, B, C\}$, $s_2: P_{0,2} = \{U, V\}$, $s_3: P_{0,3} = \{Z, E, F, G\}$, $s_4: P_{0,4} = \{H, I, J, K, L\}$,

Algorithm 20 $\text{nRecMotif}(\{P_{i-1,t}, S_{i-1,t}\}_{t=r'}^m, r', N_{i-1})$: finding motifs in noisy datasets

```

/* note: suppose  $r'$  is the 1st reference to test in this subrecursion */
input :  $\{P_{i-1,t}, S_{i-1,t}\}_{t=r'}^m, r', N_{i-1}, l, d$ , and  $k$ .
output: cliques of motif instances.
if  $i < k$  then
   $\{S_{i,t} \leftarrow S_{i-1,t}\}_{t=r'}^m$  /* inherit statuses */
   $N_i \leftarrow N_{i-1}$  /* inherit number of noises */
  for each  $\{r\}_{r=r'}^{i+m-k}$  do
    if  $S_{i,r}$  is FREE then
       $S_{i,r} \leftarrow$  REFERRED
      for each  $v_{r,j} \in P_{i,r}$  do
        iGraphConstruction /* construct graph */
        if  $V+i \geq k$  then
           $L_i \leftarrow v_{r,j}$ 
          for each  $\{r''\}_{r''=r'+1}^{m-k+i+1}$  do
            if  $S_{i,r''} =$  FREE then
              break /* find 1st reference */
            end if
          end for
           $\text{nRecMotif}(\{P_{i,t}, S_{i,t}\}_{t=r''}^m, r'', N_i)$ 
        end if
      end for
      iNoisyFreeSet( $i, r$ ) /* set  $N_i$  and  $S_{i,r}$  */
      if  $N_i > m-k$  then
        break
      end if
    end if
  end for
else
   $\{F_{j'} \leftarrow \text{TRUE}\}_{j'=1}^k$  /* indicate all references valid */
  return  $\{L_{j'}\}_{j'=1}^k$ 
end if

```

Figure 5.2: nRecMotif: graph for vertex A
Note* edges are shown with dotted lines

s_5 : $P_{0,5} = \{M, N, O, Q, R, T\}$. Moreover, let $A_{2d} = \{Z, E, G, H, J, M, O, R, T, U\}$ denote that vertex A has a Hamming distances less than or equal to $2d$ with vertices $Z, E, G, H, J, M, O, R, T, U$ and more than $2d$ with the other vertices not listed above. Similarly, assume $U_{2d} = \{A, C, F, H, J, O\}$; $Z_{2d} = \{A, H, J, M, T\}$; $C_{2d} = \{U\}$; $E_{2d} = \{A\}$; $F_{2d} = \{U\}$; $G_{2d} = \{A, T\}$; $H_{2d} = \{A, R, U, Z\}$; $J_{2d} = \{A, Z, M, T, U\}$; $M_{2d} = \{A, Z, J\}$; $O_{2d} = \{A, U\}$; $R_{2d} = \{A, H\}$; $T_{2d} = \{A, Z, G, J\}$.

According to the algorithm, the potential candidate reference sequences are $\{s_j\}_{j=i}^{i+1}$ for recursion i as $m-k=1$. That is, s_1 and s_2 for recursion 1; s_2 and s_3 for recursion 2; s_3 and s_4 for recursion 3; s_4 and s_5 for recursion 4.

In recursion 1, sequence s_1 is selected as the reference sequence, in which vertex A is selected as the reference vertex. A graph is constructed for vertex A according to the assumptions as shown in Figure 5.2.

All the remaining sequences besides s_1 contribute at least one vertex to the graph. That is, there is possibility to form cliques of size 4 by the current reference vertex together with the vertices in $\{s_j\}_{j=2}^5$. Thus the graph construction process has to be continued to recursion 2 (recursion 1 will be suspended temporarily).

In recursion 2, sequence s_2 is selected as the reference sequence, in which vertex U is firstly selected as the reference vertex. A graph is constructed for vertex U according to the assumptions on the Hamming distance of vertices, as shown in Figure 5.3. Two of the remaining three sequences contribute at least one vertex to the graph. There is a possibility to form cliques of size 4 by the two current reference vertices together with the vertices in s_4 and s_5 . Thus, the graph construction process will be continued to recursion 3 (recursion 2 will be suspended temporarily).

In recursion 3, sequence s_3 should have been the first reference sequence. However, as it contributes no vertex to the graph, it is excluded from consideration. Thus, the other candidate s_4 is selected as the reference sequence, from which vertex H is selected as the reference vertex. However, no graph can be constructed for vertex H as $D(H, O) > 2d$, as shown in Figure 5.4. Therefore, another candidate J is selected as the reference vertex. As $D(J, O) > 2d$, there is also no graph constructed for J .

Figure 5.3: nRecMotif: graph for vertex U

Figure 5.4: nRecMotif: graph for vertex H

At this point, all the candidate reference vertices in the current reference sequence s_4 have been tested. Moreover, no clique of size 4 has been formed. Thus, s_4 is determined as a noisy sequence for the reference vertices A and U . Note that when s_4 is determined as a noisy sequence, there is insufficient number of sequences to form cliques of size 4 with the current two reference vertices. Therefore, nRecMotif has to backtrack to the recursion 2.

When having backtracked to recursion 2, nRecMotif will test another reference vertex in s_2 . In the example, as there is no more candidate reference vertex, the current reference sequence s_2 is to be replaced by another. Note that as there is no clique of size 4 formed by using s_2 as the reference, s_2 is determined as noisy for the reference vertex A .

As there is still a possibility to form 4-clique after excluding s_2 , nRecMotif selects s_3 as the reference. Vertex Z is firstly selected as the reference vertex. A graph is constructed for vertex Z according to the assumptions, as shown in Figure 5.5. As shown in Section 4.2.4, cliques of size 4 can be formed as $\{A, Z, J, M\}$ and $\{A, Z, J, T\}$. Note that s_4 will be selected as the first reference sequence in recursion 3, as s_3 is being used as the reference in the previous recursion 2.

 Figure 5.5: nRecMotif: graph for vertex Z

After vertex Z is used as the reference, the other vertices E and G in s_3 will also be tested with the same process. When the test is finished for vertex G , all the candidate reference vertices in s_3 have been tested, and s_3 is to be replaced by another reference sequence. However, as s_3 is the last reference sequence to test in recursion 2, nRecMotif has to backtrack to recursion 1 and replace the current reference vertex A with another candidate vertex in s_1 .

Similar processes will be carried out for vertices B and C in s_1 . After all the vertices in s_1 have been tested as the reference vertex, there are cliques of size 4 formed, such as $\{A, Z, J, M\}$. Thus, the status of s_1 cannot be determined as NOISY but set as PENDING.

The alternative reference sequence s_2 will be selected in recursion 1. For the first reference vertex U of s_2 , a graph will be constructed according to the assumptions as shown in Figure 5.6. Note that according to the discussion in Section 5.2.1, after s_1 has been used as the reference sequence, all the cliques concerning s_1 should have been discovered. Therefore, s_1 will be excluded from graph construction by using another sequence as the reference sequence.

Figure 5.6: nRecMotif: another graph for vertex U
 Note* there is no need to connect U and A, C as s_1 has been excluded.

5.5 Time and space complexity of nTreeMotif and nRecMotif

As indicated in Section 5.2.2, nTreeMotif and nRecMotif try to find k -cliques from k sequences for C_k^m times, where $C_k^m = m! / (k!(m-k)!)$. Thus, the algorithms nTreeMotif and nRecMotif apply a factor C_k^m to the time complexities of TreeMotif-BF and RecMotif respectively. That is, the time complexity of nTreeMotif is $O(C_k^m n^k)$, and that of nRecMotif is also $O(C_k^m n^k)$.

The space complexity of nTreeMotif follows that of TreeMotif-BF. nTreeMotif constructs trees with a depth of $k-2$. Thus, its space complexity is $O(n^{k-3})$. For nRecMotif, for each reference node in recursion x ($x \in [1, k-1]$), the maximum number of nodes to record is $(m-x)(n-l+1)$. Thus, the total space requirement is $\sum_{x=1}^{k-1} (m-x)(n-l+1)$, leading to a space complexity of $O(kmn)$.

Table 5.1 shows the complexities of several algorithms. iTriplet and qPMSprune have better scalability on increasing $m-k$, i.e., the number of noisy sequences, than the others. However, when increasing l and d , their execution time increases much faster than the others. Different (l, d) -motifs can have comparable values of p . For example, $p \approx 0.057$

for both (15, 4)- and (18, 5)-motifs. The execution time of nTreeMotif, nRecMotif and GWM2 will not show large differences for such cases. nTreeMotif and nRecMotif have the same theoretical time complexity, but in practice nRecMotif has a lower algorithmic overhead than nTreeMotif. This will be supported by experiments.

Table 5.1: Time and space complexities of algorithms for handling noisy datasets

Algorithm	Complexity	
	Time	Space
iTriplet	$O(m(m-k)(m+k)n^3l^3d^2)$	$O(N(l, d))$
qPMSprune	$O(m(m-k)n^2N(l, d))$	$O(mn^2)$
GWM2	$O(C_k^m n^k)$	-
nRecMotif	$O(C_k^m n^k)$	$O(k^2n)$
nTreeMotif	$O(C_k^m n^k)$	$O(n^{k-3})$

Note* $N(l, d) = \sum_{x=0}^d C_x^l 3^x$; '-': not available.

In this worst case analysis, the time complexities of nTreeMotif and nRecMotif increases exponentially, which can limit their applications. That is, they need further improvements to become efficient in execution time.

5.6 Experimental results

5.6.1 Synthetic data

5.6.1.1 Impacts of increasing length of sequences - exact data

The improved algorithms nTreeMotif and nRecMotif are first tested on the exact datasets as described in Section 3.4.1 of Chapter 4. The comparisons of the execution time and nPC are shown in Figure 5.7 and Figure 5.8. Note that when $n=600$, it corresponds to the basic motif challenge problem [131].

Figure 5.7 shows that, nRecMotif exhibits comparable nPC to TreeMotif-BF, iTriplet, qPMSprune and GWM2. nTreeMotif, TreeMotif-DF and RecMotif shows a little higher nPC than the others when $n \leq 1500$ as they report cliques of instances of size m (OOPS). Thus, a clique possibly contains few false instances. For comparison, nRecMotif, iTriplet,

qPMSprune and GWM2, which deliver the same nPC thus shown in the same curve, report cliques of instances of size $m' \geq m$. This allows them to report more true instances ($nSn=1.0$) than the setting of OOPS. However, larger cliques can involve more false instances [67].

Figure 5.8 shows that nRecMotif/RecMotif consumes the lowest execution time for most cases, to which that of nTreeMotif/TreeMotif algorithms is comparable. These comparisons are similar to those in Section 3.4.1.2. When $n > 1400$, PatternBranching requires less execution time than nRecMotif, which grows almost linearly with sequence length. However, in such cases of long sequences, PatternBranching cannot get rid of false instances as shown by Figure 5.7. Meanwhile, qPMSprune requires less execution time than nTreeMotif and nRecMotif when $n > 1600$ and $n > 1800$ respectively. With strategies for handling noisy sequences, nTreeMotif, nRecMotif and GWM2 aim to deal with noisy data. While for exact datasets, nTreeMotif and nRecMotif are more efficient than GWM2. For example, when $n=1300$, the time consumed by nTreeMotif, nRecMotif and GWM2 is 32, 17 and 244 seconds respectively.

As with GWM2, nRecMotif has an enhancing strategy for motif refinement. It derives consensus motifs from motif instances and does local alignments to find all the possible motif instances. In this setting, when tuning background base distribution, nRecMotif can deliver the same results as GWM2, qPMSprune and iTriplet. When tuning parameters m , l , d or p , nTreeMotif and nRecMotif show comparable performances on exact datasets to TreeMotif-BF and RecMotif. Thus, such comparisons are omitted.

Overall, nTreeMotif and nRecMotif almost preserve the efficiency of TreeMotif-BF and RecMotif in execution time for exact datasets respectively. On the other hand, nTreeMotif and nRecMotif cannot find short but weak motifs due to the inhibiting execution time for dense graphs involving a large number of random l -mer strings.

Figure 5.7: Comparison: effect of increasing n on nPC on noisy datasets

Figure 5.8: Comparison: effect of increasing n on execution time on noisy datasets

5.6.1.2 Impacts of increasing number of noisy sequences - noisy data

In this section, we compare the algorithms on noisy datasets. A dataset of m sequences is created following the steps in Section 3.4.1. The difference is that a dataset has only k ($k < m$) sequences containing instances of a target (l, d) -motif. Specifically, a dataset contains a $(15, 4)$ -20-motif. The parameter n is fixed at 600, and the parameter m is increased from 21 to 40. For each m , the performance of each algorithm is averaged on ten datasets. The nPC and execution time are shown in Figure 5.9 and 5.10. For nTreeMotif, iTriplet and GWM2, we only show a part of the results that are produced in an execution time comparable to the other algorithms.

Figure 5.9 shows that the accuracy of all algorithms declines with the increase of noisy sequences, although to different degrees. Similar to the performance on exact datasets, probabilistic algorithms show low discovery accuracy. PatternBranching shows high nPC when the number of noisy sequences is small. However, when the number of noisy sequences becomes large, its performance declined quickly.

For deterministic algorithms, iTriplet shows lower nPC in several tests than the others, because it could fail to produce any motifs for certain datasets. Meanwhile, nRecMotif performs almost consistently before m reaching 40. For datasets of $m=40$, it shows decreased nPC, because false instances as conserved as true ones increase significantly. nTreeMotif can give comparable performance to nRecMotif when $m < 30$. However, when $m \geq 30$, the execution time of nTreeMotif becomes much larger than nRecMotif (and nPMSprune).

qPMSprune shows higher nPC than nRecMotif in several cases such as when $m=40$. This is because qPMSprune generates candidate motifs using a substring of a reference sequence and aligns each candidate with the remaining sequences. As a result, the reference sequence contributes exactly one instance to a clique, namely, the reference substring. If it is a true instance, fewer false instances are included in the clique, leading

Figure 5.9: Comparison: effect of increasing $m-k$ on nPC on noisy datasets

Figure 5.10: Comparison: effect of increasing $m-k$ on execution time on noisy datasets

to higher nPC. However, if it is a false instance, nSn will be less than 1.0, i.e., the true instance in the reference sequence is missed. If qPMSprune does local alignment with all the sequences, it should deliver the same nPC as nRecMotif.

Figure 5.9 shows that GWM2, nTreeMotif and nRecMotif deliver comparable nPC when the number of noisy sequences is less than or equal to 10. However, Figure 5.10 shows that GWM2 consumes a much larger amount of execution time than nTreeMotif and nRecMotif. For example, GWM2 consumes about 3399 seconds for $m=27$, while nRecMotif and nTreeMotif consume 78 and 783 seconds respectively. Besides, GWM2 suffers from high space requirement for large m , while nRecMotif and nTreeMotif do not have such an issue. Note that nPC of nTreeMotif is not as consistent as nRecMotif because it outputs k -cliques of motif instances.

Figure 5.9 and 5.10 indicate that, when $m>32$, qPMSprune achieves the best results (of highest or comparable nPC within the lowest execution time). However, nRecMotif is more scalable than qPMSprune for increasing the length of motifs, as shown in Table 5.2. Since the standard errors are small (up to %2 of mean), they are omitted here.

Table 5.2: Comparison: effect of increasing (l, d) on execution time on noisy datasets

$(l, d)-k$	n	m	nRecMotif	nTreeMotif	qPMSprune
(18, 5)-20	800	30	0.2	2.8	0.3
		40	3.3	-	2.5
(21, 6)-20	900	30	0.3	2.4	4.7
		40	3.4	-	-
(24, 7)-20	1100	30	0.4	3.8	-
		40	5.6	-	-

Note*: '-': more than 10 hours; time unit: hour.

As l and d (and n) become large, the execution time of nRecMotif increases slower than that of qPMSprune. Figure 5.10 shows that the execution time of qPMSprune and nRecMotif grows exponentially with $m-k$ for noisy data. However, as the base of the execution time of nRecMotif for a long motif is much less than that of qPMSprune, it

consumes less execution time for larger m . Tests on real biological datasets can further show that nRecMotif, as well as nTreeMotif, has better performance than qPMSprune.

5.6.2 Biological data

5.6.2.1 *LexA* dataset

In this section, we test the capability of nTreeMotif/nRecMotif on the *LexA* dataset [64], which is a typical noisy dataset.

Without excluding the two noisy sequences *himA* and *uvrC* and given $l=24$, $d=8$ and $k=16$ (OOPS), nTreeMotif and nRecMotif can discover the best matching motif as *xtactgtatataxaacagtaxat* in 10 seconds. This result is comparable to that of TreeMotif-BF or RecMotif and better than that of qPMSprune which consumes 216 seconds but finds no consensus.

Given $l=20$, $d=5$ and $k=14$ (ZOOPS), nTreeMotif and nRecMotif can discover the best matching motif as *tactgtatataxatacagta* within 1 second. Although qPMSprune and GWM2 also finish discovery quickly within one second, they report no motifs. They fail because the consensus of the motif has an inexact expression. Note that although GWM2 is sample-driven, it derives consensus motifs and carries out local alignments to refine motifs. Thus, if an exact consensus does not exist, it reports nothing.

5.6.2.2 Several datasets in ECRDB70B-X

ECRDB70B-X is constructed according to *Escherichia coli* K-12 TFBSs in RegulonDB database [63, 68]. Each sequence of all datasets is composed of three parts: a left margin (X -mer), TFBS (l -mer), and a right margin (X -mer). Thus, it consists of $2X+l$ bases. We test datasets with more than 10 sequences, of which each contains a TFBS with left/right margins as 100-mer, 400-mer or 800-mer respectively.

We discuss the discovery accuracy of the algorithms at the binding site level. If an overlap of at least one nucleotide is observed between a predicted TFBS and a target

TFBS, the prediction is referred to as a hit [68]. The larger the number of hits is, the better the discovery is (if the false positives are comparable) [68]. nTreeMotif and nRecMotif can deliver comparable accuracies on these datasets except for a few with very weak motifs. In the following, we only compare nRecMotif with other typical algorithms, as shown in Table 5.3.

The results suggests that discovering motifs from noisy datasets is practically useful. Taking *Fur* datasets as an example, each sequence contains one (very weak) instance of a 19-mer motif. When the sequence length is 219, for a (24, 8)-12 motif (OOPS), nRecMotif discovers 549,098,239 cliques of instances within nearly two hours, most of which are random discoveries. For a (19, 6)-12 motif (OOPS), it discovers 38,142,262 cliques of instances consuming about six minutes. For a (19, 5)-11 motif (ZOOPS), it discovers only 29 cliques of instances within one second.

nRecMotif delivers consistent accuracy for the preceding cases. This indicates that the execution time can be wasted on finding singular instances which exhibit low similarity to the others. Also, a large number of cliques containing random instances can be formed if attempting to find all instances simultaneously, because we have to set the number of mutation d large (relevant to length l). A few random cliques might show higher level of significance than the target one, and they decline the quality of the results of the algorithms. By considering sequences containing singular instances as noisy, we can possibly reduce the difficulty in separating false cliques. Note that singular instances might be further discovered through refinement by using known ones.

Table 5.3 also shows that nRecMotif and MEME deliver comparable accuracy when given short background sequences. For long background sequences, accuracy of MEME is not as consistent as nRecMotif. Meanwhile, qPMSprune shows lower accuracy than nRecMotif and MEME. One reason is that a number of motif instances may not converge to an exact consensus, for which qPMSprune reports no motifs. Another reason is that

Table 5.3: Comparison: performance of algorithms on ECRDB70B-X datasets

Dataset	(l, d) - k	n	m	(G+C)% seq/motif	No.-of-hits-in-1st-clique			Time
					A_1	A_2	A_3	
<i>ArcA</i>	(61, 22)-28	261	28	0.40/0.34	21	13	-	$6.1 \cdot 10^4$
	(60, 21)-21	861	21	0.47/0.34	12	10	-	$2.2 \cdot 10^4$
	(60, 21)-21	1661	21	0.49/0.34	12	1	-	$1.1 \cdot 10^6$
<i>FruR</i>	(15, 4)-10	214	10	0.41/0.40	10	10	-	$0.2 \cdot 10^0$
	(15, 4)-10	814	10	0.46/0.40	10	10	6	$1.4 \cdot 10^1$
	(18, 4)-8	1614	10	0.48/0.40	7	0	-	$2.1 \cdot 10^0$
<i>Fur</i>	(24, 8)-12	219	12	0.42/0.32	10	11	-	$6.9 \cdot 10^3$
	(19, 6)-12	219	12	0.42/0.32	10	10	-	$3.8 \cdot 10^2$
	(19, 5)-11	219	12	0.42/0.32	11	10	-	$0.1 \cdot 10^0$
	(19, 5)-11	819	12	0.49/0.32	11	9	-	$1.2 \cdot 10^0$
	(19, 5)-11	1619	11	0.51/0.32	8	9	-	$3.5 \cdot 10^0$
<i>FNR</i>	(23, 8)-39	222	39	0.40/0.33	10	18	-	$7.3 \cdot 10^4$
	(14, 3)-19	822	35	0.46/0.31	12	0	-	$2.2 \cdot 10^1$
	(14, 3)-20	1622	35	0.49/0.31	13	10	0	$2.2 \cdot 10^2$
<i>IHF</i>	(11, 3)-39	213	42	0.38/0.28	10	2	-	$1.2 \cdot 10^2$
	(14, 4)-36	813	36	0.46/0.28	5	0	-	$1.2 \cdot 10^4$
	(14, 3)-22	1613	36	0.48/0.28	9	0	-	$1.7 \cdot 10^2$
<i>Lrp</i>	(12, 2)-11	212	29	0.39/0.37	4	2	0	$0.4 \cdot 10^0$
	(12, 2)-14	812	21	0.43/0.36	4	1	4	$1.8 \cdot 10^0$
	(12, 2)-14	1612	19	0.46/0.36	2	0	3	$4.9 \cdot 10^0$
<i>NarL</i>	(19, 5)-12	219	19	0.38/0.38	1	3	-	$1.0 \cdot 10^0$
	(23, 7)-16	219	19	0.38/0.38	2	7	-	$8.3 \cdot 10^0$
	(20, 5)-9	819	15	0.46/0.37	2	0	-	$3.3 \cdot 10^1$
	(19, 5)-13	1619	15	0.49/0.37	1	0	-	$2.9 \cdot 10^1$
	(25, 7)-12	1619	15	0.49/0.37	2	0	-	$5.0 \cdot 10^2$
<i>PurR</i>	(16, 4)-14	216	14	0.47/0.46	13	13	-	$0.2 \cdot 10^0$
	(16, 4)-14	816	14	0.50/0.46	13	13	-	$1.1 \cdot 10^0$
	(16, 4)-14	1616	14	0.52/0.46	12	12	-	$6.7 \cdot 10^0$
<i>SoxS</i>	(18, 6)-15	218	15	0.44/0.41	9	3	-	$6.9 \cdot 10^2$
	(18, 6)-15	818	15	0.49/0.41	-	0	0	unreasonable
	(18, 6)-14	1618	14	0.50/0.40	-	1	-	unreasonable

Note* A_1 : nRecMotif; A_2 : MEME; A_3 : qPMSprune; '-': no motifs found; time unit: second.

qPMSprune does not have a strategy for ranking reported motifs. An arbitrary selection of the first reported motif (if any) for comparison may lead to its low accuracy. We believe that a ranking strategy, such as by calculating an E -value of a motif instance or information content of a set of instances [65, 148], is necessary for weak motif discovery as there could be a large number of random motifs.

Moreover, nRecMotif has shown better performance than qPMSprune for discovering long weak motifs. Taking *ArcA* datasets as an example, nRecMotif is capable of finding

(60, 21)-motifs from 1661-mer sequences. Finding motifs successfully with the same setting has never been reported previously. On the other hand, tests on *SoxS* datasets show that there is still room to improve nRecMotif (and nTreeMotif) for handling short but very weak motifs hidden in long sequences.

5.7 Summary

In this chapter, we have presented two algorithms, nTreeMotif and nRecMotif, which can efficiently discover all the (l, d) - k motifs from a noisy dataset. nTreeMotif and nRecMotif improve TreeMotif-BF and RecMotif respectively by recursive tree/graph construction. The important feature of the two algorithms is that they select reference sequences and determine noisy sequences. To realize this, they arrange reference sequences in a tree and traverse it recursively with a branch-and-bound strategy.

Experimental results on exact synthetic datasets show that the enhanced algorithms have efficiency in execution time or space comparable to TreeMotif-BF and RecMotif. On noisy synthetic datasets, they have better scalability in terms of the number of noisy sequences and the length of motifs than the other existing algorithms. Moreover, tests on ECRDB70B-X datasets show that the new algorithms can deliver more consistent accuracy than MEME. Besides, they show stronger capability of discovering weak motifs that are long or have inexact consensus than pattern-driven algorithms such as qPMSprune.

Chapter 6

Summary and Future Work

Discovering motifs in regulatory genomic sequences can provide biologists with primary information for studying molecular processes, which helps reveal disease mechanisms and possibly lead to novel pertinent therapies. Nevertheless, it is prohibitive to discover motifs merely relying on manpower due to the substantial amount of data from various genome projects. Computational techniques can offer help to automate the task.

In this thesis, we have introduced the background and surveyed the state-of-the-art of (weak) motif discovery. Moreover, we have presented effective graph-based algorithms for discovering weak motifs from exact or noisy datasets. The content is summarized and discussed as follows.

6.1 Summary and discussion

Aiming to increase the discovery accuracy and algorithmic efficiency, this thesis proposes improved graph-based algorithms for discovering weak motifs from exact or noisy datasets. For either case, experimental comparisons show that they can deliver better performance than the widely used or existing algorithms.

TreeMotif-BF and TreeMotif-DF (TreeMotif algorithms) are tree-structured algorithms, which represent motif instances as nodes of trees based on the initial graphs of l -mer substrings of sequences. They show improvement in execution time compared to

the other algorithms such as DPCFG. However, the space consumption of TreeMotif-BF increases exponentially when it constructs trees in a breadth-first manner. Although TreeMotif-DF, which constructs trees in a depth-first manner, removes the space limitations, it is not as efficient as TreeMotif-BF in execution time.

RecMotif is a graph-based algorithm, which drastically improves space complexity compared to the existing algorithms. Although it does not improve theoretical time complexity, it discovers long weak motifs more efficiently than the existing algorithms. For example, it practically produces (40, 14)-motifs within 5 hours while others fail due to an unreasonable execution time or shortage of space. Meanwhile, for the (18, 6)-motif problem, which can be solved by PMSprune within 1 hour, RecMotif cannot produce motifs within a reasonable time. This indicates that there is room for further improvements.

Given exact datasets, RecMotif and TreeMotif algorithms can show increased accuracy and efficiency. However, they are unable to discover motifs from noisy datasets, which arise more often in reality. By introducing strategies for selecting reference sequences and determining noisy sequences, nTreeMotif and nRecMotif extend TreeMotif-BF and RecMotif to handle noisy datasets. Experiments on synthetic datasets show that nTreeMotif and nRecMotif are more scalable than the existing algorithms with regard to the number of noisy sequences, the length of motifs and the number of mutations. Experiments on real datasets show that nTreeMotif and nRecMotif are more accurate. Nevertheless, as with all the other existing algorithms, the performance of nTreeMotif/nRecMotif drops quickly when the number of noisy sequences increases.

The graph-based algorithms proposed in this thesis can be easily adapted to other motif finding problems such as those involving RNA or protein sequences. Meanwhile, they require further improvements in several aspects.

6.2 Future work

Based on the existing work and experimental results in this thesis, future work is discussed on three main points. Firstly, the significance of motif candidates can influence the discovery accuracy. Thus, how to calculate this metric is important. Secondly, it is necessary to improve the efficiency of algorithms in discovering short weak motifs or motifs in background sequences with biased base distributions. Thirdly, gapped motif discovery has not been studied as thoroughly as monad motif discovery, and more effective and efficient gWMD algorithms might be proposed based on the latter.

6.2.1 Significance of motifs

Exact algorithms produce all possible motifs to guarantee complete discovery. As a result, however, they need to handle a large number of false motifs besides the target one when discovering very weak motifs. How to filter false signals is noteworthy as they can easily prevent the true one from being reported [57, 71, 78].

For example, we rank and report motifs based on their information content [148]. Nevertheless, for very weak motifs such as a (16, 5)-motif, it has been observed that many false cliques have higher information content than the target one. As a result, they are reported instead of the target one, leading to low discovery accuracy.

Studies have shown that, the significance of motifs becomes more comprehensive by incorporating the statistical information of the distribution of bases in the background sequences [19]. The comprehensive significance can strongly discriminate whether a motif is a real over-represented one or just a random appearance over the sequences. Future effort will be needed to study metrics for measuring the significance of motifs.

6.2.2 Algorithmic efficiency

6.2.2.1 Short weak motif

For short weak motifs shown in Table 3.8, enhancing strategies have to be incorporated into the graph-based algorithms so that they can produce results within a satisfactory execution time. This could be carried out by using the following methods.

The new graph-based algorithms are good at discovering long motifs. Based on this, a possible approach is to extend the short target motif into a long one with their weakness p fixed at the same level. We then discover the long motif instead of the short target one. For example, a (18, 6)-motif and a (24, 8)-motif show the approximately equal p . The solutions (if any) for a (24, 8)-motif might show a possibility to include the solutions for a (18, 6)-motif. For this approach, how to extend the motif is the key point.

Another possible approach is based on the experimental observation. That is, if the number of sequences is small (say $m' < m$), the algorithms might be able to find m' -cliques quickly. Therefore, we could first divide a dataset of m sequences into m/m' sub-datasets by devising a proper m' . We could then apply algorithms on these sub-datasets respectively and merge m' -cliques as m -cliques. For this approach, how to calculate a proper m' and efficiently merge m' -cliques is the key point. Future effort will be needed to explore the feasibility of these ideas.

6.2.2.2 Background base distribution

Background base distribution can influence the performance of sample-driven algorithms, especially when the (G+C)-content is heavily biased, such as lower than 30 percent (or higher than 70 percent). As shown in Chapter 3, RecMotif cannot produce results within a reasonable execution time in such cases. This is also due to the increased similarity of sequences, which results in the same problem as discussed in Section 6.2.2.1. In this case, the approaches mentioned above might also be suitable for solving the problem.

Moreover, this problem is also associated with the problem in Section 6.2.1, as it involves a large number of false cliques of motif instances.

6.2.3 Gapped motif discovery

Gapped motif discovery will be a vital part of future work. In this study, algorithms for discovering dyad motifs from exact and noisy datasets can be firstly investigated. A simple method for handling such data is proposed [48]. That is, virtual sample (l_1+l_2) -mer substrings, which represent l_1 - and l_2 -mer components of dyad motifs, can be combined from real sample l_1 - and l_2 -mer substrings of the input sequences. Dyad motifs can be discovered by arranging the candidates in a suffix tree.

However, this method for data processing can result in a large number of virtual substrings that are determined by the maximum length of gaps between the components of instances of a dyad motif. As a result, a large amount of execution time or space is required to handle redundant candidate substrings. Taking the dyad motif in Table 2.2 as an example, because a maximum gap can be 23 bp, there can be 23 candidates for each remaining component. Therefore, it is crucial to find a way that can effectively eliminate candidate substrings in order to perform discovery efficiently.

Once a method has been proved to be feasible for discovering dyad motifs, it can be further extended to discover composite gapped motifs that have more than 2 sub-motifs. This is the general case of gapped motif discovery. A simple idea is that pairs of sub-motifs (dyad motifs) can be firstly discovered. On that basis, an additional procedure can be applied to assemble these sub-motifs. In summary, possible future work is shown in Table 6.1.

6.3 Conclusion

Weak motif discovery proves to be a very challenging problem in computational biology. Nevertheless, the new graph-based algorithms presented in this thesis have met the re-

search objective. All the new algorithms have shown improved accuracy and efficiency compared to the existing ones, especially for long motifs. However, future work is still needed to handle problems involving short weak motifs and gapped motifs, perhaps based on the approaches presented in this thesis.

Table 6.1: Future work

Tasks to investigate	Type of datasets	Type of motifs
Measure on the significance of motifs	Applicable for all cases	
Short weak motif problem	Exact/noisy	Monad
Dyad motif discovery problem		Gapped
Gapped motif discovery problem		

Appendices

Source codes of **TreeMotif** algorithms, **RecMotif**, **nTreeMotif** and **nRecMotif** can be downloaded from:

<https://www.dropbox.com/sh/q5nl2f3s50gq24p/rCMs-r3HmK>

Author's Publications

Journal: 2

- (i) **H.Q. Sun**, M.Y.H. Low, W.J. Hsu and J.C. Rajapakse. RecMotif: A Novel Fast Algorithm for Weak Motif Discovery. *BMC Bioinformatics* (Source: Proceedings of the Twenty-first International Conference on Genome Informatics (GIW2010)), 11(suppl.11), 2010.
- (ii) **H.Q. Sun**, M.Y.H. Low, W.J. Hsu, C.W. Tan and J.C. Rajapakse. Tree-structured Algorithms for Weak Motif Discovery. *Bioinformatics*, 27(19):2641-2647, 2011.

Conference: 1

- (i) **H.Q. Sun**, M.Y.H. Low, W.J. Hsu and J.C. Rajapakse. ListMotif: A Time and Memory Efficient Algorithm for Weak Motif Discovery. In *Proceedings of the 5th International Conference on Intelligent Systems and Knowledge Engineering*, pages 254-260. Hangzhou, China, Nov.15-16, 2010.

Other: 2

- (i) **H.Q. Sun**, W.J. Hsu, and J.C. Rajapakse. Improved Graph-based Algorithms for Weak Motif Discovery in Noisy Datasets. (Under preparation).
- (ii) A. Siau, X. Huang, X.Y. Yam, N.S. Bob, **H.Q. Sun**, J.C. Rajapakse, L. Renia and P.R. Preiser. Targeting Malaria Proteins into Host Erythrocyte: Identification of a New Export Signal. *Cellular Microbiology*, 16(5):673-86, 2014.

References

- [1] F. N. Abu-khzam, M. A. Langston, and P. Shanbhag. Scalable Parallel Algorithms for Difficult Combinatorial Problems: A Case Study in Optimization. In *International Conference on Parallel and Distributed Computing and Systems*, pages 563–568, 2003.
- [2] S. Aerts, P. Van Loo, Y. Moreau, and B. De Moor. A genetic algorithm for the detection of new cis-regulatory modules in sets of coregulated genes. *Bioinformatics*, 20(12):1974–1976, 2004.
- [3] S. Aerts, P. Van Loo, G. Thijs, Y. Moreau, and B. De Moor. Computational Detection of Cis-Regulatory Modules. *Bioinformatics*, 19(suppl. 2):ii5–14, 2003.
- [4] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *Proceedings of the Twentieth International Conference on Very Large Data Bases*, pages 487–499, 1994.
- [5] I. Al-Turaiki, G. Badr, and H. Mathkour. Trie-based Apriori Motif Discovery Approach. In *ISBRA*, volume 7292 of *Lecture Notes in Computer Science*, pages 1–12, 2012.
- [6] D. Allocco, I. Kohane, and A. Butte. Quantifying The Relationship between Co-Expression, Co-Regulation and Gene Function. *BMC Bioinformatics*, 5(1):1–18, 2004.
- [7] M. I. Arnone and E. H. Davidson. The Hardwiring of Development: Organization and Function of Genomic Regulatory Systems. *Development*, 124:1851–1864, 1997.

REFERENCES

- [8] K. B. Arvind. Bioinformatics in Microbial Biotechnology-a Mini Review. *Microbial Cell Factories*, 4(19):1–11, 2005.
- [9] T. L. Bailey and C. Elkan. Fitting a Mixture Model by Expectation Maximization to Discover Motifs in Biopolymers. In *Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology*, pages 28–36, 1994.
- [10] T. L. Bailey and C. Elkan. Unsupervised Learning Of Multiple Motifs In Biopolymers Using Expectation Maximization. *Machine Learning*, 21:51–80, 1995.
- [11] T. L. Bailey and M. Gribskov. The Megaprior Heuristic for Discovering Protein Sequence Patterns. In *Proceedings of the Fourth International Conference on Intelligent Systems for Molecular Biology*, pages 15–24, 1996.
- [12] T. L. Bailey and W. S. Noble. Searching for Statistically Significant Regulatory Modules. *Bioinformatics*, 19(suppl. 2):ii16–ii2, 2003.
- [13] Y. Barash, G. Bejerano, and N. Friedman. A Simple Hyper-Geometric Approach for Discovering Putative Transcription Factor Binding Sites. In *Proceedings of the First International Workshop on Algorithms in Bioinformatics*, pages 278–293, 2001.
- [14] Y. Barash, G. Elidan, N. Friedman, and T. Kaplan. Modeling Dependencies in Protein-DNA Binding Sites. In *Proceedings of the Seventh Annual International Conference on Research in Computational Molecular Biology*, pages 28–37, 2003.
- [15] B. P. Berman, Y. Nibu, B. D. Pfeiffer, P. Tomancak, S. E. Celniker, M. Levine, G. M. Rubin, and Mi. B. Eisen. Exploiting Transcription Factor Binding Site Clustering to Identify Cis-regulatory Modules Involved in Pattern Formation in the *Drosophila* Genome. *Proceedings of the National Academy of Sciences*, 99(2):757–762, 2002.
- [16] C. Bi. A Monte Carlo EM Algorithm for De Novo Motif Discovery in Biomolecular Sequences. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 6:370–386, 2009.

REFERENCES

- [17] M. Blanchette, B. Schiwikowski, and M. Tompa. Algorithms for Phylogenetic Footprinting. *Journal of Computational Biology*, 9(2):211–223, 2002.
- [18] M. Blanchette, B. Schwikowski, and M. Tompa. An Exact Algorithm to Identify Motifs in Orthologous Sequences from Multiple Species. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, pages 37–45, 2000.
- [19] M. Blanchette and S. Sinha. Separating Real Motifs from Their Artifacts. *Bioinformatics*, 17(suppl. 1):S30–S38, 2001.
- [20] M. Blanchette and M. Tompa. Discovery of Regulatory Elements by a Computational Method for Phylogenetic Footprinting. *Genome Research*, 12(5):739–748, 2002.
- [21] K. Blekas, D. I. Fotiadis, and A. Likas. Greedy Mixture Learning for Multiple Motif Discovery in Biological Sequences. *Bioinformatics*, 19(5):607–617, 2003.
- [22] S. Bouamama, A. Boukerram, and A. F. Al-Badarneh. Motif Finding using Ant Colony Optimization. In *Proceedings of the Seventh International Conference on Swarm Intelligence*, pages 464–471, 2010.
- [23] C. Boucher, D. Brown, and P. Church. A Graph Clustering Approach to Weak Motif Recognition. In *the Seventh Workshop on Algorithms in Bioinformatics*, Lecture Notes in Computer Science, pages 149–160, 2007.
- [24] A. Bråzma, I. Jonassen, I. Eidhammer, and D. Gilbert. Approaches to the Automatic Discovery of Patterns in Biosequences. *Journal of Computational Biology*, 5(2):279–305, 1998.
- [25] A. Bråzma, J. Vilo, E. Ukkonen, and K. Valtonen. Data Mining for Regulatory Elements in Yeast Genome. In *Proceedings of the Fifth International Conference on Intelligent Systems for Molecular Biology*, pages 65–74, 1997.
- [26] P. Šustar. Crick’s Notion of Genetic Information and the ‘Central Dogma’ of Molecular Biology. *The British Journal for the Philosophy of Science*, 58(1):13–24, 2007.

REFERENCES

- [27] J. Buhler and M. Tompa. Finding Motifs Using Random Projections. *Journal of Computational Biology*, 9(2):225–242, 2002.
- [28] M. L. Bulyk, P. L. F. Johnson, and G. M. Church. Nucleotides of Transcription Factor Binding Sites Exert Interdependent Effects on the Binding Affinities of Transcription Factors. *Nucleic Acids Research*, 30(5):1255–1261, 2002.
- [29] H. Bussemaker, H. Li, and E. D. Siggia. Building a Dictionary for Genomes: Identification of Presumptive Regulatory Sites by Statistical Analysis. *Proceedings of the National Academy of Sciences*, 97(18):10096–10100, 2000.
- [30] H. Bussemaker, H. Li, and E. D. Siggia. Regulatory Element Detection using Correlation with Expression. *Nature Genetics*, 27:167–171, 2001.
- [31] L. R. Cardon and G. D. Stormo. Expectation Maximization Algorithm for Identifying Protein-binding Sites with Variable Lengths from Unaligned DNA Fragments. *Journal of Molecular Biology*, 223(1):159–170, 1992.
- [32] A. M. Carvalho, A. T. Freitas, A. L. Oliveira, and M.-F. Sagot. An Efficient Algorithm for the Identification of Structured Motifs in DNA Promoter Sequences. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(2):126–140, 2006.
- [33] A. M. Carvalho and A. L. Oliveira. GRISOTTO: A Greedy Approach to Improve Combinatorial Algorithms for Motif Discovery with Prior Knowledge. *Algorithms for Molecular Biology*, 6:1–13, 2011.
- [34] X. Chen, L. Guo, Z. Fan, and T. Jiang. W-AlignACE: an Improved Gibbs Sampling Algorithm based on More Accurate Position Weight Matrices Learned from Sequence and Gene Expression/ChIP-chip Data. *Bioinformatics*, 24(9):1121–1128, 2008.
- [35] X. Chen and T. Jiang. An Improved Gibbs Sampling Method for Motif Discovery via Sequence Weighting. In *Computational Systems Bioinformatics*, pages 239–247, 2006.

REFERENCES

- [36] F. Y. L. Chin and H. C. M. Leung. Voting Algorithms for Discovering Long Motifs. In *the Third Asia-Pacific Bioinformatics Conference*, pages 261–271, 2005.
- [37] S.-J. Chou and E. Hermesz. Conserved Regulatory Elements Establish the Dynamic Expression of Rpx/HesI in Early Vertebrate Development. *Developmental Biology*, 292:533–545, 2006.
- [38] B. D. Chuong and B. Serafim. What Is the Expectation Maximization Algorithm? *Nature Biotechnology*, 26(8):897–899, 2008.
- [39] P. Collas. The Current State of Chromatin Immunoprecipitation. *Molecular Biotechnology*, 45(1):87–100, 2010.
- [40] E. M. Conlon, X. S. Liu, J. D. Lieb, and J. S. Liu. Integrating Regulatory Motif Discovery and Genome-wide Expression Analysis. *Proceedings of the National Academy of Sciences*, 100(6):3339–3344, 2003.
- [41] M. K. Das and H.-K. Dai. A Survey of DNA Motif Finding Algorithms. *BMC Bioinformatics*, 8(S21):1–13, 2007.
- [42] L. G. David, A. V. Miguel, A. G. Juan, and M. S. Juan. Finding Motifs in DNA Sequences Applying a Multiobjective Artificial Bee Colony (MOABC) Algorithm. In *Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*, volume 6623, pages 89–100. 2011.
- [43] J. Davila, S. Balla, and S. Rajasekaran. Space and Time Efficient Algorithms for Planted Motif Search. In *the Sixth International Conference on Computational Science / Second International Workshop on Bioinformatics Research and Applications*, pages 822–829, 2006.
- [44] J. Davila, S. Balla, and S. Rajasekaran. Fast and Practical Algorithms for Planted (l, d) Motif Search. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4(4):544–552, 2007.
- [45] R. E. Dickerson. DNA structure from A to Z. *Methods in Enzymology*, 211:67–111, 1992.

REFERENCES

- [46] C. E. Donnelly and W. S. Reznikoff. Mutations in the Lac P2 Promoter. *Journal of Bacteriology*, 169(5):1812–1817, 1987.
- [47] L. Duret and P. Bucher. Searching for Regulatory Elements in Human Noncoding Sequences. *Current Opinion in Structural Biology*, 7(3):399–406, 1997.
- [48] E. Eskin and P. A. Pevzner. Finding Composite Regulatory Patterns in DNA Sequences. *Bioinformatics*, 18(1):354–363, 2002.
- [49] P. Evans and A. Smith. Toward Optimal Motif Enumeration. In *Algorithms and Data Structures*, Lecture Notes in Computer Science, pages 47–58. Springer Berlin/Heidelberg, 2003.
- [50] F. Fauteux, M. Blanchette, and M. V. Stromvik. Seeder: Discriminative Seeding DNA Motif Discovery. *Bioinformatics*, 24(20):2303–2307, 2008.
- [51] A. V. Favorov, M. S. Gelfand, A. V. Gerasimova, D. A. Ravcheev, A. A. Mironov, and V. J. Makeev. A Gibbs Sampler for Identification of Symmetrically Structured, Spaced DNA Motifs with Improved Estimation of the Signal Length. *Bioinformatics*, 21(10):2240–2245, 2005.
- [52] C. Francis. Central Dogma of Molecular Biology. *Nature*, 227:561–563, 1970.
- [53] E. Fratkin, B. T. Naughton, D. L. Brutlag, and S. Batzoglou. MotifCut: Finding Regulatory Motifs with Maximum Density Subgraphs. *Bioinformatics*, 22(14):e150–157, 2006.
- [54] M. C. Frith, U. Hansen, and Z. Weng. Detection of Cis-element Clusters in Higher Eukaryotic DNA. *Bioinformatics*, 17(10):878–889, 2001.
- [55] M. C. Frith, M. C. Li, and Z. Weng. Cluster-Buster: Finding Dense Clusters of Motifs in DNA Sequences. *Nucleic Acids Research*, 31(13), 2003.
- [56] M. C. Frith, N. F. W. Saunders, B. Kobe, and T. L. Bailey. Discovering Sequence Motifs with Arbitrary Insertions and Deletions. *PLoS Computational Biology*, 4(5):e1000071, 2008.

REFERENCES

- [57] Martin C. Frith, John L. Spouge, Ulla Hansen, and Zhiping Weng. Statistical Significance of Clusters of Motifs Represented by Position Specific Scoring Matrices in Nucleotide Sequences. *Nucleic Acids Research*, 30(14):3214–3224, 2002.
- [58] D. J. Galas and A. Schmitz. DNase Footprinting: a Simple Method for the Detection of Protein-DNA Binding Specificity. *Nucleic acids research*, 5(9):3157–3170, 1978.
- [59] M. S. Gelfand, E. V. Koonin, and A. A. Mironov. Prediction of Transcription Regulatory Sites in Archaea by a Comparative Genomic Approach. *Nucleic Acids Research*, 28(3):695–705, 2000.
- [60] W. N. Grandy, T. L. Bailey, and C. P. Elkan. ParaMEME: A Parallel Implementation and a Web Interface for a DNA and Protein Motif Discovery Tool. *CABIOS*, 12(4):303–310, 1996.
- [61] D. GuhaThakurta and G. D. Stormo. Identifying Target Sites for Cooperatively Binding Factors. *Bioinformatics*, 17(7):608–621, 2001.
- [62] M. Gupta and J. S. Liu. Discovery of Conserved Sequence Patterns Using a Stochastic Dictionary Model. *Journal of the American Statistical Association*, 98:55–66, 2003.
- [63] S. Heladia, G.-C. Socorro, M.-A. Agustino, D.-P. Edgar, S.-S. Fabiola, P.-G. Martín, G.-A. Delfino, J.-J. Verónica, S.-Z. Alberto, B.-M. César, and C.-V. Julio. Regulondb (Version 4.0): Transcriptional Regulation, Operon Organization and Growth Conditions in *Escherichia Coli* K-12. *Nucleic Acids Research*, 32(suppl. 1):D303–D306, 2004.
- [64] G. Z. Hertz, G. W. Hartzell, and G. D. Stormo. Identification of Consensus Patterns in Unaligned DNA Sequences Known to Be Functionally Related. *Computer Applications in the Biosciences*, 6(2):81–92, 1990.
- [65] G. Z. Hertz and G. D. Stormo. Identifying DNA and Protein Patterns with Statistically Significant Alignments of Multiple Sequences. *Bioinformatics*, 15(7/8):563–577, 1999.

REFERENCES

- [66] E. S. Ho, C. D. Jakubowski, and S. I. Gunderson. iTriplet, a Rule-based Nucleic Acid Sequence Motif Finder. *Algorithms for Molecular Biology*, 4(14), 2009.
- [67] L. S. Ho and J. C. Rajapakse. Graphical Approach to Weak Motif Recognition in Noisy Data Sets. In *Workshop on Pattern Recognition in Bioinformatics (PRIB 2006)*, pages 23–31, 2006.
- [68] J. Hu, B. Li, and D. Kihara. Limitations and Potentials of Current Motif Discovery Algorithms. *Nucleic Acids Research*, 33(15):4899–4913, 2005.
- [69] Y.-J. Hu, S. Sandmeyer, C. McLaughlin, and D. Kibler. Combinatorial Motif Analysis and Hypothesis Generation on a Genomic Scale. *Bioinformatics*, 16(3):222–232, 2000.
- [70] Y.-J. Hu, S. Sandmeyer, C. McLaughlin, and D. Kibler. Finding Subtle Motifs with Variable Gaps in Unaligned DNA Sequences. *Computer Methods and Programs in Biomedicine*, 70(1):11–20, 2003.
- [71] J. D. Hughes, P. W. Estep, S. Tavazoie, and G. M. Church. Computational Identification of Cis-regulatory Elements Associated with Groups of Functionally Related Genes in *Saccharomyces Cerevisiae*. *Journal of Molecular Biology*, 296(5):1205–1214, 2000.
- [72] H. Huo, Z. Zhao, V. Stojkovic, and L. Liu. Optimizing genetic algorithm for motif discovery. *Mathematical and Computer Modelling*, 52(11-12):2011–2020, 2010.
- [73] V. Jackson. Studies on Histone Organization in the Nucleosome using Formaldehyde as a Reversible Cross-linking Agent. *Cell*, 15(3):945–954, 1978.
- [74] L. M. Jakt, L. Cao, K. S. Cheah, and D. K. Smith. Assessing Clusters and Motifs from Gene Expression Data. *Genome Research*, 11(1):112–123, 2001.
- [75] L. J. Jensen and S. Knudsen. Automatic Discovery of Regulatory Patterns in Promoter Regions based on Whole Cell Expression Data and Functional Annotation. *Bioinformatics*, 16(4):326–333, 2000.

REFERENCES

- [76] S. T. Jensen and J. S. Liu. BioOptimizer: a Bayesian Scoring Function Approach to Motif Discovery. *Bioinformatics*, 20(10):1557–1564, 2004.
- [77] C. Jia, R. Lu, and L. Chen. A Frequent Pattern Mining Method for Finding Planted (l, d) -Motifs of Unknown Length. In *Proceedings of the Fifth International Conference on Rough Set and Knowledge Technology*, pages 240–248, 2010.
- [78] Ö. Johansson, W. Alkema, W. W. Wasserman, and J. Lagergren. Identification of Functional Clusters of Transcription Factor Binding Motifs in Genome Sequences: the MSCAN Algorithm. *Bioinformatics*, 19(suppl. 1):i169–i176, 2003.
- [79] I. Jonassen. Efficient Discovery of Conserved Patterns using a Pattern Graph. *Computer Applications in the Biosciences: CABIOS*, 13(5):509–522, 1997.
- [80] N. C. Jones and P. A. Pevzner. *An Introduction to Bioinformatics Algorithms*. MIT Press, 2004.
- [81] S. Kazuhito. GibbsST: a Gibbs Sampling Method for Motif Discovery with Enhanced Resistance to Local Optima. *BMC Bioinformatics*, 7(486):1–18, 2006.
- [82] U. Keich and P. A. Pevzner. Finding Motifs in the Twilight Zone. *Bioinformatics*, 18(10):1374–1381, 2002.
- [83] U. Keich and P.A. Pevzner. Subtle Motifs: Defining the Limits of Motif Finding Algorithms. *Bioinformatics*, 18(10):1382–1390, 2002.
- [84] O. V. Kel-Margoulis, T. G. Ivanova, E. Wingender, and A. E. Kel. Automatic Annotation of Genomic Regulatory Sequences by Searching for Composite Clusters. In *Pacific Symposium on Biocomputing*, pages 187–198, 2002.
- [85] S. Keles, M. Laan, and M. Eisen. Identification of Regulatory Elements using a Feature Selection Method. *Bioinformatics*, 18(9):1167–1175, 2002.
- [86] O. D. King and F. P. Roth. A non-parametric model for transcription factor binding sites. *Nucleic Acids Research*, 31(19):e116, 2003.

REFERENCES

- [87] K. Klepper, G. K. Sandve, O. Abul, J. Johansen, and F. Drablos. Assessment of Composite Motif Discovery Methods. *BMC Bioinformatics*, 9(1):1–16, 2008.
- [88] A. Klingenhoff, K. Frech, K. Quandt, and T. Werner. Functional Promoter Modules Can Be Detected by Formal Models Independent of Overall Nucleotide Sequence Similarity. *Bioinformatics*, 15(3):180–186, 1999.
- [89] L. J. Korn, C. L. Queen, and M. N. Wegman. Computer Analysis of Nucleic Acid Regulatory Sequences. *Proceedings of the National Academy of Sciences*, 74(10):4401–4405, 1977.
- [90] I. Ladunga. An overview of the computational analyses and discovery of transcription factor binding sites. *Methods in molecular biology (Clifton, N.J.)*, 674:1–22, 2010.
- [91] E. Larsson, P. Lindahl, and P. Mostad. HeliCis: a DNA Motif Discovery Tool for Colocalized Motif Pairs with Periodic Spacing. *BMC Bioinformatics*, 8(418):1–9, 2007.
- [92] D. S. Latchman. Transcription Factors: Bound to Activate or Repress. *Trends in Biochemical Sciences*, 26(4):211–213, 2001.
- [93] C. E. Lawrence, S. F. Altschul, M. S. Boguski, J. S. Liu, A. F. Neuwald, and J. C. Wootton. Detecting Subtle Sequence Signals: a Gibbs Sampling Strategy for Multiple Alignment. *Science*, 262(5131):208–214, 1993.
- [94] C. E. Lawrence and A. A. Reilly. An Expectation Maximization Algorithm for the Identification and Characterization of Common Sites in Unaligned Biopolymer Sequences. *PROTEINS: Structure, Function, Genetics*, 7:41–51, 1990.
- [95] C. Lefèvre and J.-E. Ikeda. A Fast Word Search Algorithm for the Representation of Sequence Similarity in Genomic DNA. *Nucleic Acids Research*, 22(3):404–411, 1994.
- [96] H. C. M. Leung and F. Y. L. Chin. An Efficient Algorithm for the Extended (l , d)-Motif Problem with Unknown Number of Binding Sites. In *Proceedings of the Fifth IEEE Symposium on Bioinformatics and Bioengineering*, pages 11–18, 2005.

REFERENCES

- [97] H. C. M. Leung and F. Y. L. Chin. Finding Exact Optimal Motifs in Matrix Representation by Partitioning. *Bioinformatics*, 21(2):ii86–92, 2005.
- [98] H. C. M. Leung and F. Y. L. Chin. Finding motifs from all sequences with and without binding sites. *Bioinformatics*, 22(18):2217–2223, 2006.
- [99] H. Li and J. Li. Discovery of Stable and Significant Binding Motif Pairs from PDB Complexes and Protein Interaction Datasets. *Bioinformatics*, 21(3):314–324, 2005.
- [100] L. Li. GADEM: A Genetic Algorithm Guided Formation of Spaced Dyads Coupled with an EM Algorithm for Motif Discovery. *Journal of Computational Biology*, 16(2):317–329, 2009.
- [101] L. Li, R. L. Bass, and Y. Liang. fdrMotif: Identifying Cis-elements by an EM Algorithm Coupled with False Discovery Rate Control. *Bioinformatics*, 24(5):629–636, 2008.
- [102] M. Li, B. Ma, and L. Wang. Finding Similar Regions in Many Sequences. *Journal of Computer and System Sciences*, 65(1):73–96, 1999.
- [103] M. Li, B. Ma, and L. Wang. On the Closest String and Substring Problems. *Journal of the ACM*, 49(2):157–171, 2002.
- [104] S. Liang, M. P. Samanta, and B. A. Biegel. cWINNOWER Algorithm for Finding Fuzzy DNA Motifs. *Journal of Bioinformatics and Computational Biology*, 1(1):47–60, 2004.
- [105] J. S. Liu and C. E. Lawrence. Bayesian Inference on Biopolymer Models. *Bioinformatics*, 15(1):38–52, 1999.
- [106] X. Liu, D. L. Brutlag, and J. S. Liu. Bioprospector: Discovering Conserved DNA Motifs In Upstream Regulatory Regions Of Co-Expressed Genes. In *Pacific Symposium on Biocomputing*, pages 127–138, 2001.
- [107] X. S. Liu, D. L. Brutlag, and J. S. Liu. An Algorithm for Finding Protein-DNA Binding Sites with Applications to Chromatin Immunoprecipitation Microarray Experiments. *Nature Biotechnology*, 20(8):835–839, 2002.

REFERENCES

- [108] H. Lodish, A. Berk, S. L. Zipursky, P. Matsudaira, D. Baltimore, and J. Darnell. *Molecular Cell Biology*. Freeman, W. H. and Company, 5 edition, 2000.
- [109] N. M. Luscombe, D. Greenbaum, and M. Gerstein. What Is Bioinformatics? A Proposed Definition and Overview of the Field. *Methods of Information in Medicine*, 40(4):346–358, 2001.
- [110] X. Ma, A. Kulkarni, Z. Zhang, Z. Xuan, R. Serfling, and M. Q. Zhang. A Highly Efficient and Effective Motif Discovery Method for ChIP-seq/ChIP-chip Data using Positional Information. *Nucleic Acids Research*, 40(7):e50, 2012.
- [111] L. Marsan and M.-F. Sagot. Algorithms for Extracting Structured Motifs Using a Suffix Tree with an Application to Promoter and Regulatory Site Consensus Identification. *Journal of Computational Biology*, 7(3-4):345–362, 2000.
- [112] T. Marschall and S. Rahmann. Efficient Exact Motif Discovery. *Bioinformatics*, 25(12):i356–i364, 2009.
- [113] E. Martha. The Prion Challenge to the ‘Central Dogma’ of Molecular Biology, 1965-1991: Part I: Prelude to Prions. *Studies in History and Philosophy of Science*, 30(1):1–19, 1999.
- [114] L. A. McCue, W. Thompson, C. S. Carmack, M. P. Ryan, J. S. Liu, V. Derbyshire, and C. E. Lawrence. Phylogenetic Footprinting of Transcription Factor Binding Sites in Proteobacterial Genomes. *Nucleic Acids Research*, 29(3):774–782, 2001.
- [115] J. D. McKinney and N. Heintz. Transcriptional Regulation in the Eukaryotic Cell Cycle. *Trends in Biochemical Sciences*, 16:430–435, 1991.
- [116] M. L. Metzker. Emerging Technologies in DNA Sequencing. *Genome Research*, 15(12):1767–1776, 2005.
- [117] M. L. Metzker. Sequencing Technologies-the Next Generation. *Nature Reviews Genetics*, 11(1):31–46, 2010.

REFERENCES

- [118] T.-C. Morgane, H. Carl, D. Matthieu, S. Olivier, T. Denis, and van-H. Jacques. RSAT Peak-motifs: Motif analysis in Full-Size ChIP-seq Datasets. *Nucleic Acids Research*, pages 1–9, 2011.
- [119] D. W. Mount. *Bioinformatics-Sequence and Genome Analysis*. Cold Spring Harbor Laboratory Press, 1 edition, 2001.
- [120] M. E. Mulligan, D. K. Hawley, R. Entriken, and W. R. McClure. Escherichia Coli Promoter Sequences Predict in Vitro RNA Polymerase Selectivity. *Nucleic Acids Research*, 12(1):789–800, 1984.
- [121] G. J. Narlikar, H.-Y. Fan, and R. E. Kingston. Cooperation between Complexes that Regulate Chromatin Structure and Transcription. *Cell*, 108(4):475–487, 2002.
- [122] A. F. Neuwald, J. S. Liu, and C. E. Lawrence. Gibbs Motif Sampling: Detection of Bacterial Outer Membrane Protein Repeats. *Protein Science*, 4(8):1618–1632, 1995.
- [123] P. Ng and U. Keich. GIMSAN: a Gibbs Motif Finder with Significance Analysis. *Bioinformatics*, 24(19):2256–2257, 2008.
- [124] G. Orphanides and D. Reinberg. A Unified Theory of Gene Expression. *Cell*, 108(4):439–451, 2002.
- [125] P. J. Park, A. J. Butte, and I. S. Kohane. Comparing Expression Profiles of Genes with Similar Promoter Regions. *Bioinformatics*, 18(12):1576–1584, 2002.
- [126] T. K. Paul and H. Iba. Identification of Weak Motifs in Multiple Biological Sequences Using Genetic Algorithm. In *Genetic And Evolutionary Computation Conference-Proceedings of the Eighth Annual Conference on Genetic and Evolutionary Computation*, pages 271–278, 2006.
- [127] G. Pavesi, G. Mauri, and G. Pesole. An Algorithm for Finding Signals of Unknown Length in DNA Sequences. *Bioinformatics*, 17(suppl. 1):S207–214, 2001.

REFERENCES

- [128] G. Pavesi, P. Mereghetti, G. Mauri, and G. Pesole. Weeder Web: Discovery of Transcription Factor Binding Sites in a Set of Sequences from Co-Regulated Genes. *Nucleic Acids Research*, 32(suppl. 2):W199–203, 2004.
- [129] G. Pesole, N. Prunella, S. Liuni, M. Attimonelli, and C. Saccone. WORDUP: an Efficient Algorithm for Discovering Statistically Significant Patterns in DNA Sequences. *Nucleic Acids Research*, 20(11):2871–2875, 1992.
- [130] E. Pettersson, J. Lundeberg, and A. Ahmadian. Generations of Sequencing Technologies. *Genomics*, 93(2):105–111, 2009.
- [131] P. A. Pevzner and S.-H. Sze. Combinatorial Approaches to Finding Subtle Signals in DNA Sequences. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, pages 269–278, 2000.
- [132] N. Pisanti, A. M. Carvalho, L. Marsan, and M.-F. Sagot. RISOTTO: Fast Extraction of Motifs with Mismatches. In *Proceedings of the Seventh Latin American Theoretical Informatics Symposium*, Lecture Notes in Computer Science, pages 757–768, 2006.
- [133] Y. Pollack, A. L. Katzen, D. T. Spira, and J. Golenser. The Genome of Plasmodium Falciparum. I: DNA Base Composition. *Nucleic acids research*, 10(2):539–546, 1982.
- [134] A. Price, S. Ramabhadran, and P. A. Pevzner. Finding Subtle Motifs by Branching from Sample Strings. *Bioinformatics*, 19(suppl. 2):ii149–ii155, 2003.
- [135] Z. S. Qin, L. A. McCue, W. Thompson, L. Mayerhofer, C. E. Lawrence, and J. S. Liu. Identification of Co-regulated Genes Through Bayesian Clustering of Predicted Regulatory Binding Sites. *Nature Biotechnology*, 21(4):435–439, 2003.
- [136] S. Rajasekaran, S. Balla, and C. H. Huang. Exact Algorithms for Planted Motif Challenge Problems. In *Proceedings of the Third Asia-Pacific Bioinformatics Conference*, pages 249–259, 2005.

REFERENCES

- [137] S. Rajasekaran and H. Dinh. A Speedup Technique for (l, d) -motif Finding Algorithms. *BMC Research Notes*, 4(54):1–7, 2011.
- [138] B. Raphael, L-T. Liu, and G. Varghese. A Uniform Projection Method for Motif Discovery in DNA Sequences. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1(2):91–94, 2004.
- [139] M. Rebeiz, N. L. Reeves, and J. W. Posakony. SCORE: A Computational Approach to the Identification of Cis-regulatory Modules and Target Genes in Whole-genome Sequence Data. *Proceedings of the National Academy of Sciences*, 99(15):9888–9893, 2002.
- [140] T. Reichhardt. It’s Sink or Swim as a Tidal Wave of Data Approaches. *Nature*, 399:517–520, 1999.
- [141] I. Rigoutsos and A. Floratos. Combinatorial Pattern Discovery in Biological Sequences: the TEIRESIAS Algorithm. *Bioinformatics*, 14(1):55–67, 1998.
- [142] E. Rocke and M. Tompa. An Algorithm for Finding Novel Gapped Motifs in DNA Sequences. In *Proceedings of the Second Annual International Conference on Computational Molecular Biology*, pages 228–233, 1998.
- [143] F. P. Roth, J. D. Hughes, P. W. Estep, and G. M. Church. Finding DNA Regulatory Motifs within Unaligned Non-Coding Sequences Clustered by Whole-Genome mRNA Quantitation. *Nature Biotechnology*, 16(10):939–945, 1998.
- [144] M.-F. Sagot. Spelling Approximate Repeated or Common Motifs Using a Suffix Tree. In *Theoretical Informatics: Third Latin American Symposium (LATIN’98)*., pages 374–390, Brazil, 1998.
- [145] G. K. Sandve, O. Abul, V. Walseng, and F. Drabløs. Improved Benchmarks for Computational Motif Discovery. *BMC Bioinformatics*, 8(193):1–13, 2007.
- [146] G. K. Sandve and F. Drabløs. A Survey of Motif Discovery Methods in an Integrated Framework. *Biology Direct*, 1(11):1–16, 2006.

REFERENCES

- [147] M. Scherf, A. Klingenhoff, and T. Werner. Highly Specific Localization of Promoter Regions in Large Genomic Sequences by PromoterInspector: a Novel Context Analysis Approach. *Journal of Molecular Biology*, 297(3):599–606, 2000.
- [148] T. D. Schneider and R. M. Stephens. Sequence Logos: a New Way to Display Consensus Sequences. *Nucleic Acids Research*, 18(20):6097–6100, 1990.
- [149] E. Segal, Y. Barash, I. Simon, N. Friedman, and D. Koller. From Promoter Sequence to Expression: a Probabilistic Framework. In *Proceedings of the Sixth Annual International Conference on Computational Biology*, pages 263–272, 2002.
- [150] E. Segal and R. Sharan. A Discriminative Model for Identifying Spatial cis-Regulatory Modules. In *Proceedings of the Eighth Annual International Conference on Research in Computational Molecular Biology*, pages 141–149, 2004.
- [151] R. Sharan, I. Ovcharenko, A. Ben-Hur, and R. M. Karp. CREME: a Framework for Identifying cis-Regulatory Modules in Human-Mouse Conserved Segments. *Bioinformatics*, 19(suppl. 1):i283–i291, 2003.
- [152] A. A. Sharov and M. S. H. Ko. Exhaustive Search for Over-represented DNA Sequence Motifs with CisFinder. *DNA Research*, pages 261–273, 2009.
- [153] R. Siddharthan, E. D. Siggia, and E. van Nimwegen. PhyloGibbs: a Gibbs Sampling Motif Finder That Incorporates Phylogeny. *PLoS Computational Biology*, 1(7):e67, 2005.
- [154] S. Sinha. Discriminative Motifs. *Journal of Computational Biology*, 10(3-4):599–615, 2003.
- [155] S. Sinha, Y. Liang, and E. Siggia. Stubb: a Program for Discovery and Analysis of Cis-Regulatory Modules. *Nucleic Acids Research*, Web Server Issue(34):w555–559, 2006.
- [156] S. Sinha and M. Tompa. A Statistical Method for Finding Transcription Factor Binding Sites. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, pages 344–354, 2000.

REFERENCES

- [157] S. Sinha and M. Tompa. Discovery of Novel Transcription Factor Binding Sites by Statistical Overrepresentation. *Nucleic Acids Research*, 30(24):5549–5560, 2002.
- [158] S. Sinha and M. Tompa. YMF: a Program for Discovery of Novel Transcription Factor Binding Sites by Statistical Overrepresentation. *Nucleic Acids Research*, 31(13):3586–3588, 2003.
- [159] S. Sinha, E. van Nimwegen, and E. D. Siggia. A Probabilistic Method to Detect Regulatory Modules. *Bioinformatics*, 19(suppl 1):i292–i301, 2003.
- [160] G. D. Stormo. DNA Binding Sites: Representation and Discovery. *Bioinformatics*, 16(1):16–23, 2000.
- [161] G. D. Stormo. Motif Discovery using Expectation Maximization and Gibbs’ Sampling. *Methods in Molecular Biology*, 674:85–95, 2010.
- [162] G. D. Stormo and G. W. Hartzell. Identifying Protein-binding Sites from Unaligned DNA Fragments. *Proceedings of the National Academy of Sciences of the United States of America*, pages 1183–1187, 1989.
- [163] G. D. Stormo, T. D. Schneider, L. Gold, and A. Ehrenfeucht. Use of the ‘Perceptron’ Algorithm to Distinguish Translational Initiation Sites in *E. coli*. *Nucleic Acids Research*, 10(9):2997–3011, 1982.
- [164] M. P. Styczynski, K. L. Jensen, I. Rigoutsos, and G. N. Stephanopoulos. An Extension and Novel Solution to the (l, d) -Motif Challenge Problem. *Genome Informatics*, 15(2):63–71, 2004.
- [165] S.-H. Sze, M. S. Gelfand, and P. A. Pevzner. Finding Weak Motifs in DNA Sequences. In *Pacific Symposium on Biocomputing*, pages 235–246, 2002.
- [166] S.-H. Sze, S. Lu, and J. Chen. Integrating Sample-Driven and Pattern-Driven Approaches in Motif Finding. In *Algorithms in Bioinformatics*, volume 3240 of *Lecture Notes in Computer Science*, pages 438–449. Springer Berlin/Heidelberg, 2004.

REFERENCES

- [167] S.-H. Sze and X. Zhao. Improved Pattern-Driven Algorithms for Motif Finding in DNA Sequences. In *Proceedings of the 2005 Joint RECOMB Satellite Workshops on Systems Biology and Regulatory Genomics, Lecture Notes in Bioinformatics*, pages 198–211, 2006.
- [168] D. G. Thakurta. Computational Identification of Transcriptional Regulatory Elements in DNA Sequence. *Nucleic Acids Research*, 34:3585–3598, 2006.
- [169] D. Thieffry and S. Sarkar. Forty Years under the Central Dogma. *Trends in Biochemical Sciences*, 23(8):312–316, 1998.
- [170] G. Thijs, M. Lescot, K. Marchal, S. Rombauts, B. De Moor, P. Rouzé, and Y. Moreau. A Higher-order Background Model Improves the Detection of Promoter Regulatory Elements by Gibbs Sampling. *Bioinformatics*, 17(21):1113–1122, 2001.
- [171] G. Thijs, K. Marchal, M. Lescot, S. Rombauts, B. De Moor, P. Rouzé, and Y. Moreau. A Gibbs Sampling Method to Detect Over-represented Motifs in the Upstream Regions of Co-expressed Genes. In *Proceedings of the Fifth Annual International Conference on Computational Biology*, pages 305–312, 2001.
- [172] W. Thompson, E. C. Rouchka, and C. E. Lawrence. Gibbs Recursive Sampler: Finding Transcription Factor Binding Sites. *Nucleic Acids Research*, 31:3580–3585, 2003.
- [173] M. Tompa. An Exact Method for Finding Short Motifs in Sequences, with Application to the Ribosome Binding Site Problem. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, pages 262–271, 1999.
- [174] M. Tompa, N. Li, T. L. Bailey, G. M. Church, B. De Moor, E. Eskin, A. V. Favorov, M. C. Frith, Y. Fu, W. J. Kent, V. J. Makeev, A. A. Mironov, W. S. Noble, G. Pavesi, G. Pesole, M. Régnier, N. Simonis, S. Sinha, G. Thijs, J. van Helden, M. Vandenbogaert, Z. Weng, C. Workman, C. Ye, and Z. Zhu. Assessing Computational Tools for the Discovery of Transcription Factor Binding Sites. *Nature Biotechnology*, 23:137–144, 2005.

REFERENCES

- [175] J. van Helden, B. André, and J. Collado-Vides. Extracting Regulatory Sites from the Upstream Region of Yeast Genes by Computational Analysis of Oligonucleotide Frequencies. *Journal of Molecular Biology*, 281(5):827–842, 1998.
- [176] J. van Helden, A. F. Rios, and J. Collado-Vides. Discovering Regulatory Elements in Non-coding Sequences by Analysis of Spaced Dyads. *Nucleic Acids Research*, 28(8):1808–1818, 2000.
- [177] P. Van Loo and P. Marynen. Computational Methods for the Detection of Cis-regulatory Modules. *Briefings in Bioinformatics*, 10(5):509–524, 2009.
- [178] A. Vanet, L. Marsan, A. Labigne, and M.-F. Sagot. Inferring Regulatory Elements from a Whole Genome: an Analysis of Helicobacter Pylori Sigma(80) Family of Promoter Signals. *Journal of Molecular Biology*, 297(2):335–353, 2000.
- [179] P. S. Verma and V. K. Aarwal. *Cell Biology, Genetics, Molecular Biology, Evolution and Ecology*. S. Chand & Company., 2004.
- [180] S. Vikas, S. Deepak, T. Ritu, and S. Anupam. Motif Discovery Using Artificial Bee Colony Algorithm. *Journal of Information Technology Research*, 5(4):30–47, 2012.
- [181] A. Wagner. Genes Regulated Cooperatively by One or More Transcription Factors and Their Identification in Whole Eukaryotic Genomes. *Bioinformatics*, 15(10):776–784, 1999.
- [182] T. Waleev, D. Shtokalo, T. Konovalova, N. Voss, E. Cheremushkin, P. Stegmaier, O. Kel-Margoulis, E. Wingender, and A. Kel. Composite Module Analyst: Identification of Transcription Factor Binding Site Combinations Using Genetic Algorithm. *Nucleic Acids Research*, 34(Web Server issue):w541–545, 2006.
- [183] D. Wang and X. Li. GAPK: Genetic Algorithms with Prior Knowledge for Motif Discovery in DNA Sequences. In *Proceedings of the Eleventh Conference on Congress on Evolutionary Computation*, pages 277–284, 2009.
- [184] D. Wang and X. Li. iGAPK: Improved GAPK Algorithm for Regulatory DNA Motif Discovery. In *Proceedings of the Seventeenth International Conference on Neural Information Processing: Models and Applications*, pages 217–225, 2010.

REFERENCES

- [185] G. Wang and W. Zhang. Build a Dictionary, Learn a Grammar, Decipher Stego-scripts, and Discover Genomic Regulatory Elements. In *Systems Biology and Regulatory Genomics*, Lecture Notes in Computer Science, pages 80–94. Springer Berlin/Heidelberg, 2006.
- [186] T. Wang and G. D. Stormo. Combining Phylogenetic Data with Co-regulated Genes to Identify Regulatory Motifs. *Bioinformatics*, 19(18):2369–2380, 2003.
- [187] M. Waterman, R. Arratia, and D. Galas. Pattern Recognition in Several Sequences: Consensus And Alignment. *Bulletin of Mathematical Biology*, 46(4):515–527, 1984.
- [188] G. C. G. Wei and M. A. Tanner. A Monte Carlo Implementation of the EM Algorithm and the Poor Man’s Data Augmentation Algorithms. *Journal of the American Statistical Association*, 85:699–704, 1990.
- [189] J. N. Weinstein and Y. Pommier. Connecting Genes, drugs and diseases. *Nature Biotechnology*, 24:1365–1366, 2006.
- [190] H. Weintraub, R. Davis, D. Lockshon, and A. Lassar. MyoD Binds Cooperatively to Two Sites in a Target Enhancer Sequence: Occupancy of Two Sites Is Required for Activation. *Proceedings of the National Academy of Sciences*, 87(15):5623–5627, 1990.
- [191] T. Werner. Models for Prediction and Recognition of Eukaryotic Promoters. *Mammalian Genome*, 10:168–175, 1999.
- [192] E. Wijaya, K. Rajaraman, S.-M. Yiu, and W.-K. Sung. Detection of Generic Spaced Motifs Using Submotif Pattern Mining. *Bioinformatics*, 23(12):1476–1485, 2007.
- [193] L. Wodicka, H. Dong, M. Mittmann, H.-M. Ho, and D. J. Lockhart. Genome-wide Expression Monitoring in *Saccharomyces Cerevisiae*. *Nature Biotechnology*, 15(3):1359–1367, 1997.
- [194] F. Wolfertstetter, K. Frech, G. Herrmann, and T. Werner. Identification of Functional Elements in Unaligned Nucleic Acid Sequences by a Novel Tuple Search Algorithm. *Bioinformatics*, 12(1):71–80, 1996.

REFERENCES

- [195] C. T. Workman and G. D. Stormo. ANN-Spec: a Method for Discovering Transcription Factor Binding Sites with Improved Specificity. In *Pacific Symposium on Biocomputing*, pages 467–478, 2000.
- [196] E. P. Xing, M. I. Jordan, R. M. Karp, and S. Russell. A Hierarchical Bayesian Markovian Model for Motifs in Biopolymer Sequences. In *Proceedings of Advances in Neural Information Processing Systems*, pages 200–203, 2003.
- [197] E. P. Xing, W. Wu, M. I. Jordan, and R. M. Karp. Logos: a Modular Bayesian Model for de novo Motif Detection. *Journal of Bioinformatics and Computational Biology*, 2(1):127–154, 2004.
- [198] X. Yang and J. C. Rajapakse. Graphical Approach to Weak Motif Recognition. *Genome Informatics Series*, 15(2):52–62, 2004.
- [199] Z. Yao, Z. Weinberg, and W. L. Ruzzo. CMfinder-A Covariance Model Based RNA Motif Finding Algorithm. *Bioinformatics*, 22(4):445–452, 2006.
- [200] C. H. Yuh, H. Bolouri, and E. H. Davidson. Genomic Cis-regulatory Logic: Experimental and Computational Analysis of a Sea Urchin Gene. *Science*, 279(5358):1896–1902, 1998.
- [201] F. Zambelli, G. Pesole, and G. Pavesi. Motif Discovery and Transcription Factor Binding Sites Before and After the Next-Generation Sequencing Era. *Briefings in Bioinformatics*, pages 1–13, 2012.
- [202] F. Zare-Mirakabad, H. Ahrabian, M. Sadeghi, S. Hashemifar, A. Nowzari-Dalini, and B. Goliaei. Genetic Algorithm for Dyad Pattern Finding in DNA Sequences. *Genes & Genetic Systems*, 84(1):81–93, 2009.
- [203] X. Zhao and S.-H. Sze. Motif Finding in DNA Sequences Based on Skipping Non-conserved Positions in Background Markov Chains. *Journal of Computational Biology*, 18(5):759–770, 2011.
- [204] Q. Zhou and J. S. Liu. Modeling within-motif Dependence for Transcription Factor Binding Site Predictions. *Bioinformatics*, 20(6):909–916, 2004.

REFERENCES

- [205] C. Zhu, K. J. R. P. Byers, R. P. McCord, Z. Shi, M. F. Berger, D. E. Newburger, K. Saulrieta, Z. Smith, M. V. Shah, M. Radhakrishnan, A. A. Philippakis, Y. Hu, F. De-Masi, M. Pacek, A. Rolfs, T. Murthy, J. LaBaer, and M. L. Bulyk. High-Resolution DNA-Binding Specificity Analysis of Yeast Transcription Factors. *Genome Research*, 19(4):556–566, 2009.