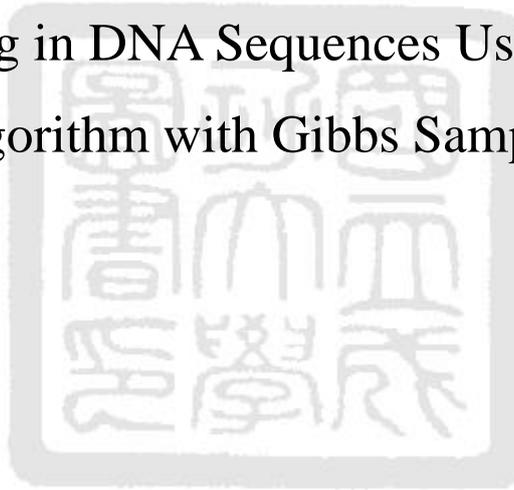


國 立 成 功 大 學
電 機 工 程 學 系
碩 士 論 文

結合基因演算法與吉氏抽樣數值方法預測 DNA 序
列上之特殊功能序列

Motif Finding in DNA Sequences Using a Genetic
Algorithm with Gibbs Sampler



研 究 生：黃瑋婷

指 導 教 授：王振興

中 華 民 國 九 十 六 年 七 月

博碩士論文授權書
(國科會科學技術資料中心版本93.2.6)

本授權書所授權之論文為本人在 國立成功大學 (學院) 電機工程學 系所
控制 組 九十五 學年度第 二 學期取得 碩 士學位之論文。

論文名稱：結合基因演算法與吉氏抽樣數值方法預測 DNA 序列上之特殊功能序列

同意 不同意

本人具有著作財產權之論文全文資料，授予行政院國家科學委員會科學技術資料中心(或其改制後之機構)、國家圖書館及本人畢業學校圖書館，得不限地域、時間與次數以微縮、光碟或數位化等各種方式重製後散布發行或上載網路。

本論文為本人向經濟部智慧財產局申請專利(未申請者本條款請不予理會)的附件之一，申請文號為：_____，註明文號者請將全文資料延後半年再公開。

同意 不同意

本人具有著作財產權之論文全文資料，授予教育部指定送繳之圖書館及本人畢業學校圖書館，為學術研究之目的以各種方法重製，或為上述目的再授權他人以各種方法重製，不限地域與時間，惟每人以一份為限。

上述授權內容均無須訂立讓與及授權契約書。依本授權之發行權為非專屬性發行權利。依本授權所為之收錄、重製、發行及學術研發利用均為無償。上述同意與不同意之欄位若未鈎選，本人同意視同授權。

指導教授姓名：王 振 興

研究生簽名：黃孝婷
(親筆正楷)

學號：N26944127
(務必填寫)

日期：民國 96 年 8 月 2 日

-
1. 本授權書(得自<http://sticnet.stic.gov.tw/sticweb/html/theses/authorize.html>下載或至<http://www.stic.gov.tw>首頁右下方下載)請以黑筆撰寫並影印裝訂於書名頁之次頁。
 2. 授權第一項者，請確認學校是否代收，若無者，請自行寄論文一本至台北市(106)和平東路二段106號1702室 國科會科學技術資料中心 黃善平小姐。(電話：02-27377606 傳真:02-27377689)

國立成功大學
碩士論文

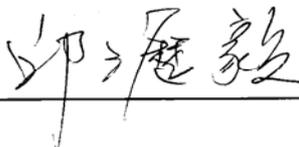
結合基因演算法與吉氏抽樣數值方法預測 DNA 序
列上之特殊功能序列

Motif Finding in DNA Sequences Using a Genetic
Algorithm with Gibbs Sampler

研究生：黃瑋婷

本論文業經審查及口試合格特此證明
論文考試委員







指導教授：王振興

系(所)主管：  

中華民國九十六年七月十四日

Motif Finding in DNA Sequences Using a Genetic
Algorithm with Gibbs Sampler

By

Wei-Ting Huang

*A Thesis Submitted to the Graduate Division in Partial Fulfillment
of the Requirement for the Degree of*

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

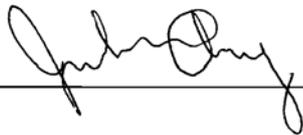
NATIONAL CHENG KUNG UNIVERSITY

TAINAN, TAIWAN

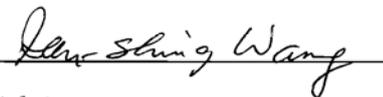
REPUBLIC OF CHINA

JULY 14, 2007

Approved by







Advisor



Chairman



結合基因演算法與吉氏抽樣數值方法預測 DNA 序列上之特殊功能序列

黃瑋婷* 王振興**

國立成功大學電機工程學系

摘要

在分子生物學中，共同序列通常代表保留功能區域(conserved functional domains)。預測去氧核糖核酸(DNA)序列上之特殊功能序列在計算生物學上是一個十分重要的問題，其重要的應用為尋找保留功能區域。在這篇論文中，我們評估數種預測 DNA 序列上之特殊功能序列的方法，包括基因演算法、結構基因演算法以及吉氏抽樣數值方法(Gibbs sampler method)。其中結構基因演算法不需輸入所要尋找之功能序列的長度，而能自行收斂至較合適的長度是與其他兩種方法不同的地方。從這三種方法的模擬結果中，基因演算法的效能比其他兩種方法好。然而，吉氏抽樣數值方法在計算上較有效率。為了在預測特殊功能序列問題上有更好的效能及計算效率，我們結合基因演算法與吉氏抽樣數值方法。我們所提之方法的有效性可從測試範例的模擬結果證實。

* 研究生

** 指導教授

Motif Finding in DNA Sequences Using a Genetic Algorithm with Gibbs Sampler

Wei-Ting Huang^{*} and Jeen-Shing Wang^{**}

Department of Electrical Engineering
National Cheng Kung University, Tainan 701, Taiwan, R.O.C.

Abstract

In molecular biology, consensus sequences often represent conserved functional domains. Motif finding in DNA sequences is a fundamental problem in computational biology, with important applications in finding conserved functional domains. In this thesis, we evaluate several methods including a genetic algorithm (GA), the structured genetic algorithm (S-GA), and Gibbs sampler method in the motif finding problem. The structured genetic algorithm can search for motifs of varying lengths without a fixed motif length as the input, which is different from the other two methods. From the simulation results of these three methods, the GA performs better than the other two approaches. However, the Gibbs sampler method is more efficient in computing. To enjoy better performance and efficiency, we combine the GA with the Gibbs sampler method for motif finding problems. We have validated the effectiveness of our approach through extensive simulations in diverse benchmark examples.

* Student

** Advisor

Acknowledgement

I would like to express my gratitude to my advisor, Prof. Jeen-Shing Wang, who patiently encouraged me to complete this thesis. I am deeply indebted to his help and suggestions. Without his guidance and inspiration, this thesis could not be successfully completed in time. During the last period, Prof. Wang usually stayed up revising our theses. I deeply appreciate his dedication.

Also, I offer my heartfelt thanks to Prof. Lih-Yih Chiou and Prof. Jung-Hsien Chiang, the committee members during the oral defense. Thanks for reading a draft of this thesis and giving me precious comments and suggestions.

I would like to thank my team members in CILS lab for their discussions with me when I felt confused. During my thesis writing, I especially want to thank for my friends (Yi-Han Liu, Chieh-Chih Yang, Jian-Hui Wu, Tai-Chi Kuo, Ming-Chuan Xie, Sheng-An Chang, Jun-Ming Hu, Shi-Ze Wei, Ren-Jie Hong., etc). When I feel down, Yi-Han Liu and Jian-Hui Wu are always by my side to give me support and comfort to overcome my difficulties. Jun-Ming Hu and Shi-Ze Wei always play table-tennis with me when my mood is really bad. I especially thank for Chieh-Chih Yang, he always gives me help and pushes me to go ahead. My numerous complaints almost drove him mad. Thank for all my friends that your company supports me during the really hard period.

My deepest appreciation goes to my parents. They are my strongest support in heart. Finally, I would like to thank everyone in CILS lab and all people who give me help to finish this thesis in the last stage of my student life.

TABLE OF CONTENTS

	Page
CHINESE ABSTRACT	i
ABSTRACT	ii
ACKNOWLEDGEMENT	iii
LIST OF TABLES.....	vi
LIST OF FIGURES.....	vii
1 Introduction.....	1-1
1.1 Biological Background.....	1-1
1.1.1 The Representation and Technicalities	1-1
1.1.2 Protein Synthesis	1-2
1.1.3 Transcription Factor Binding Sites.....	1-2
1.2 Motivation.....	1-4
1.3 Literature Review	1-5
1.4 Goal.....	1-6
1.5 Architecture	1-7
2 The Motif Finding Problem	2-1
2.1 The Motif Finding Problem	2-1
2.2 Some Methods for Motif Finding.....	2-5
2.2.1 Gibbs Sampler	2-6
2.2.2 CONSENSUS	2-7
2.2.3 Expectation Maximization.....	2-8
3 Method	3-1
3.1 Introduction of the Genetic Algorithm	3-1
3.2 Chromosome and Encoding.....	3-2
3.2.1 Chromosome.....	3-2

3.2.2	Encoding	3-4
3.3	The Flow of the Algorithm	3-4
3.4	Genetic Operators	3-6
3.4.1	Selection	3-6
3.4.2	Crossover	3-7
3.4.3	Mutation.....	3-9
3.5	Fitness Function.....	3-9
3.6	Structured Genetic Algorithm.....	3-10
3.7	Combine Gibbs Sampler with Genetic Algorithm.....	3-12
4	Simulation Results	4-1
4.1	Experimental Results	4-1
4.1.1	Genetic Algorithm	4-2
4.1.2	Genetic Algorithm vs. Structured Genetic Algorithm	4-4
4.1.3	Gibbs Sampler Method.....	4-5
4.1.4	Gibbs Sampler Method with Genetic Algorithm.....	4-6
5	Conclusion.....	5-1
5.1	Conclusion and Contribution.....	5-1
5.2	Future Work	5-2

References

LIST OF TABLES

Table	Page
1 The alignment matrix, profile matrix, and consensus pattern.	2-4
2 The positional weight matrix (PWM)	2-5
3 Encoder table for DNA letters.....	3-4
4 The results tested with motif length 10 and sequence length 100.....	4-2
5 The results tested with motif length 10 and sequence length 300.....	4-2
6 The results tested with motif length 10 and sequence length 500.....	4-3
7 The results tested with motif length 15	4-3
8 The results tested with motif length 10 and sequence length 100.....	4-4
9 The results tested with motif length 10 and sequence length 300.....	4-4
10 The results tested with motif length 10 and sequence length 500.....	4-4
11 The results tested with motif length 10	4-5
12 The results tested with motif length 15	4-6
13 The results tested with motif length 15	4-7
14 The results tested with motif length 15	4-7
15 The results tested with the motif length as 15 and the sequence length as 1000	4-8

LIST OF FIGURES

Figure	Page
1 The other factors will automatically continue to bind with the complex.....	1-3
2 The transcription process is initiated by the binding of the transcription factors, the RNA polymerase II, and the promoter region in the DNA.	1-3
3 Illustration of the motif finding problem with a set of t DNA sequences.....	2-3
4 The population and the representation of the genetic algorithm are shown above.	3-3
5 The flowchart of the genetic algorithm in motif finding.....	3-5
6 The single point crossover diagram is shown above.....	3-7
7 The two-point crossover diagram is shown above.....	3-8
8 The uniform crossover diagram is shown above.....	3-8
9 A diagram for mutation is shown above.....	3-9
10 (a) This is the multi-layered structure of the representation. (b) A flattened linear representation encoding form is shown.....	3-11
11 This procedure is to combine Gibbs sampler with the genetic algorithm.....	3-14

Chapter 1

Introduction

1.1 Biological background

In this section, some main concepts in biology relevant to our discussion of motif finding in DNA sequences will be presented. Other common biological terms will also be described briefly.

1.1.1 The representation and technicalities

In molecular biology, a nucleic acid is a complex biochemical macromolecule and is found in all living cells and viruses. The nucleic acid is composed of nucleotide chains. The most common nucleic acids are deoxyribonucleic acid (DNA) and ribonucleic acid (RNA). DNA is a long molecule that has the form of a double helix. There are four types of nucleotides constituting the DNA sequences: adenine (A), guanine (G), cytosine (C), and thymine (T). The four base letters of the RNA sequence are A, G, C, and uracil (U), where the T in DNA is replaced by U in RNA.

A gene is a segment of a DNA molecule. Genes are like sentences built out of the nucleotide alphabet and contain the information necessary to produce a functional product, usually a protein. Proteins are large organic compounds and are composed of 20 different kinds of amino acids.

1.1.2 Protein synthesis

Proteins are essential to life and are important components of cells. The information for producing proteins is stored in DNA. Transcription and translation make DNA convert to protein, a process which is viewed as the central dogma in molecular biology. This central dogma was first enunciated by Francis Crick in 1958 and re-stated in an article published in 1970 [1] in *Nature*. The main principle states that genetic information flows from DNA to mRNA to protein. During the process of transcription from DNA to messenger RNA (mRNA), the message stored in DNA will be recorded by mRNA. Afterwards, the mRNA will be read by a ribosome to translate to the protein. Transcription may be thought of as a rewriting of the information from DNA to mRNA. The language both in DNA and mRNA is nucleic acid-based. In contrast to the translation, the information is translated from one language, which is nucleic acid-based in mRNA, into the amino acid-based language of proteins.

1.1.3 Transcription factor binding sites

During the process of transcription, the specific transcription factor protein will bind to the specific binding sites in the DNA. The binding sites are usually located in the upstream promoter region. And in different organizations, there are different transcription factors to recognize the tissue-specific promoter. For example, muscle-specific transcription factors will recognize the muscle-specific promoter, and the liver-specific promoter will be found by liver-enriched transcription factors.

Then the transcription factor proteins bind each other and associate with an enzyme called RNA polymerase II to form a complex. And the other factors will automatically continue to bind with the complex on the specific site in the promoter to activate the transcription and this is shown in Fig 1. The initiation of the transcription process is shown [2] in Fig. 2. Some transcription factors bind directly to specific sites in the DNA

and others bind to each other, but most of them bind both to the DNA as well as to other transcription factors.

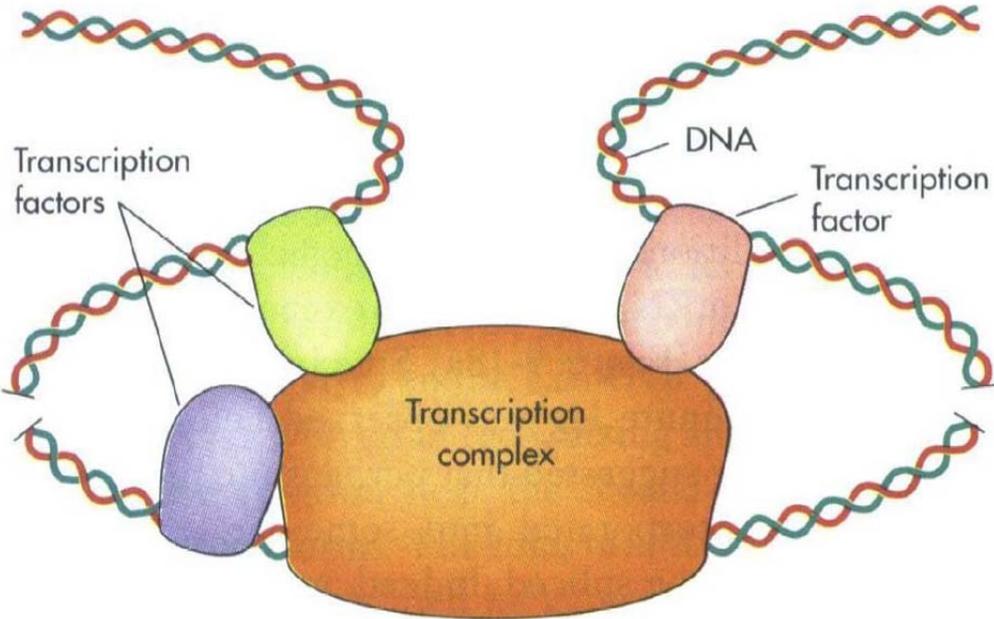


Fig. 1. The other factors will automatically continue to bind with the complex.

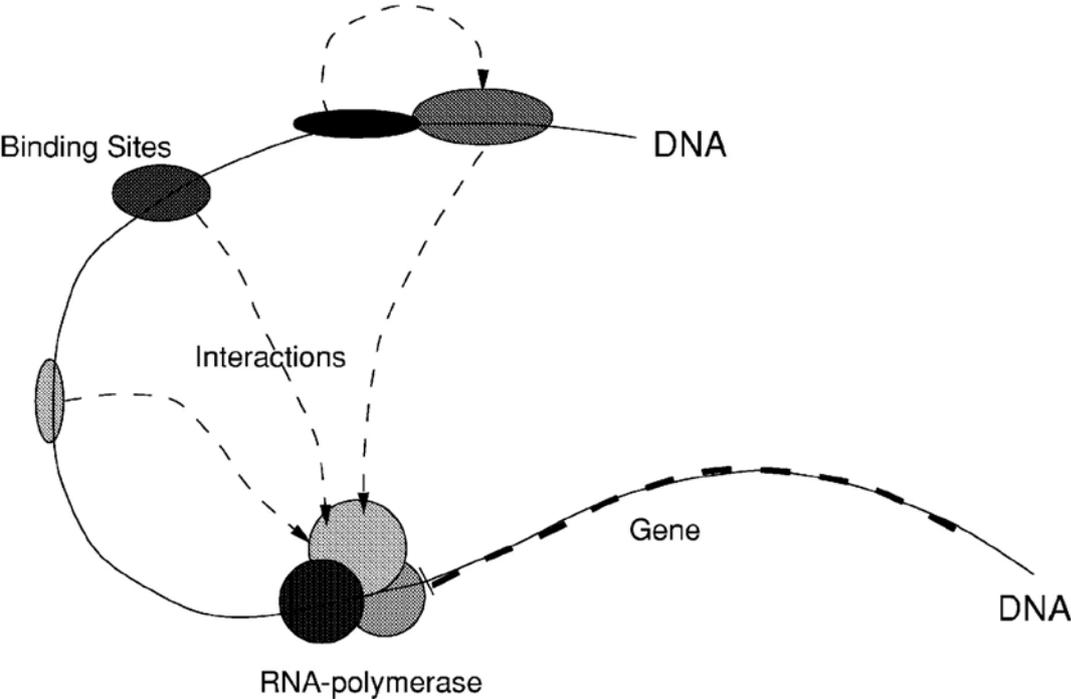


Fig. 2. The transcription process is initiated by the binding of the transcription factors, the RNA polymerase II, and the promoter region in the DNA.

1.2 Motivation

The action of the transcription factors is very important because it can control the transcription rate directly. The regulation of transcription is a highly complex process as it is dependent upon a number of factors, including transcription factors and other co-regulatory proteins [3]. More specifically, transcription factors regulate the binding of RNA polymerase II to DNA. And as a result, transcription factors control the subsequent translation of DNA into mRNA and eventually protein. In fact, the transcription factors are effective in the initiation, stimulation, and termination of the transcription process. When the transcription factor significantly decreases the transcription of a gene, it is called a repressor; if the expression of a gene increases, it is an activator. In other words, the transcription factor is the regulation protein. Why is the regulation protein so important? Many human diseases are formed on account of cells or genes being controlled abnormally. The regulation proteins can cure diseases by blocking the deleterious effects of certain disease-causing genes to make the next physiological functions unable to proceed. But in some situations, the regulation proteins cure the diseases by assisting the metabolism to improve the resistance of the human body instead.

The transcription factor binding site is a conserved region and is considered as the consensus sequence [4]. Actually, transcription factor binding sites are just a kind of consensus sequence. In the previous paragraphs, we take transcription factor binding sites as an example of consensus sequences. The description of the transcription procedure is in order to clarify the importance of finding consensus sequences from a biological point of view.

Consensus sequences often represent conserved functional domains. Current research on finding consensus sequences include: predicting motifs on the promoter, finding consensus sequences in the introns, detection of conserved secondary structures, and finding domains on proteins. Consensus sequences, also called motifs, are defined as the

sequences which give the most common nucleotide or amino acid at each position in a set of DNA, RNA, or protein sequences such as transcription factor binding sites to affect or regulate the forming, expression, or function of the protein. Motif discovery in DNA sequences is a fundamental problem in computational biology, with important applications in finding functional domains. In the next section, the historical literature review will be presented.

1.3 Literature review

Motif finding problems in sequence analysis typically involve the detection of transcription factor binding sites or conserved domains. Some commonly used algorithms to find motifs in nucleotide and amino acid sequences have been proposed, including Lawrence *et al.*'s Gibbs sampler in 1993 [5], Bailey and Elkan's MEME in 1994 [6], Hertz and Stormo's CONSENSUS in 1999 [7], Buhler and Tompa's PROJECTION in 2001 [8], Eskin and Pevzner's MITRA in 2002 [9], Keich and Pevzner's MULTIPROFILER in 2002 [10], and Price, Ramabhadran, and Pevzner's ProfileBranching and PatternBranching in 2003 [11].

Recently developed methods include the structured genetic algorithm in 2003 [23], the genetic algorithm in 2004 [3], the uniform projection algorithm in 2004 [12], and use of a self-organizing neural network in 2006 [13]. The uniform projection algorithm replaces the random selection of projections by a greedy heuristic that approximately equalizes the coverage of the projections. The genetic algorithm and structured genetic algorithm will be discussed in chapter 3 of this thesis.

Algorithms for motif finding can be categorized into two types: profile-based algorithms and pattern-based algorithms [11]. Profile-based algorithms predict the starting positions of the occurrences of the motif in each sequence, while pattern-based algorithms predict the motif itself.

In profile-based methods, such as in stochastic Gibbs sampler, MEME, greedy CONSENSUS, and ProfileBranching, the motif is represented by a position weight matrix. Gibbs sampler branches stochastically in the space of starting positions. MEME applies the Expectation Maximization algorithm [14], [15] to selected substrings of the sample as seeds. These algorithms all try to find a motif that maximizes some score. There is a detailed description about Gibbs sampler method, the Expectation Maximization algorithm, and the CONSENSUS algorithm in chapter 2.

In pattern-based methods, such as PROJECTION, MITRA, MULTIPROFILER, and PatternBranching, the motif is represented as a string with mismatches. PROJECTION applies locality-sensitive hashing to search for conserved patterns in a set of sequences. MITRA uses a mismatched tree data structure and splits the space of all possible patterns into disjointed subspaces that start with a given prefix. PatternBranching starts with random seed strings and performs local searches starting from these seeds. In this thesis, profile-based algorithms are discussed.

1.4 Goal

Genetic algorithms are applied in many fields. In this thesis, the basic type to the advanced type and a combination with other methods for the motif finding problem are discussed extensively. There are many parts to modify in a basic genetic algorithm such as selection, crossover, and mutation schemes. Parameter regulation is also a direction to adjust for improving performance, but this is case by case. When the evolution stops proceeding, regulating the crossover and mutation rate to an appropriate degree is also being discussed. Two common sources of modifications are very prominent; one is the transformation of fitness function and the other one is the effective selection scheme. The main purpose of these techniques is to maintain diversity in the population in order to avoid premature convergence.

Another direction of modifications is to change the traditional representation form such as the structured genetic algorithm. The purpose of changing the representation form is to allow multiple bit changes to enhance the diversity. On the other hand, in most cases of the motif finding problem, the motif length is usually fixed input. The structured genetic algorithms can search for motifs of varying lengths without a fixed motif length as input. It can converge to motifs of the size that fits best in the search space, which is different from other methods in motif finding problem.

We also combine the genetic algorithm with Gibbs sampler method. The genetic algorithm provides a better set of initial positions for Gibbs sampler method. How this combination improves the performance of Gibbs sampler method is discussed.

1.5 Architecture

This thesis is organized as follows. In chapter 1, the biological background relevant to the motif finding problem is introduced. A detailed description of the motif finding problem and several well-known motif finding methods such as Gibbs sampler, CONSENSUS, and Expectation Maximization are presented in Chapter 2. In Chapter 3, we present the genetic algorithm approach, structured genetic algorithm, and the combination method with Gibbs sampler and genetic algorithm for predicting motifs. The experimental results are shown in Chapter 4. Finally, the discussion of the simulation results, the conclusions, and future work are summarized in Chapter 5.

The motif finding problem

2.1 The motif finding problem

Identifying meaningful consensus patterns (motifs) from biological data has been studied extensively in computational biology on account of its paramount importance. A significant application is to discover binding sites for transcription factors in a set of DNA sequences as in the description in Chapter 1. This is like deciphering a message encrypted in a fragment. If a word occurs considerably more frequently than expected, then it is more likely to be some kind of signal, which is crucially important in figuring out the biological meaning of the signal.

Briefly, motif is a conserved pattern that is common in two or more sequences and can be found in DNA, RNA, and protein [16]. In general, the length of the DNA sequences is several hundred to several thousand nucleotides, and the motif pattern is usually 6 to 20 nucleotides in length. The pattern is relatively short and not completely conserved as a result of a naturally-occurring mutation. The motif is not exactly the same in each sequence because random point mutations may occur in the sequences. That is, a motif may have slight variances in the different sequences. In fact, it is quite difficult for all sequences to have a completely matched pattern, and no single sequence can define these patterns exactly. Besides, it is not necessary for the motif to appear in each of the given set of sequences. The patterns or motifs should be common in most, but not all sequences.

The description of *the motif finding problem* in DNA sequences is as follows [17]: Given a set of *DNA* sequences, there are t sequences in the set. The length of each sequence is n . M is the motif that occurs once in each sequence, and its length is l . The problem is to find the motif M . Fig. 3 illustrates the problem with t DNA sequences [18].

- Input:

DNA – The set of DNA sequences

t – The number of DNA sequences

n – The length of each DNA sequence

l – The length of the motif

- Output:

M – The motif patterns embedded in each sequence

We give an easy example to make the problem and the parameters more specific.

Assume S is a set of five DNA sequences, S_1 , S_2 , S_3 , S_4 , and S_5 .

S_1 : ACCTTGGGGGACT

S_2 : CTGAGAGACTAC

S_3 : GTTGGGGGACGA

S_4 : TTGAGAGACGAT

S_5 : AGCGCTGAGGGAC

Each sequence is 12 in length. This means that $t = 5$, $n = 12$, and the input *DNA* sequences is S . Assume the length of the motif, $l = 7$. Identify the most common base at each position. Then we get the consensus pattern, with mutations in position 3 and 5.

```

TGGGGGA
TGAGAGA
TGGGGGA
TGAGAGA
TGAGGGA
  
```

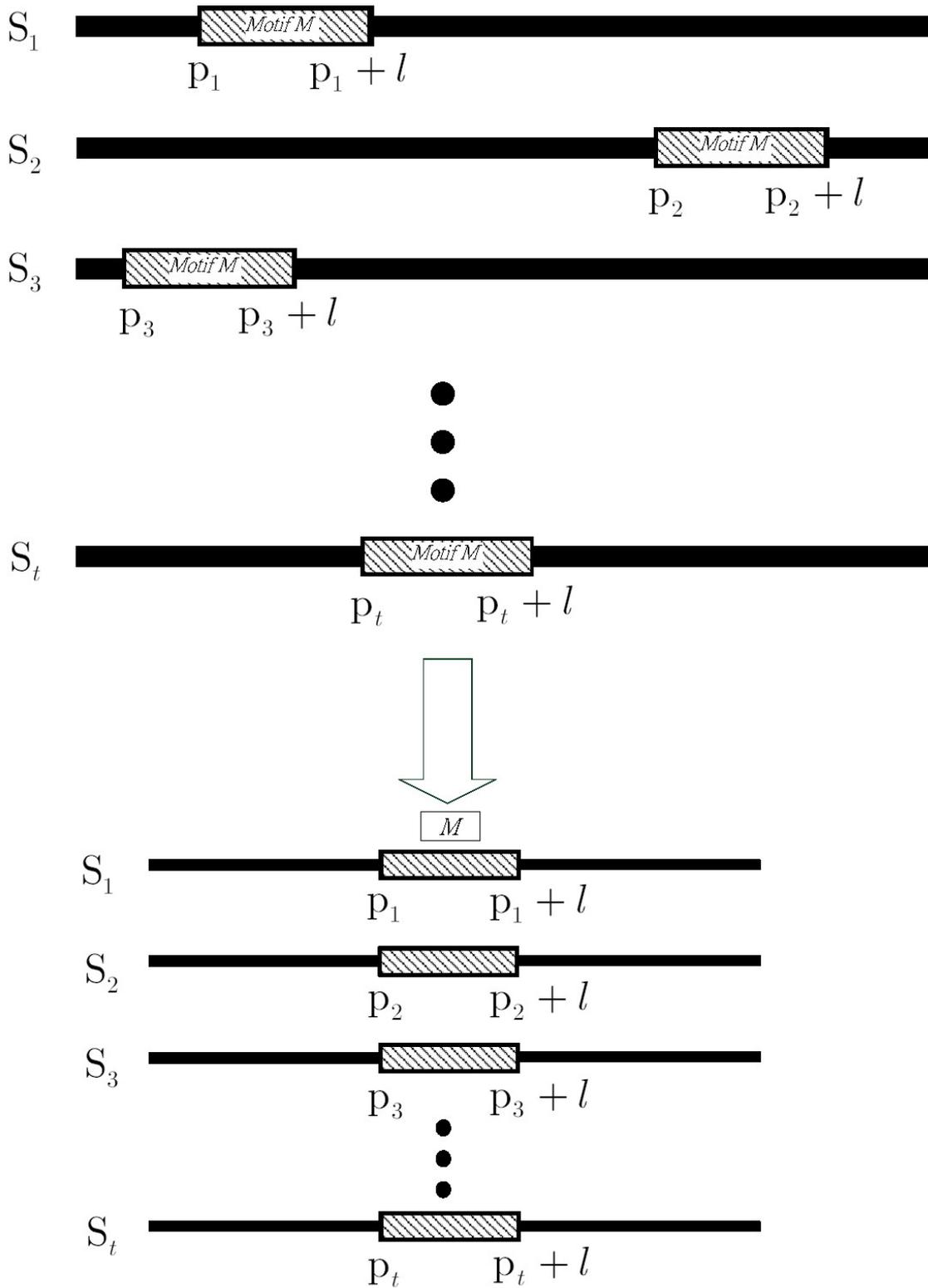


Fig. 3. Illustration of the motif finding problem with a set of t DNA sequences.

The exact algorithms to solve the motif finding problem are the brute force approach. The concept of the brute force algorithm is to compute the scores at all possible positions in each DNA sequence. If one sequence contains one pattern, there will be $(n - l + 1)$ positions, probably the starting position of the pattern which we want to find. Therefore, one sequence needs to check the position $(n - l + 1)$ times. There are t sequences, so for all sequences, the program needs to examine all possible combinations of these positions. There are $(n - l + 1)^t$ combinations to examine. For each possible starting position, the algorithm requires l operations to calculate the scores of the profile matrix. Thus, the overall complexity of the algorithm is evaluated as $O(l * (n - l + 1)^t) \approx O(l * n^t)$. We take the previous example to list the profile matrix. The alignment matrix, profile matrix, and consensus pattern are shown in Table 1.

Table 1. The alignment matrix, profile matrix, and consensus pattern

Alignment		TGGGGGA
		TGAGAGA
		TGGGGGA
		TGAGAGA
		TGAGGGA
Profile	A	0 0 3 0 2 0 5
	T	5 0 0 0 0 0 0
	C	0 0 0 0 0 0 0
	G	0 5 2 5 3 5 0
Consensus		TGAGGGA

The best scores will show in the profile matrix, and the consensus pattern is obtained according to the best score of the profile matrix. The goal of the brute force algorithm is to find the maximum score of the profile by examining all possible positions. The brute force algorithm always finds the optimal solution, though it may be too slow to perform practical tasks. We take the previous example to explain how slow the algorithm is. In the previous paragraph, we calculate the complexity, which is $O(l*(n-l+1)^t) \approx O(l*n^t)$. This means that if $t = 8, n=1000, l = 10$, the program must perform $1.0000e+025$ computations. Assuming each computation takes a cycle on a 3 GHz CPU, it would take over one billion years to search all the possibilities. Many algorithms today sacrifice optimal solution for speed. In the next section, we describe some representative algorithms in the area of solving the motif finding problem.

2.2 Some methods for motif finding

In chapter 1, several algorithms to find motifs have been described briefly. Following are three representative algorithms in the area of solving the motif finding problem. They will be demonstrated in detail. Before discussing these algorithms, positional weight matrices (PWM) are a basic concept we should know. One can create a PWM by counting the number of times each base is seen at each position, and then dividing by the total number of sequences to get the probability. Table 2 shows the PWM of the previous example.

Table 2. The positional weight matrix (PWM)

	1	2	3	4	5	6	7
A	0.00	0.00	0.60	0.00	0.40	0.00	1.00
T	1.00	0.00	0.00	0.00	0.00	0.00	0.00
C	0.00	0.00	0.00	0.00	0.00	0.00	0.00
G	0.00	1.00	0.40	1.00	0.60	1.00	0.00

2.2.1 Gibbs sampler

Lawrence *et al.* proposed the Gibbs sampler method, which was published in *Science* in 1993 [5]. The main concept of the Gibbs sampler method is to guess systematically. The algorithm starts with a guess about where a motif is located in each input sequence, and then uses those guesses to make more informed guesses. It chooses motif locations randomly, so it is not a greedy algorithm, but it is affected by initial locations. The initial guesses are important because they will affect the performance directly.

The algorithm is as follows [26]:

Step 1: Randomly choose an initial position in each sequence.

Step 2: Pick one sequence at random. Call it S .

Step 3: Compute the PWM Q using the motif locations for the remaining $N-1$ sequences.

$Q_{r,j}$ represents an occurrence frequency, which is the number of sequences where nucleotide r appears at position j .

Step 4: Use the PWM Q to assign a score to each possible motif location in S . The formula for score calculation is as follows:

$$q_p = \sum_{r=1}^4 \sum_{j=1}^l Q_{r,j} \log \frac{Q_{r,j}}{B_j}. \quad (2.1)$$

$Q_{r,j}$ is the component of Q , q_p is the score at position p , and B_j is the background probability.

Step 5: We choose a motif location in S according to its score. Rather than just choosing the highest scoring motif location in S , we randomly choose a motif location. However, this is not like flipping a coin or rolling a fair die; otherwise, there would be no point in using the PWM to assign scores. Rather, think of rolling a weighted die, where each motif location is weighted according to its PWM score. The highest scoring motif location will have the highest probability to be chosen, but it is still possible that some other lower-scoring location will be chosen. This degree of uncertainty is a hallmark of the Gibbs sampling strategy. The new motif location replaces the original motif location on S .

Step 6: If the motif locations do not change after a predefined number of iterations, stop the algorithm; otherwise, go to Step 2.

2.2.2 CONSENSUS

Hertz and Stormo proposed the CONSENSUS method in 1999 [7] in an article in *Bioinformatics*. The CONSENSUS requires no additional prior information other than the size of the desired motif. Generally speaking, it works by extracting all possible subsequences of the correct length that exist in the sequences. Then it iteratively combines these subsequences together and calculates the PWM for each set, keeping the best ones at each step. Thus, the CONSENSUS method is a greedy algorithm. Before the description of this algorithm, we present the definition of “represent”: we say that an input sequence is represented in a set if the set contains a subsequence that is created from that input sequence.

The algorithm is as follows [26]:

Step 1: Input k sequences with the goal of finding motifs of length L . Using a sliding window of length L , extract all possible subsequences from every input sequence. For example, an input sequence of ‘AATCGG’ will yield 4 subsequences of length 3: AAT, ATC, TCG, and CGG.

Step 2: For each subsequence extracted from step 1, create a new set containing only that subsequence. Note that these sets may not contain unique subsequences because it is possible and indeed likely that different input sequences will contain identical subsequences. If each of the k subsequences are 6 nucleotides long, then for $L = 3$ there will be $k*4$ different sets.

Iterate $k-1$ times:

Step 3: Choose a set and call it A . Now we can create several new sets containing the contents of A and one more subsequence, and remove the original A from our pool of sets.

Create one such set for each possible subsequence derived from an input sequence that was not represented in A . Do this for every original set.

Step 4: For each new set, calculate the PWM and the relative entropy according to pre-defined background probabilities which can just be the overall base probabilities in all input sequences.

Step 5: Rank the sets according to relative entropy, and keep only the top d sets.

At the end of the algorithm, each set should contain k subsequences from each input sequence. Each set represents a possible motif.

2.2.3 Expectation Maximization

Lawrence and Reilly proposed the Expectation Maximization algorithm in 1990 [14], [15]. “Expectation Maximization” is a term for a class of algorithms that estimates the values of some set of unknowns based on a set of parameters. This part is called expectation step. Then it uses those estimated values to refine the parameters over several iterations. This part is called Maximization step. In the case of motif finding, our parameters are the entries in the PWM and the background nucleotide probabilities, and the unknowns are the scores for each possible motif position in all sequences.

The algorithm is as follows [26]:

Step 1: Start with N sequences and one initial guess for the motif position for each sequence.

Step 2: Compute a PWM using the motif locations for each of the N sequences and the background probabilities for each base using the non-motif locations.

Iterate until the values do not change between iterations.

Step 3 (Expectation step): Use the PWM and background probabilities to calculate the probability of each possible motif location in each sequence: for some motif location in a sequence, use the PWM to calculate the probability of that motif; then multiply that by

the probability that the remaining non-motif nucleotides in the sequence are background nucleotides. Then normalize all the probabilities so that they add up to 1 over each sequence. Thus, for each sequence, each index, minus a few at the end depending on the length of the motif will have associated with it the probability of being an instance of the motif.

Step 4 (Maximization step): Now use the probabilities of the motif locations to recalculate the PWM and background probabilities. Instead of using raw base counts for the number of times each base in the position is observed, use weighted counts based on the probabilities.



3.1 Introduction of the genetic algorithm

A genetic algorithm (GA) is a search technique used to find or approximate solutions. Because of its outstanding performance in optimization, genetic algorithms have been regarded as a function optimizer. The basic concept of genetic algorithms is designed to simulate processes in the natural system necessary for evolution. In the mid 1800s, Charles Darwin, a British naturalist, published a book, *On the Origin of Species*. Darwin proposed that populations evolve over the course of generations through the process of natural selection to adapt to their environment for the purpose of survival. The genetic algorithms proposed by John Holland in 1975 with the publication of his book, *Adaption in Natural and Artificial Systems*, follow the principles of Charles Darwin concerning survival of the fittest. This book was the first to present the concept of using selection, crossover, and mutation systematically as a problem-solving strategy to simulate processes of biological evolution. Individuals with a higher fitness will survive longer and produce more offspring. Hence, these individuals become more suited to their environment over time. Genetic algorithms are a particular class of evolutionary algorithms that use techniques inspired by evolutionary biology such as inheritance, selection, crossover, and mutation. As such, they represent an intelligent exploitation of a random search within a defined search space to solve a problem [19].

Genetic algorithms are categorized as global search methods. They are implemented as a computer simulation in which a population of abstract representations, which called chromosomes, the genotype, or the genome, of candidate solutions. These solutions, called individuals, members, or phenotypes to an optimization problem, evolve toward better solutions. Traditionally, solutions are represented in binary strings of 0s and 1s, but other encodings are also possible. The evolution usually starts from a population of randomly generated individuals and happens in generations. In each generation, the fitness of every individual in the population is evaluated. Multiple individuals are stochastically selected from the current population based on their fitness and modified to form a new population by recombining and possibly randomly mutating. The new population is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. If the algorithm has terminated due to a maximum number of generations, a satisfactory solution may or may not have been reached. Genetic algorithms find application in biogenetics, computer science, engineering, economics, chemistry, manufacturing, mathematics, physics, and other fields [20].

3.2 Chromosome and encoding

Before a genetic algorithm can be put to work on any problem, a method is needed to encode potential solutions to that problem in a form that a computer can process. In this section, the meanings of some key terms and the representation of genetic algorithms will be described.

3.2.1 Chromosome

In genetic algorithms, a population means a group of all individuals and an individual means a candidate solution. A genetic representation of the individual is usually called a

chromosome [21]. A standard representation of the solution is to encode solutions as a binary string: sequences of 0s and 1s, where the digit at each position represents the value of some aspect of the solution. Another similar approach is to encode solutions as strings of integers or decimal numbers. This approach allows for greater precision and complexity than the comparatively restricted method of using binary strings. A third approach is to represent individuals in a genetic algorithm as strings of letters. Fig. 4 illustrates the population and the chromosomal representation. It is important to note that evolutionary algorithms do not need to represent candidate solutions as data strings of fixed length. Some do represent them in this way, but others do not.

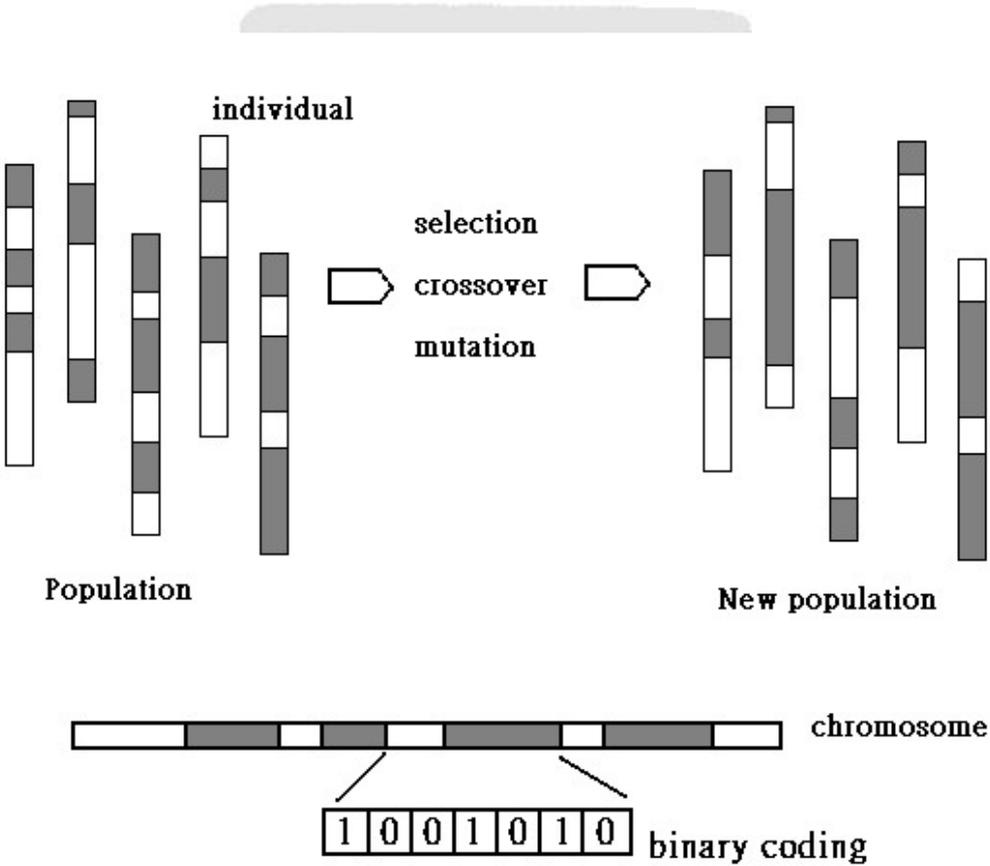


Fig. 4. The population and the representation of the genetic algorithm are shown above.

3.2.2 Encoding

In this thesis, evolved motifs are encoded as binary strings. In DNA sequence, there are four nucleotides: A, T, C, and G. We use two bits to encode the four letters. Table 3 shows the encoder table for DNA letters. Within double-stranded DNA, adenine (A) always pairs with thymine (T) and guanine (G) always pairs with cytosine (C) [22]. Based on this relationship, the binary code of adenine and thymine will be complementary. Guanine and cytosine are encoded to be complementary, too.

Table 3. Encoder table for DNA letters

Nucleotide	Binary code
A	00
T	11
C	10
G	01

3.3 The flow of the algorithm

According to [3], we use the genetic algorithm in the motif finding problem. The input DNA sequences will be encoded as binary strings. The population P is the set of candidate motif patterns, which are randomly produced with binary code. The evolution process will terminate after G generations. Fitness score function will evaluate each candidate motif, and the highest two will be kept as individuals of the new population. This strategy is known as Elitist selection. Through selection, crossover, and mutation, the new population is used in the next generation of the algorithm. If the highest fitness score stops increasing for a predefined number of generations, the algorithm will do floating regulation to crossover rate and mutation rate. The purpose of floating regulation is to increase crossover rate and mutation rate slightly to avoid the algorithm converging at a local optimum.

The architecture of the genetic algorithm in motif finding is stated in Fig. 5 [3]:

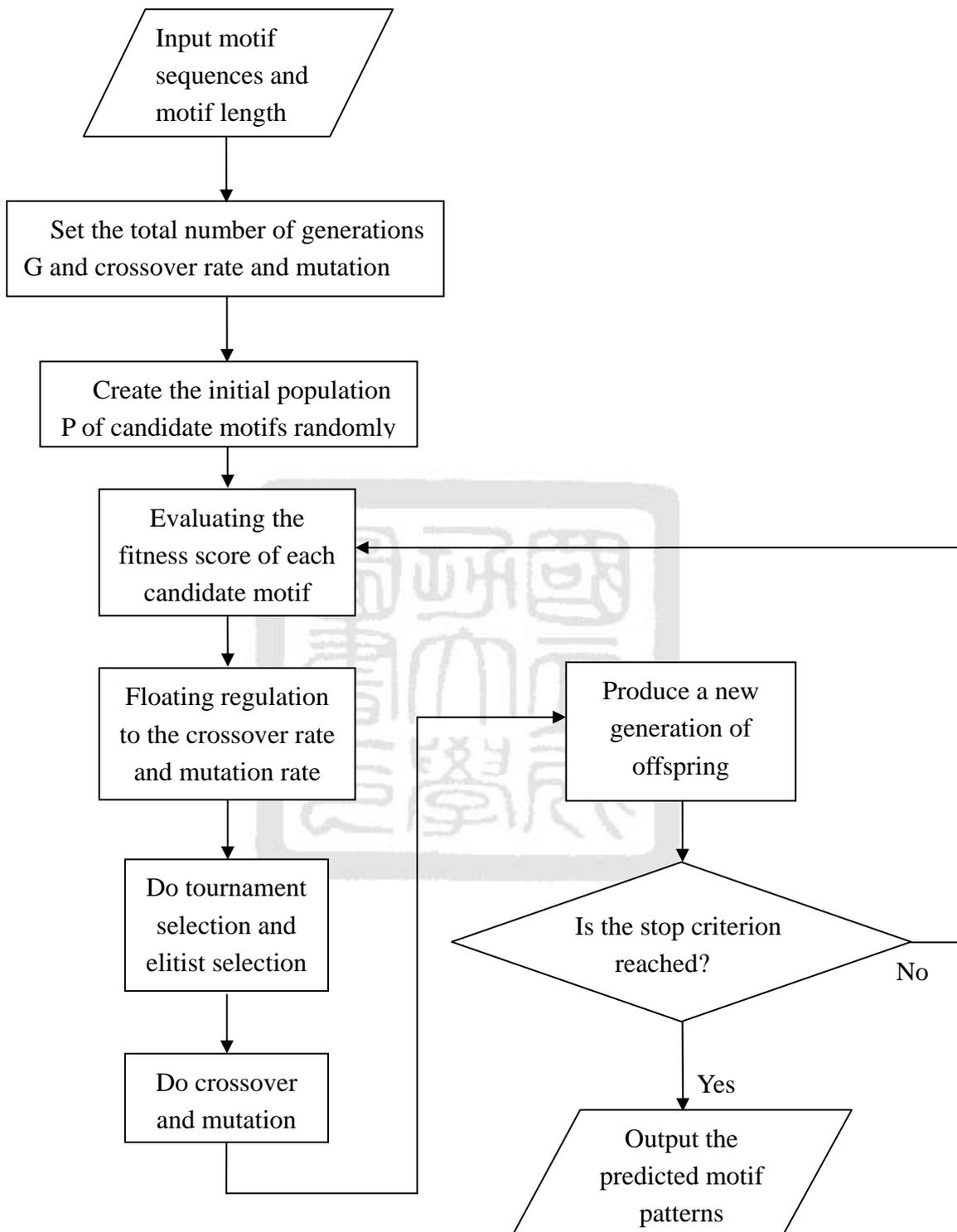


Fig. 5. The flowchart of the genetic algorithm in motif finding.

3.4 Genetic operators

There are three standard genetic operators during the process of evolution. The first step is selection. Individual solutions are selected according to fitness. The next step is to generate a new population of individual solutions from those selected through crossover and mutation. These operators are described in detail in this section.

3.4.1 Selection

The purpose of selection is to preserve individuals with higher fitness scores. Fitter solutions measured by a fitness function are typically more likely to be selected. There are many different methods which genetic algorithms can use to select the individuals, who will be copied into the next generation. Some methods select the best individuals, while some methods pick up individuals in a stochastic way in order to make all individuals possibly be selected.

The most common selection method is roulette wheel selection. Conceptually, this method is like a game of roulette; each individual gets a slice of the wheel, but the fitter ones get larger slices than the less fit ones. Hence, the fitter ones have a higher probability to be selected. This prevents premature convergence on poor solutions.

Another common method is tournament selection. In this method, the first step is to select two individuals from the population in a stochastic way. Generate a number randomly. Compare this number with a predefined number; if this number is smaller than the predefined number, select the individual with the higher fitness score from the two individuals selected before for the next generation. If this number is higher than the predefined number, select the worse one. In this thesis, we combine tournament selection with elitist selection. The elitist selection scheme is to guarantee that the fittest individual of each generation is selected. Although the fittest individuals are kept for the next generation, these individuals still do genetic operators.

3.4.2 Crossover

The first step of the crossover is to select a pair of parent individuals for breeding from the pool selected previously. Then, by exchanging the characteristics of the parent individuals, the offspring individuals are produced with one segment from each parent. This process is intended to simulate the analogous process of recombination that occurs to chromosomes during sexual reproduction. In order to exchange information with each other, the crossover site of the parent individuals should be decided upon. The information about each parent will swap by the crossover site. Then, both offspring are created and put into the new population, but sometimes only one child is created.

Crossover does not always occur. It is based on the crossover rate. If no crossover occurs, the parents are copied directly to the new population. There are many different kinds of crossover, the most common types of which are single point crossover, two-point crossover, and uniform crossover. They are stated as follows:

(1) Single point crossover:

In single point crossover, randomly select one crossover point, and then swap the information of the two parents with each other by the crossover point, as shown in Fig. 6.

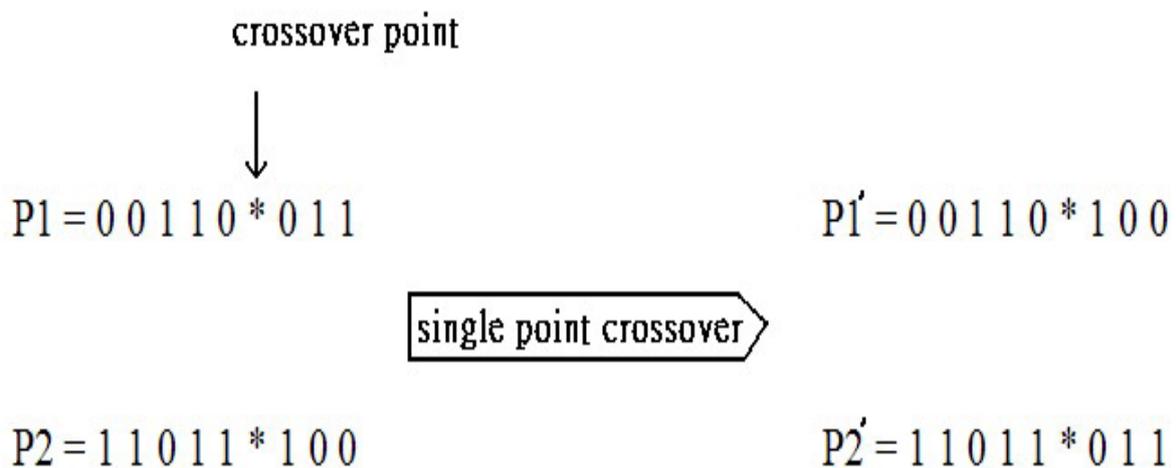


Fig. 6. The single point crossover diagram is shown above.

(2) Two-point crossover:

In two-point crossover, randomly select two crossover points, and then swap the information of the two parents with each other by the two crossover points, as shown in Fig. 7.

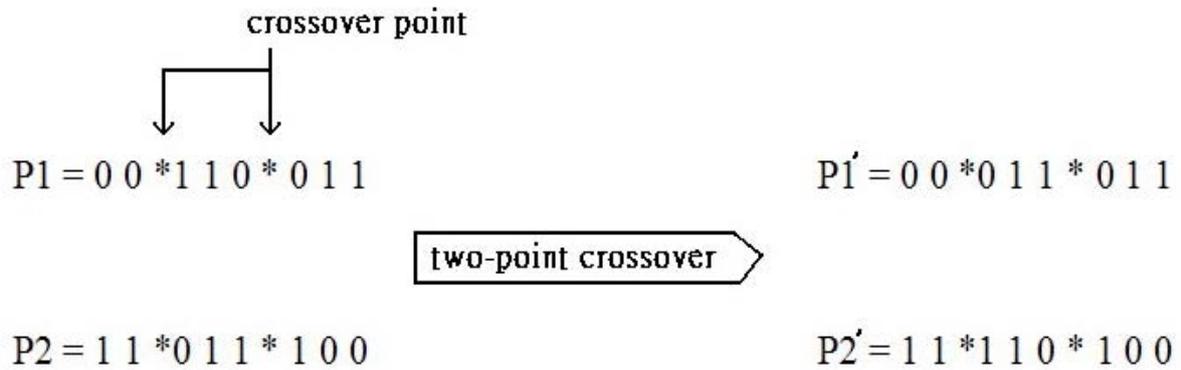


Fig. 7. The two-point crossover diagram is shown above.

(3) Uniform crossover:

In uniform crossover, randomly generate a mask with binary code the same length as the parent individual. If the bit of the mask is 1, perform an exchange in this bit of the two parent individuals, as shown in Fig. 8.

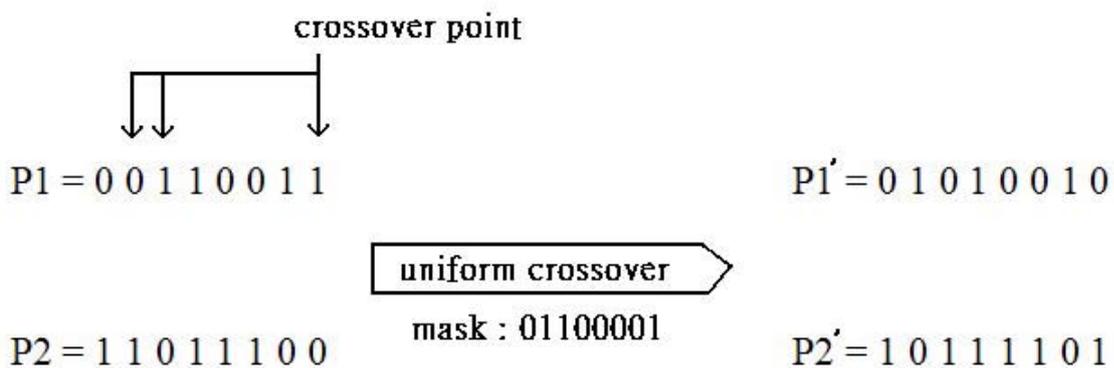


Fig. 8. The uniform crossover diagram is shown above.

3.4.3 Mutation

After selection and crossover, a new population of individuals is generated. In genetic algorithms, mutation is used to maintain genetic diversity from one generation of a population of individuals to the next. This is analogous to biological mutation. The method of mutation is as follows: generate a number randomly for each bit of the individual; if the number is smaller than the mutation rate, the value of this bit will become a complement of the original value. The purpose of mutation is to allow the algorithm to avoid local optimum by preventing the population of individuals from becoming too similar to each other. This will lead to slowing or even stopping evolution. Having genetic algorithms that not only take the fittest of the population is based on the same reason. A diagram for mutation is shown in Fig. 9.



Fig. 9. A diagram for mutation is shown above.

3.5 Fitness function

The fitness function is to evaluate how good the individuals are. In this part, the fitness function is adopted from [3]. The scores of each candidate motif pattern will be checked at all possible positions in every sequence. Hence, every pattern will have a score. First, one motif pattern will check the score of every possible position for one single sequence by comparing all the letters. If the letter of the motif pattern is the same as the one in the sequence, 1 will be added to the score. Thus, in one single sequence, the position with the highest score will be determined. For example, suppose the motif pattern P_1 and DNA sequence S_1 are as follows:

P_1 : AAGGCG

S_1 : TTCC**AAGGGG**CCATCGGACCTCT

Obviously, the score at position 5 in S_1 is the highest. There are five letters identical to P_1 . Hence, the score at position 5 is five. In the same way, the position with the highest score will be determined in all sequences. Second, the total fitness score of one motif pattern is determined by adding the highest score of all sequences. Finally, the score of each motif pattern is calculated.

Generally speaking, the average fitness score will increase generation by generation since the elitist selection scheme is used.

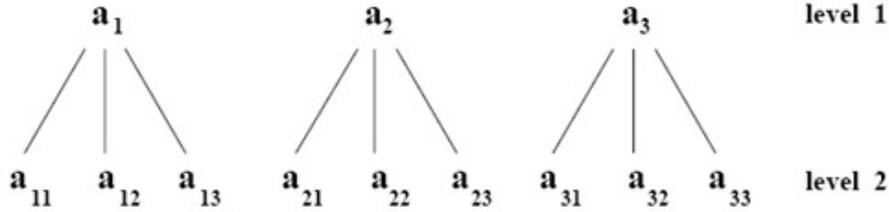
3.6 Structured genetic algorithm

In genetic algorithms, diversity of population is the key point to keeping a robust search. When the diversity is not sufficient, premature convergence will occur. This causes the search process to come to a halt before a true optimum is found. Furthermore, genetic operators may not work effectively due to a lack of genetic diversity. For example, crossover will not work well because of the homogeneity in the population. This makes offspring about the same. The simplest method to enhance the diversity is increasing the number of individuals. If the number of individuals in a population is too small, after the initial generation, the diversity will decrease through a select scheme. This will lead to an evolution process slowdown and reduced improvement. But increasing the number of individuals will increase the calculation time, too.

A structured genetic algorithm is a multi-layered structure for the chromosomal representation that allows multiple bit changes to occur simultaneously at a different level [24]. This leads to a large variation in the chromosome with a greater probability of maintaining high viability. Therefore, it is able to function well in many complex real world environments. On the other hand, in most cases of the motif finding problem, the

motif length is usually fixed input. The structured genetic algorithms can search for motifs of varying lengths, converging to motifs of the size most represented in the search space [23]. This is different from other methods in motif finding problem.

The central feature of the structured genetic algorithm is using the concept of multi-level chromosomal structures. High level genes activate or deactivate sets of lower level genes. Fig. 10 (a) shows the architecture of the two-level representation [25]. Level 1 is the activation level and level 2 is the expression level. The bit at a high level is like a switch to turn on the bit at a low level. The bits which are not active do not disappear but remain in the chromosome structure. The encoding form is shown in Fig.10 (b). The chromosome is divided into two parts; on the left side are high level bits and on the right side are low level bits.



(a) A 2-level structure of sGA.

(a₁ a₂ a₃ a₁₁ a₁₂ a₁₃ a₂₁ a₂₂ a₂₃ a₃₁ a₃₂ a₃₃) -a chromosome
 and
 (0 1 0 1 0 0 1 1 0 0 0 1) - a binary coding

(b) An encoding process of sGA.

Fig. 10. (a) This is the multi-layered structure of the representation. (b) A flattened linear representation encoding form is shown.

In this thesis, a two-level structured chromosome is used. Each bit in level 1 contains two bits in level 2 because the nucleotides are encoded as two bits. The chromosome is of the form as follows:

$$A = (S_1, S_2) = ([a_i], [a_{ij}]). \quad (3.1)$$

A represents an ordered set containing string S_1 in level 1 and S_2 in level 2. The bits in chromosome A are not all interpreted as the traditional genetic algorithms are, but interpreted according to the high level bits. B is the interpreted string, and is constructed from A by joining each a_{ij} together when $a_i = 1$. In other words, B is not a fixed length string. Although chromosome A does all genetic operators, B is the interpreted string to express the predicted motif. This is the reason that structured genetic algorithms have the ability to search for motifs of varying lengths. B will be decoded by the encoding table. Each two bit expression block encodes a letter of the nucleotide alphabet. For example, if $B = 0011011010$, then B will be expressed as ATGCC. Besides the representation of chromosomes, the other part of a structured genetic algorithm is similar to that of a traditional genetic algorithm.

3.7 Combine Gibbs sampler with genetic algorithm

In chapter 2, some methods for the motif finding problem are described, and Gibbs sampler is one of them. The basic concept of the Gibbs sampler method is to guess systematically. But in the Gibbs sampler method, there are two disadvantages. First, the initial positions of each sequence are randomly chosen. This makes the Gibbs sampler very uncertain to converge to a better solution. Second, the method calculates the score at each position in step 4. In fact, most positions do not provide any help in finding a better convergent solution. Therefore, calculation of the scores at most positions is worthless. In this section, we propose a procedure which combines Gibbs sampler method with a genetic algorithm to overcome these weaknesses and improve the performance [18].

After the work of the genetic algorithm, the individual with the highest total fitness score is determined. Because this individual has the highest score, it is valuable to provide some reference positions to help Gibbs sampler method work better. This individual has different scores at every position of all sequences. Take k sets of positions with higher scores in each sequence to be the better candidate position sets. In the better candidate position sets, take the best set as the initial positions in Gibbs sampler method. Then, follow steps 2~5 in Gibbs sampler method. But there is a small difference from the original method in step 4. In the original step 4, scores at all possible positions of sequence S are calculated, but now only k sets of candidate positions are calculated. If the sequence length is very long, the time saved is considerable. Steps 2~5 are executed repeatedly until the stop criterion is reached. The architecture of this procedure is illustrated in Fig. 11.

Most time spent in Gibbs sampler is for scores that are calculated at every position in the selected sequence S . This combination could reduce the calculation significantly. In the original step 4, sequence S needs to check the scores $(n - l + 1)$ times, but now it only needs to check k times because k sets of candidate positions are maintained from the genetic algorithm. This reduces the calculation time indeed. In addition, the initial positions guessed randomly in Gibbs sampler method make it hard to converge. The genetic algorithm provides a better set of initial positions for Gibbs sampler method. In chapter 2, it is mentioned that the initial positions are important because they will affect the performance directly. Hence, this combination does make sense.

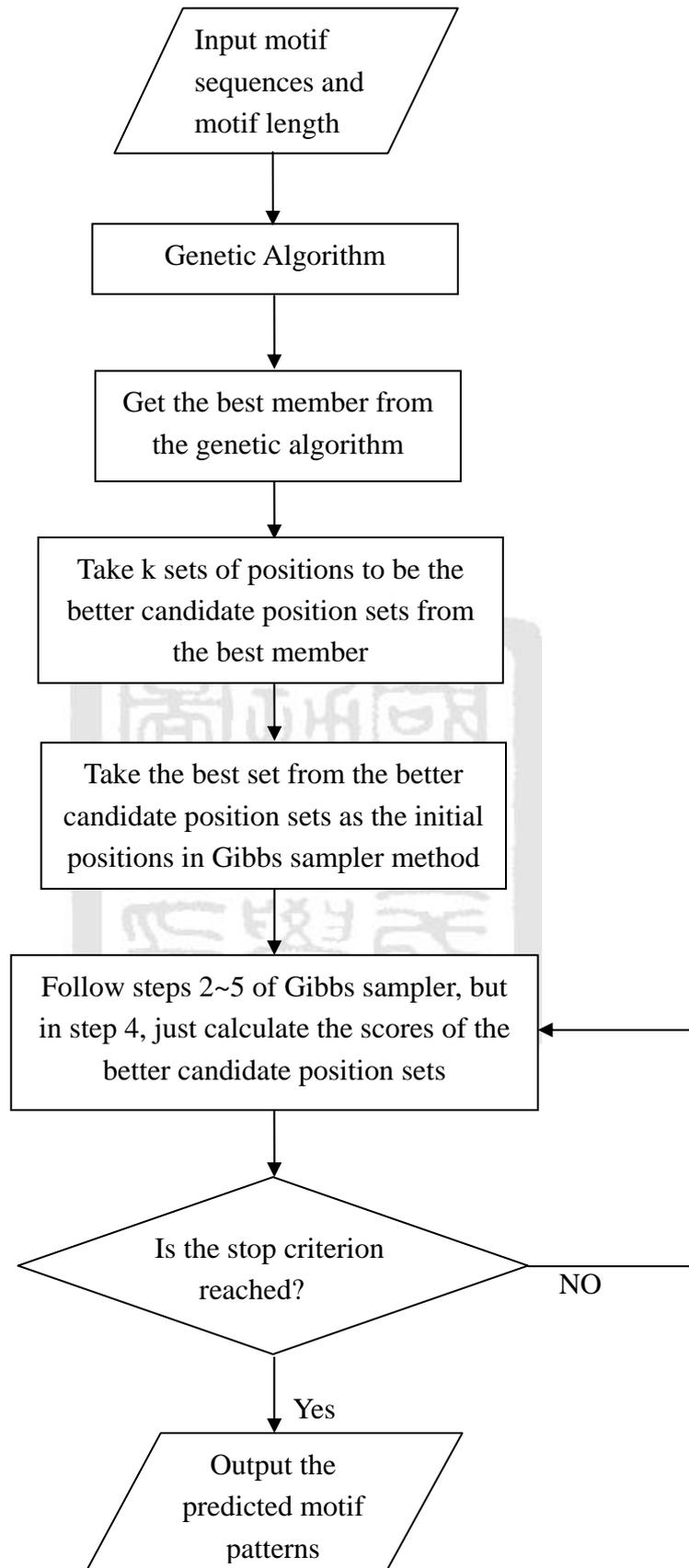


Fig. 11. This procedure is to combine the Gibbs sampler with the genetic algorithm.

4.1 Experimental results

The methods stated in chapter 3 were tested on a data set of 10 randomly generated independent identically distributed (i.i.d.) sequences with an inserted motif in each sequence. These sequences were tested for 100, 300, 500bp in length, respectively. Every sequence was inserted with a designed motif (AA...A) of 10 or 15 in length because motifs typically vary in length from 6-15 base pairs in length [23]. The purpose of this experiment was to find the inserted motif. The insertion positions were random in each sequence. The test method is as follows [11], [23]. The reason that the inserted motif is not generated randomly but designed as a concatenated string with the same letter is that the probability of this motif happening occasionally is small. The sequences were randomly generated, but the same letter occurring fifteen times consecutively is almost impossible. Hence, the probability that some fragments in the sequence are similar to the inserted motif is small. If the inserted motif is also randomly generated, it will have many disturbances in finding the inserted motif. The designed motif is in order to avoid confusing the fragment generated randomly in the sequence with the inserted motif.

The equipment for testing is stated as below. The hardware environment in this thesis for testing programs is a PC with AMD CPU 1.61 GHz and RAM 256 MB. The software environment is based on Windows XP with MATLAB.

4.1.1 Genetic algorithm

In this experiment, the length of the inserted motif is 10 and the sequence length is 100. The genetic algorithm was tested with different individual numbers and generation numbers. The elapsed time and hit percentage averaged across 100 runs are shown in Table 4. In Tables 5 and 6 are the results tested in the same condition but the sequence length is 300 and 500, respectively. In Table 7, the length of the inserted motif is 15.

Table 4. The results tested with motif length 10 and sequence length 100

Individual number	Generation number	Elapsed time (seconds)	Hit percentage
20	30	3.78s	81%
	50	6.23s	85%
40	30	7.49s	99%
	50	12.17s	93%

Table 5. The results tested with motif length 10 and sequence length 300

Individual number	Generation number	Elapsed time (seconds)	Hit percentage
20	30	12.45s	62%
	50	19.35s	65%
40	30	22.51s	69%
	50	38.09s	71%

Table 6. The results tested with motif length 10 and sequence length 500

Individual number	Generation number	Elapsed time (seconds)	Hit percentage
20	30	18.16s	57%
	50	31.29s	55%
40	30	37.95s	58%
	50	62.03s	65%

Table 7. The results tested with motif length 15

Individual number	Generation number	Sequence length	Elapsed time (seconds)	Hit percentage
40	30	100	10.67s	96%
		300	31.28s	72%
		500	52.87s	42%

In the simulation results of the genetic algorithm in Tables 4~6, we can see that the hit percentage is higher if the individual number is bigger. According to the observation of the evolution of the scores, the scores usually stop increasing during generations 30~40. The scores do not change any more when the generation number is over 50. Hence, in the following simulations, we set the individual number at 40 and the generation number at 30.

According to the result in Table 7, it is obvious that the hit percentage is worse if the sequence is longer. When the inserted motif length is 10 and the sequence length is 500, the hit percentage is 65%. However, when the inserted motif length is 15 and the sequence length is 500, the hit percentage drops to 42%.

4.1.2 Genetic algorithm vs. Structured genetic algorithm

In this experiment, the comparison between the genetic algorithm and the structured genetic algorithm is shown. The condition for this experiment is that the length of the inserted motif is 10, the individual number is 40, the generation number is 30, and the sequence length is 100. Elapsed time and hit percentage averaged across 100 runs are shown in Table 8. In Tables 9 and 10 are the results tested in the same condition but the sequence length is 300 and 500, respectively.

Table 8. The results tested with motif length 10 and sequence length 100

Algorithm	Elapsed time (seconds)	Hit percentage
Genetic algorithm	7.49s	99%
Structured genetic algorithm	8.19s	83%

Table 9. The results tested with motif length 10 and sequence length 300

Algorithm	Elapsed time (seconds)	Hit percentage
Genetic algorithm	22.51s	69%
Structured genetic algorithm	22.56s	60%

Table 10. The results tested with motif length 10 and sequence length 500

Algorithm	Elapsed time (seconds)	Hit percentage
Genetic algorithm	37.95s	58%
Structured genetic algorithm	38.74s	33%

Regarding the structured genetic algorithm, in Tables 8~10, the elapsed time is a little more than that of the genetic algorithm and the hit percentage is worse than that of the genetic algorithm. The range of the predicted motif length is from 13 to 15. From the simulation, we come to the conclusion that although the structured genetic algorithm can search for motifs of varying lengths, its performance is worse than that of the genetic algorithm.

4.1.3 Gibbs sampler method

In this experiment, the length of the inserted motif is 10. The Gibbs sampler method was tested with different iteration numbers. The elapsed time and hit percentage averaged across 100 runs with sequence length in 100, 300, and 500 are shown, respectively, in Table 11. In Table 12 are the results tested in the same condition but the length of the inserted motif is 15.

Table 11. The results tested with motif length 10

Sequence length	Iteration	Elapsed time (seconds)	Hit percentage
100	300	0.84s	67%
	500	1.25s	73%
300	300	1.89s	56%
	500	2.94s	41%
500	300	3.09s	22%
	500	4.78s	23%

Table 12. The results tested with motif length 15

Sequence length	Iteration	Elapsed time (seconds)	Hit percentage
100	300	1.02s	55%
	500	2.37s	64%
300	300	2.5s	36%
	500	5.3s	41%
500	300	4.02s	18%
	500	6.06s	20%

In Tables 11~12, we can see that the elapsed time of Gibbs sampler is much smaller than that of the genetic algorithm. When the sequence length is 100 and 300, the hit percentage is acceptable. But when the sequence length is 500, the hit percentage is bad. This shows that although the elapsed time of the Gibbs sampler method is shorter, when the sequence length is long, the performance is not good. The results show that the genetic algorithm is better than Gibbs sampler method.

4.1.4 Gibbs sampler method with genetic algorithm

In this experiment, the length of the inserted motif is 15. The Gibbs sampler method with genetic algorithm was tested with 300 iterations. The condition of the genetic algorithm is that the individual number is 40 and the generation number is 30. Elapsed time and hit percentages averaged across 100 runs, with sequence length in 100, 300, 500, and 1000 shown respectively in Table 13. With different sequence lengths, there are different k values to be tested. The elapsed time column shows the total time spent in the genetic algorithm and Gibbs sampler. Table 14 shows the comparison of elapsed time between using the Gibbs sampler method with genetic algorithm and using the Gibbs sampler method only.

Table 13. The results tested with motif length 15

Sequence length	k	Elapsed time in genetic algorithm (seconds)	Elapsed time in Gibbs sampler (seconds)	Elapsed time totally (seconds)	Hit percentage
100	30	11.64s	0.40s	12.04s	91%
	50	11.49s	0.49s	11.98s	92%
300	100	33.83s	1.02s	34.85s	63%
	200	34.99s	1.58s	36.57s	70%
500	200	55.92s	1.85s	57.7s	50%
	300	61.39s	2.33s	63.72s	50%
1000	500	113.87s	4.12s	117.99s	20%

Table 14. The results tested with motif length 15

Sequence length	Algorithm	Elapsed time in Gibbs sampler (seconds)	Hit percentage
100	Gibbs sampler with GA	0.49s	92%
	Gibbs sampler	1.02s	55%
300	Gibbs sampler with GA	1.58s	70%
	Gibbs sampler	2.5s	36%
500	Gibbs sampler with GA	2.33s	50%
	Gibbs sampler	4.02s	18%

In the last simulation using a genetic algorithm with Gibbs sampler, we can see in Table 14 that the time spent in the Gibbs sampler is indeed shorter than the time spent using only Gibbs sampler method without a genetic algorithm. But in Table 13 we can see that the total time spent is more than that of the genetic algorithm. In Table 14 shows that the hit percentage of using Gibbs sampler method with a genetic algorithm is better than that only using Gibbs sampler. This is because the good performance of the genetic algorithm leads to the Gibbs sampler converging to better solutions.

On the other hand, taking k sets of positions with higher scores in each sequence as the better candidate position sets in Gibbs sampler makes the choices of position limited. If k is too small, the algorithm may miss some positions that are supposed to be the solution.

Table 15. The results tested with the motif length as 15 and the sequence length as 1000

Algorithm	Elapsed time (seconds)	Hit percentage
Gibbs sampler with GA	117.99s	20%
Gibbs sampler	7.49s	<5%
Structured genetic algorithm	104.13s	<5%
Genetic algorithm	109.06s	<5%

We can see in Table 15 that when the sequence length increases to 1000, the results of the Gibbs sampler, structured genetic algorithm, and genetic algorithm are poor. The hit percentage of using the Gibbs sampler with genetic algorithm is only 20%. Although the results of our approach are better than other three methods, the results are not good enough. Hence, the action in which a genetic algorithm provides a better position set to the Gibbs sampler does not work well.

5.1 Conclusion and contribution

According to the simulation results, our conclusions are as follows. The genetic algorithm performs better than the structured genetic algorithm and Gibbs sampler method. The Gibbs sampler method performs worst but spends less time in computing. Combining the genetic algorithm with the Gibbs sampler method makes the performance better than using the Gibbs sampler only. The time spent in the Gibbs sampler is indeed shorter than the time spent using only the Gibbs sampler method without a genetic algorithm.

The contribution of this thesis is as follows:

- Combining the genetic algorithm with the Gibbs sampler method makes the time spent in the Gibbs sampler indeed shorter than that using Gibbs sampler method only.
- Combining the genetic algorithm with the Gibbs sampler method makes the performance better than using Gibbs sampler only. This is because the robustness of the genetic algorithm leads to the Gibbs sampler converging to better solutions.
- When the tested sequence length is long, the performance of other three methods is poor but the performance of our approach is still 20%.

5.2 Future work

First, we focus on the genetic algorithm. During the simulation process, we have several parameters such as individual number, generation number, crossover rate, and mutation rate to adjust. In this thesis, we use floating regulation for crossover and mutation rate. Applying more systematic algorithms to parameter optimization could be a future effort to make this method work better.

As regards the fitness scoring function, we add scores for matched letters in this thesis. However, some methods were mentioned where scoring with penalty is another direction to try. The fitness function regulation such as linear scaling is also a good direction to reform, but this needs more theoretical background knowledge to do the research.

In this thesis, we do the simulations for DNA sequences with two bits to encode the four letters. If we want to do the simulations for proteins, the encoding method has to change. Because proteins are made up of 20 different kinds of amino acids, two bits to encode 20 letters are not enough. It needs at least five bits to encode. The scoring function should change, too. Similarity matrices such as PAM, BLOSM could be the basis to score.

Second, we focus on the structured genetic algorithm. During the simulation process, we found that although the structured genetic algorithms can search for motifs of varying lengths, there were cases in which detected motifs with zero length sometimes occurred. But the probability of this case is very small. We should develop a way to deal with this situation.

References

- [1] F. Crick, "Central Dogma of Molecular Biology," *Nature*, vol. 227, pp. 561-563, Aug 1970.
- [2] Y. Moreau, F. de Smet, G. Thijs, K. Marchal and B. de Moor, "Functional Bioinformatics of Microarray Data: From Expression to Regulation," *Proceedings of the IEEE*, vol. 90, no. 11, Nov 2002.
- [3] F. F. M. Liu, J. J. P. Tsai, R. M. Chen, S.N. Chen and S.H. Shih, "FMGA : Finding Motifs by Genetic Algorithm," *Proceedings of the Fourth IEEE Symposium on Bioinformatics and Bioengineering (BIBE'04)*, pp. 459-466, May 2004.
- [4] G. D. Stormo, "DNA binding sites: representation and discovery," *Bioinformatics*, vol. 16, no. 1, pp. 16-23, 2000.
- [5] C. E. Lawrence, S. F. Altshul, 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*, vol. 262, pp. 208-214, 1993.
- [6] T. L. Bailey, C. Elkan, "Fitting a mixture model by expectation maximization to discover motifs in biopolymers," *Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology*, vol. 2, pp. 28-36, 1994.
- [7] G. Z. Hertz and G. D. Stormo, "Identifying DNA and protein patterns with statistically significant alignments of multiple sequences," *Bioinformatics*, vol. 15, no. 1, pp. 563-577, 1999.
- [8] J. Buhler and M. Tompa, "Finding Motifs Using Random Projections," *Journal of Computational Biology*, vol. 9, no. 2, pp. 225-242, 2002.
- [9] E. Eskin and P.A. Pevzner, "Finding Composite Regulatory Patterns in DNA Sequences," *Bioinformatics*, vol. 18, no. 1, pp. 354-363, 2002.
- [10] U. Keich and P.A. Pevzner, "Finding Motifs in the Twilight Zone," *Bioinformatics*, vol. 18, no. 10, pp. 1374-1381, 2002.

- [11] A. Price, S. Ramabhadran and P. A. Pevzner, "Finding subtle motifs by branching from sample Strings," *Bioinformatics*, vol. 1, no. 1, pp. 1-7, 2003.
- [12] B. Raphael, L. T. Liu and G. Varghese, "A Uniform Projection Method for Motif Discovery in DNA Sequences," *IEEE Transactions on Computational biology and Bioinformatics*, vol. 1, no. 2, pp. 91-94, 2004.
- [13] D. Liu, X. Xiong, B. DasGupta and H. Zhang, "Motif discoveries in unaligned molecular sequences using self-organizing neural networks," *IEEE Transaction on Neural networks*, vol. 17, no. 4, July 2006.
- [14] 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*, vol. 223, pp. 159-70, 1992.
- [15] 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, and Genetics*, vol. 7, iss. 1, pp.41-51, Feb 2004.
- [16] P. A. Pevzner and S. H. Sze, "Combinatorial approaches to finding subtle signals in DNA sequences," *Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology*, pp. 269-78, 2000.
- [17] N. C. Jones and P. A. Pevzner, *An Introduction to Bioinformatics Algorithms*, MIT Press Cambridge, Mass, 2004.
- [18] Y. J. Liao, "Motif finding in biological sequences", M.S. thesis, Department of Computer Science and Engineering, National Sun Yat-sen University, Taiwan, 2003.
- [19] J. H. Holland, "Genetic algorithms," *Scientific American*, vol. 267, pp. 44-50, 1992.
- [20] D. E. Goldberg, *Genetic algorithms in Search, optimization and machine learning*, Kluwer Academic Publishers, Boston, MA, 1989.

- [21] P. J. Fleming, R. C. Purshouse. "Evolutionary algorithms in control systems engineering: a survey." *Control Engineering Practice*, vol.10, pp. 1223-1241, 2002.
- [22] A. J. F. Griffiths et al, *An Introduction to Genetic Analysis*, W. H. Freeman, New York, seventh edition, 2000.
- [23] M. Stine, D. Dasgupta and S. Mukatira, "Motif Discovery in Upstream Sequences of Coordinately Expressed Genes", *Evolutionary Computation of the IEEE*, vol. 3, pp. 1596-1603, Dec 2003.
- [24] D. Dasgupta and D. R. McGregor, "Non-stationary function optimization using the structured genetic algorithm," *Proceedings of the Second International Conference on Parallel Problem Solving From Nature (PPSN-2) Conference*, pp. 145-154, Sep 1992.
- [25] D. Dasgupta and D. R. McGregor, "sGA: A structured genetic algorithm," Technical Report IKBS-8-92, University of Strathclyde, 1992.
- [26] M. T. Lee, "Motif finding", class notes for GCB 535 / CIS 535, Department of Computer and Information Science, University of Pennsylvania, 10 Oct 2004.