APPLICATION OF HEURISTIC METHODS IN FINDING PATTERNS COMMON TO GROUPS OF BIOLOGICAL SEQUENCES

YANG LIANG

(B.Comp.(Hons.), NUS)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2006

Acknowledgements

I would like to express my deep gratitude to all those who gave me the possibility to complete this thesis. I am deeply grateful to my supervisor, Professor Vladimir. Bajic, Head of the Knowledge Extraction Lab, Institute for Infocomm Research. His wide knowledge and creative way of thinking have been of great value for me. His simulating suggestions and encouragement helped me in all the time of the research and the writing of the thesis.

I am deeply indebted to my co-supervisor Prof. Ashraf Kassim, Vice-Dean of Electrical and Computer Engineering department of National University of Singapore, who kept an eye on the progress of my research. As a vice-dean of the department, he is very busy with his work, but he is always available whenever I need his advises. He also helped me a lot in editing the thesis. Without Prof. Kassim‘s supervision, it is definitely not possible for me to complete the master project smoothly.

I would like to thank all the members in Knowledge Extraction Lab for their help, support and interest in my research work. First, I would like to express my warm and sincere thanks to Dr. Rajar, who spent a lot of his personal time in helping me modifying the previous version of the thesis and gave me a lot of wonderful suggestions to make this final version possible. Also, I am grateful for Sinlam and Suisheng, who taught me a lot of bio-informatics knowledge. I also would like to thank Enli, Zhuo Zhang, Edward, Vidhu and Manisha for all their valuable hints to my research. I thank you all.

Table of Contents

Summary

In this project, we studied a motif discovery problem, which relates to extracting the conserved patterns from a set of unaligned DNA sequences to predict the Transcription Factor Binding Sites (TFBSs). This problem is NP-hard (Non-polynomial time solvable), and currently, there is no best solution algorithm for it. Although there are many surveys and algorithms for motif discovery problem, the problem is still far from being solved and most of the algorithms can only provide the local optimal solutions.

We provide three heuristic algorithms for this motif discovery problem (Huang *et al*. 2005, Yang *et al*. 2005, Bajic *et al*. 2004). These algorithms have the ability to escape from the local optimum and search for the global optimal solutions. They are based on three existing heuristic algorithms, which are Simulated Annealing, Tabu Search and Genetic Algorithm. The algorithms and the program structures are presented in detail in the thesis. At the same time, a web-accessible motif search tool is also implemented based on our three heuristic algorithms, and is free for academic users at http://sdmc.i2r.a-star.edu.sg/DRAGON/Motif_Search/ (Yang *et al*. 2005).

From the comparison result with the other existing well-known motif discovery algorithms using Tompa's benchmark dataset, we conclude that Simulated Annealing, Tabu Search and Genetic Algorithm are very useful to the motif discovery problem. They perform much better than those existing algorithms in terms of sensitivity; they also perform better than several existing algorithms based on some other measures of prediction success. However we get low positive predictive values for our algorithms. That is because some of the authors of Tompa's study have done manual elimination of predictions they considered not good. This process is not explained in any detail and

thus makes not possible to compare our ‘raw’ results with theirs. We believe that if such manual cleaning of predictions is made, our results could well be somewhere in the upper group of better performing predictors. As it stands now, the results are definitely better than at least one of the existing algorithms, which is QuickScore.

By analyzing the results, we also discovered that the consensus approaches perform much better than profile model approaches at least in terms of sensitivity. The definition of these two types of approaches is given in Section 1.4 of the main text. From the different definitions of the similarity between the two motifs given by these two types of approaches, we conclude that the motifs in one group found by consensus approaches are much more compact than that found by profile model approaches.

To increase the accuracy of finding the TFBSs of our algorithms, some further work can be done. We suggest introducing some biological features into the algorithms to filter out the false predictions: a) DNA has a double-stranded structure, where one strand complements the other; b) the specific binding sites could be located in the same region in promoters; c) the binding sites in the sequences could be located in the same order; d) compact motif group should have a high Information Content.

List of Publications

[i] Huang E, Yang L, Chowdhary R, Kassim A, Bajic V.B., An algorithm for *ab initio* DNA motif detection, Chapter 4 in *Information Processing and Living Systems*, *World Scientific*, 611-614, 2005

[ii] Yang L, Huang E, Bajic V.B., Some implementation issues of heuristic methods for motif extraction from DNA sequences, *Int.J.Comp.Syst.Signals*, 5(2) (in print) (2005)

[iii] Bajic V.B., Huang E, Yang L, Modeling methodology for detection of regulatory motifs in DNA/RNA and proteins, *Int.J.Comp.Syst.Signals,* (accepted) 2004

[iv] Krishnan SPT, E Huang, L Yang, V B Bajic, Statistical Properties of region around PolyA sites in Human, *5th HUGO Pacific meeting and 6th Asia Pacific meeting on Human genetics,* 17-20 November 2004, Singapore.

List of Tables

List of Figures

List of Abbreviations

DMB	Dragon Motif Builder
EM	Expectation Maximization
GA	Genetic Algorithm
IC	Information Content
llr	log likelihood ratio
NP	Non-Polynomial
PSPM	Position-Specific letter-Probability Matrix
PWM	Position Weight Matrix
SA	Simulated Annealing
TF	Transcription Factor
TFBS	Transcription Factor Binding Site
TS	Tabu Search

Chapter 1 Introduction

Bioinformatics is the field of science that uses computers to store, retrieve and analyze biological information for the purpose of predicting the function, structure or composition of biomolecules (http://bioinformatics.org). Using bioinformatics, the biological research can be accelerated and enhanced. "Biomolecules" include genetic material---nucleic acids---and the products of genes: *proteins*. "Classical" bioinformatics deals primarily with sequence analysis. There are three important sub-disciplines within bioinformatics such as:

a) analysis and interpretation of various types of data including nucleotide and amino acid sequences, protein domains, protein structures, etc.;
b) development and implementation of tools that enable efficient access and management of different types of information;
c) development of new algorithms and statistics to access relationships between members of large data sets.

This project falls under the first and second categories. It deals with identification of a set of short sequence *motifs* that are mutually very similar and which may be common for many members of specific sequence sets. Such motifs frequently have strong biological relevance, for example they may represent binding sites of regulatory proteins. To efficiently determine such motifs, some biological features and their representation are introduced into specific existing meta-heuristic algorithms. Moreover, the project is typically related to transcription regulation and identification of regulatory elements in DNA. Thus, we focus our attention on this class of problems.

1.1 Gene Regulation

Molecular processes within cell are controlled in many ways and gene regulation is one of the principal mechanisms utilized in cells (Latchman 2002). Genes are selectively expressed in time and in different cells. Only a subset of genes in the genome is transcribed at a given time in specific cells under specific conditions. This is accomplished through interaction of binding regulatory proteins (e.g. *transcription factors, TFs*) to the DNA regulatory sites (e.g. *TF binding sites, TFBSs*). These sites usually are located in the regions called *promoters*. When genes are controlled by a similar set of TFs, we usually call them co-regulated. For this reason, the promoters of co-regulated genes normally contain and share conserved DNA sequence patterns called *motifs*. The sequence patterns corresponding to a motif are called *instances* of that motif. To identify motifs and their corresponding instances is very important in biology research. Many biological sequences belonging to a group of functionally related genes or proteins usually contain a number of biologically active sequence patterns shared among many and sometimes all members of the functional group.

DNA motifs that we consider are usually not very long and extend at most up to 30 nucleotides without gaps. However, promoters containing the motifs are long (usually from several hundreds to 2000 nucleotides, and sometimes even more). Every instance of a motif normally has the same length, but they could be different in composition. Such motifs could be also determined by experimental approaches such as gel-shift analysis, DNA footprinting or Chip-CHIP (Liu 2004). However, such biological experiments are tedious and require a long time to get the result. For this reason the use of computer programs to identify such TFBSs are a good alternative to experimental methods.

1.2 Motif Discovery Problem and Its Motivation

Common motifs shared by different DNA sequences frequently have relevant biological interpretation. A typical example represents promoters of a group of co-expressed genes that contain many common transcriptional regulatory elements, which also share similar positional organization such as order and mutual distances of transcriptional elements (Werner 1999).

There are several surveys (Brazma *et al*. 1998, Brejova *et al*. 2000, Rigoutsos *et al*. 2000, Sinha *et al*. 2000, Sinha *et al*. 2003, Tompa *et al*. 2005) related to motif identification problems, that discuss several algorithms used for this purpose. These algorithms usually produce mutually quite different results. This is not necessarily a bad thing as this may be useful for the users to make selections to suite their need most appropriately. However, most of the existing algorithms can only provide local optimum, and there are still a lot of unsolved technical problems in this computation biology problem.

As said, although, a lot of work has been done for such motif discovery problems, these problems are still far from being solved. In the research area of computer algorithm, these problems are defined as NP-hard (Non-polynomial time solvable). *Brute force* algorithm can be developed to search for the optimal solution. The practical algorithms are those that can give good enough solutions within acceptable time. These algorithms are either *greedy* or *heuristic*. All of the existing approaches for the motif discovery problems belong to these two categories.

However, none of them is good for all types of dataset. Some algorithm may be the best for some dataset, but it may not be suitable for the others. These problems are still interesting many bioinformatics researchers. Developing new algorithms based on the features of the problems is a good idea. On the other hand, introducing the biology features to the existing algorithms is also a good try, especially for those NP-hard problems. There are some heuristic algorithms such as Simulated Annealing (SA) (Booker 1987, Dowsland 1993, Eglese 1990, Fleischer 1995, Ingber 1993, Ingber 1996, Johnson *et al*. 1989, Kirkpatrick *et al*. 1983, Tovey 1988), Tabu Search (TS) (Glover 1989, Glover 1990, Glover *et al*. 1993, Glover *et al*. 1997, Randall 1999) and Genetic Algorithms (GAs) (Barbulescu *et al*. 2000, Booker 1987, Davis 1991, Denning 1982, Goldberg 1989, Koza 1992, Koza 1994, Mitchell 1996, Mühlenbein 1992, Reeves 1993, Reeves 1997). All of these algorithms have the ability to escape from the local optimum and search for the global optimal solutions. The drawback of these algorithms is lower speed, but consistency of the extracted pattern groups is usually considerably higher than what is obtained with some traditional algorithms such as EM (Bailey *et al*. 1994, Bailey *et al*. 1995b, Lawrence *et al*. 1993) or Gibbs sampling (Casella *et al*. 1992, Favorov *et al*. 2004, Thijs *et al*. 2001). These existing algorithms could be applied in determination of functional patterns in DNA/RNA sequences. From the best of our knowledge, Simulated Annealing, Tabu Search or Genetic Algorithm has not been used in predicting the TFBSs.

1.3 Our Problem

In the project, a motif discovery problem is studied. It relates to extracting the conserved patterns (motifs) from a set of unaligned DNA sequences to predict the TFBSs.

1.3.1 The Definition of Our Problem

In the problem, Hamming distance is used to calculate the distance between two motifs: The number of nucleotides which differ between two motifs. The smaller the distance, the similar the two motifs are.

> The data set represents a collection of DNA sequences. These are given as strings of 5 characters, A, C, G, T, and N. These characters stand for the four bases (adenosine, cytosine, guanine and thymine), and character N indicates that it is not clear which base occupies the given position. Our intention is to extract significant groups of motifs from such collections of sequences. There is a consensus motif for each group. The distance between every motif and the consensus motif of the group should be less than or equal to some user defined threshold.

That means motifs in the same group should share a great level of mutual similarity, but could slightly differ from each other.

1.3.2 Criteria in Selecting Motif Group

Three types of occurrences of a single motif among sequences are considered in the project. The criteria describe the distribution of the motifs among the sequences.

a) One per sequence: Each sequence must contain ONE motif. Only the best motif (motif with the highest similarity score) is chosen. For this criterion, the intention is to find the group of motifs whose total similarity score is the highest. An example is given in Figure 1.1

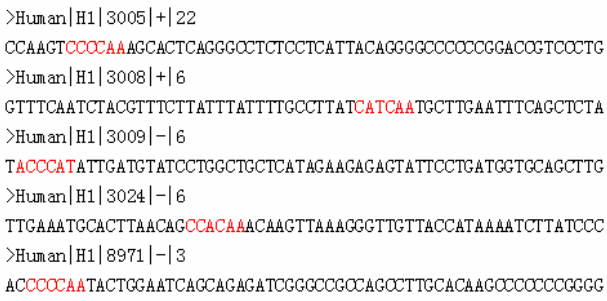

Figure 1.1: Example of 'One per Sequence'

b) Zero or one per sequence: The motif may or may not appear in a sequence. Only ONE best motif can be identified in each sequence. For this criterion, the intention is to find a group of motifs that can cover as many sequences as possible. An example is given in Figure 1.2

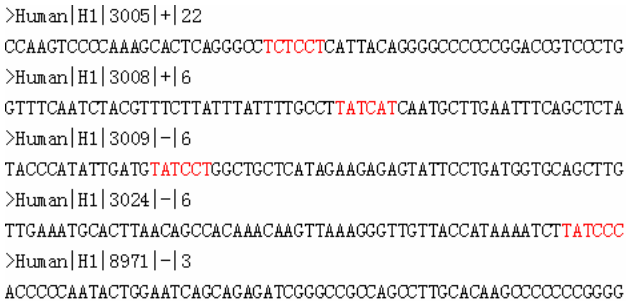

Figure 1.2: Example of 'Zero or One per Sequence'

c) Any number of repetitions: The motifs may or may not appear in the sequences. A sequence may contain one or more motifs. For this criterion, the intention is to find the largest number of motifs that can be grouped together. An example is given in Figure 1.3

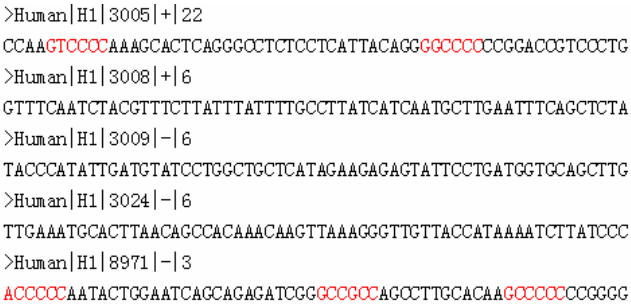

Figure 1.3: Example of 'Any Number of Repetitions'

1.3.3 Methods of Eliminating Used Patterns

Generally, the objective of the algorithms is to extract several groups of motifs from a given sequence set. Since the process sequentially determines the set of motifs that belong to one motif family, and then searches for the next collection of motifs to form another family, we have to make sure not to mix the different motif families in the process of motif identification. A condition has to be satisfied before assigning a motif to any family, which is if the motif already belongs to one motif family no part of it can belong to another motif family. This means that if several motif groups are required in the final search report, those motifs already grouped have to be removed before searching the next motif group. To implement it we provide two methods:

a) Eliminate Motifs Only: The motifs, which were chosen for constructing the previous motif groups, will be excluded in searching for a new motif group.
b) Eliminate Sequence: Those sequences that contain motifs previously identified will be excluded from the search for a new motif group.

1.3.4 Some Restrictions for Motifs within a Group

All the motifs must be selected from the user supplied input sequences. Among those motifs grouped together, there must be a consensus motif, which usually has to be one of the motifs within this group. We have such a condition, because all of our algorithms search for the motif groups reversely. In the algorithms, we assume that a consensus motif has already been found, and the other motifs are selected from the input sequences according to the similarity level by comparing with the presumed consensus motif to build the motif group model (Chapter 3, 4, 5). Sometimes, the motif chosen to be the consensus motif does not appear in the input sequences.

1.3.5 Notations

To present the pseudocode of the algorithms and discuss their time complexity, the following notations are defined:

S_0 = current solution;

S_1 = a neighbor of the current solution;

C(S) = the cost of a solution S;

$\Delta(C) = C(S_1)–C(S_0)$;

l = the number of nucleotides in the chosen pattern;

n = the number of sequences in the input file;

m = the number of nucleotides in each sequence in the input file;

(*l*, *d*) motif model = a motif group inside which all the motifs have the length *l* and the maximum mutual distance is *d*.

1.3.6 Objectives

The objective of the study is to solve the described motif discovery problem by introducing the biological features into there existing meta-heuristic algorithms, which are Simulated Annealing, Tabu Search and Genetic Algorithm.

1. Increase the effect of the algorithms in finding the biological meaningful motifs is one of the goals. By comparing with those well-known motif discovery algorithms, our algorithms should perform better in some of the measurements, such as sensitivity, positive prediction, etc.

2. Increase the efficiency is the other goal. The three algorithms proposed should be able to find the good enough result within acceptable time.

3. With the reasonable short calculation time of the algorithms, a web-based motif search tool can be built, which can be directly applied in determination of functional patterns in DNA and RNA.

1.4 Two Types of Approaches for Motif Discovery Problems

Depending on how the motifs are represented, motif discovery algorithms can be split into two groups: a) those based on motif profiles and b) those based on consensus motifs (Liu 2004, Stormo 2000).

1.4.1 Approaches Based on Profile Motifs

These algorithms use profiles of a set of similar motifs, which is presented as a position-specific letter-probability matrix (PSPM). This matrix describes the probability of each possible letter at each position in the pattern. In these algorithms we assume that the motif alignments contained in the sequence set correspond to the letter distribution that differs most from the background distribution. Therefore, these approaches try to maximize the likelihood ratio of the motif model relative to the adopted background model. Usually, in these algorithms the information content (IC) is used as a score function. Here we define also the other auxiliary scores.

The log likelihood ratio (llr) of a motif is:

$$llr = \log (Pr(sites \mid motif) / Pr(sites \mid background)) \quad (1.1)$$

llr is a measure of how different the sites are from the background model.

Pr (sites | motif) is the probability of the occurrences given the model consisting of the PSPM of the motif. Pr (sites | background) is the probability of the occurrences given the background model.

The IC of the sequence alignment is just a normalized llr:

$$IC = llr/n \quad (1.2)$$

Here, *n* is the length of the motif.

The algorithms used in these approaches are iteratively improved greedy and statistical procedures. The drawback of these algorithms is that they can only provide the local optimum. So, the solutions may be very different for the different selected starting points – the initial guesses of the solution. CONSENSUS (Hertz *et al*. 1999), GibbsDNA (Lawrence *et al*. 1993) and MEME (Bailey *et al*. 1995a) are the most popular programs for profile based motif discovery.

1.4.2 Approaches Based on Consensus Motifs

These approaches attempt to search for a consensus motif first. Then this consensus motif is used to scan the sequences in search for those motifs that are very similar to the consensus motif (similarity can be defined by using some distance measure and threshold). These motifs will be grouped together, and all the motifs within this group will share a high level of mutual similarity. A motif model can be built from such a motif group.

Compared with approaches based on profile motifs, the results obtained from consensus motif approaches are usually much better in the sense that motif groups are more

coherent, i.e. the motifs usually share a greater level similarity. However, these approaches are frequently inefficient in practice as they require an extremely long computation time, such as COPIA (Liang 2001) and GLAM (Frith 2004 *et al*.).

1.5 Organization of the Thesis

In Chapter 2, we discuss the existing algorithms for the motif discovery problem. According to their different motif representation, the algorithms are grouped in two categories. In Chapter 3, 4 and 5, we describe Simulated Annealing, Tabu Search and Genetic Algorithm for the motif discovery problem respectively. For each algorithm, the standard algorithm is first described; and this is followed by a discussion on how the algorithm is applied to the motif discovery problem. In Chapter 6, a motif discovery software with the name Dragon Motif Builder, which is implemented using the provided algorithms, is described. In Chapter 7, we present the experiment results and compare them with the results obtained from other motif discovery algorithms, such as MEME. In the last Chapter, we provide a conclusion and also some suggestions for further work.

Chapter 2 Literature Review

Many approaches have been developed for motif discovery problem. Several computer programs also exist which can be used for this purpose. MEME, GibbsDNA, CONSESUS, GLAM, Improbizer (Ao *et al*. 2004) and QuickScore (Régnier *et al*. 2004) are some of the popular statistical approaches for the profile motif model. WINNOWER, SP_STAR (Pevzner *et al*. 2000), COPIA, PROJECTION (Buhler *et al*. 2001) and probabilistic suffix trees (Eskin *et al*. 2002, Sagot 1998) are approaches for the consensus motif model. In the following sections, we give a brief review of these approaches.

2.1 Profile Model Algorithms

2.1.1 MEME

The expectation-maximization (EM) algorithms have been used in artificial intelligence as a statistical learning technique. EM concept in learning has been originally developed by Lawrence *et al*. (1993). The EM algorithms have certain limitations and the MEME algorithm (Bailey *et al*. 1994, Bailey *et al*. 1995b, Lawrence *et al*. 1990) represents one of its possible extensions. MEME is an unsupervised learning algorithm, which is guaranteed to converge to a local maximum. The MEME implementation relaxes the assumption and allows zero or many occurrences of a motif to be searched for.

MEME is based on the maximum likelihood estimation for fitting the model to the training data. It aims at optimizing the parameters of the model so that the likelihood of the data is maximized through the EM algorithm. Using the initial motif model, EM iteratively improves the model through the expectation step (E-step) and the maximization step (M-

step). The expected value of the log likelihood of the current model parameters over the training data set is determined in the E-step, while in the M-step the parameters of the model are updated.

The implemented version of MEME allows for finding a motif of any length in the pre-specified range. One of the drawbacks of all EM algorithms is that they find only local maximum of the likelihood function. The theoretical time complexity is quadratic in the size of the data set and linear in the length of the motif (O $((nm)^2 l)$), with n, m and l previously defined.

The website of the algorithm is:

http://meme.sdsc.edu/

2.1.2 GibbsDNA

GibbsDNA (Lawrence *et al*. 1993) uses a supervised learning algorithm that assumes that each DNA sequence contains exactly one motif instance of fixed length. It attempts to maximize the similarity among the motifs in a family. The best motif on each sequence is the one that maximizes the ratio of the corresponding pattern probability relative to the background probability, which is expressed as the log likelihood ratio. The algorithm uses Gibbs sampling to random seeds and takes DNA structure and constraints into account.

GibbsDNA first randomly selects an initial position on each sequence where the assumed motif is; then it repeats the iterative improvement of the motif model family through the “predictive update step” and the “sampling step”. In the predictive step, one motif is deleted and the truncated motif model is determined. In the sampling step, the

selected motif is deleted and the remaining sequence is examined so that the algorithm determines the likelihood of every motif found in the remaining sequence to fit the motif model. Then, it selects the one with the highest likelihood and adds to the truncated model.

This algorithm suffers from the large space of starting positions that impacts on the time required for the algorithm to end. Also, it may never reach the global optimum. The time complexity is proportional to the number of iterations before converging.

The website of the algorithm is:

http://rulai.cshl.edu/people/ioschiks/gibbsDNA/

2.1.3 CONSENSUS

CONSENSUS (Hertz *et al*. 1999) algorithm is based on position weight matrices for pattern discovery in DNA or protein sequences. It uses greedy multiple alignments aiming at a motif alignment that maximizes the information content of the model. CONSENSUS first randomly selects one sequence as start sequence, and extracts subsequences with fixed length *l* as single pattern motifs; then it attempts to select the best motif model through the top *Q* (where *Q* is a user-designated parameter, the default *Q* in CONSENSUS is 1000) pair-wise pattern similarities between this start sequence and one of the remaining sequences; then it iteratively assembles the top *Q* motifs into multiple similarities by adding more and more pattern instances from different sequences with a greedy selection algorithm. The problem with this algorithm is that it is dealing with two sequences at a time, thus making locally optimal selection at each of the steps without having insights how this reflects to the rest of sequences. The time complexity of this algorithm is $O(nm^2 + Qn^2ml)$, where *m* is the average length of sequences.

The website of the algorithm is:

http://bifrost.wustl.edu/consensus/

2.1.4 GLAM

GLAM (Frith *et al*. 2004) is based on Gibbs sampling and also automatically optimizes the alignment length and evaluates the statistical significance of its output. The algorithm is searching for the motifs by obtaining the best possible alignments without gaps of multiple sequence segments. The 'best' alignment is used to select the motifs. Maximally one segment from each sequence is included in the alignment. If the alignment is better without the sequence, such a sequence may be excluded. Because the algorithm cannot find multiple motif instances in one sequence, long sequences can be fragmented into shorter ones with the alignment transformed to a weight matrix and used to scan the sequences to obtain the final motif predictions.

The website of the algorithm is (currently there is no web server):

http://zlab.bu.edu/glam/

2.1.5 Improbizer

Improbizer (Ao *et al*. 2004) utilizes a version of EM algorithm and determines for a collection of DNA motifs the position weight matrices that characterize the collection. The concept is based on using the enrichment of the motifs in the input data as opposed to certain background. The background can be up to a second-order Markov model randomly generated. As an option, it may construct a Gaussian model of motif placement, so that motifs that occur in similar positions in the input sequences are more likely to be found.

The website of the algorithm is:

http://www.soe.ucsc.edu/~kent/improbizer

2.1.6 QuickScore

QuickScore (Régnier *et al*. 2004) uses an exhaustive search to estimate probabilities of rare or frequent words in genomic sequences. It is based on an extended consensus allowing well defined mismatches. It calculates z-scores and P values, depending on the statistical models used, such as the Bernoulli model and the Markov model.

The website of the algorithm is:

http://algo.inria.fr/dolley/QuickScore/

2.1.7 AlignACE

AlignACE (Aligns Nucleic Acid Conserved Elements) (Hughes *et al*. 2000) finds sequence motifs given as position weight matrices. These motifs are presumed to be conserved in a set of DNA sequences. Using iterative masking allows multiple distinct motifs to be found within a single data set. AlignACE is based on a Gibbs sampling and the quality of alignments is estimated using a maximum *a prior* log-likelihood score that reflect the overrepresentation of the motifs.

The website of the algorithm is:

http://atlas.med.harvard.edu/

2.2 Consensus Model Algorithms

2.2.1 WINNOWER

WINNOWER (Pevzner *et al*. 2000) converts a multiple local alignment problem into a maximal clique search problem in a multipartite graph and attempts to solve the clique problem by filtering. If the input sequences are provided it attempts to find the (*l*, *d*) motif model (the motif length is *l*, and each motif can be constructed from the consensus motif by changing at most *d* nucleotides). It constructs the graph *G* as follows: Every vertex in *G* corresponds to a length-*l* motif; two motifs in different sequences are connected by an edge if their distance is at most 2*d*. *G* is an m-partite graph, where m is the number of sequences in the input dataset. The original problem reduces to finding the largest clique. It should be mentioned; however, that search for cliques is an NP-hard problem. WINNOWER removes edges that are definitely not contained in a large clique. Then it attempts to filter out all spurious edges iteratively: a) filter weak vertices, which are vertices not supported by a neighbor in every part of G; b) filter weak edges, which are unsupported edges; c) filter such weak triangles. In the general case, the time complexity of this approach is (O $(nm)^{k+1}$), which is very consuming. Another disadvantage of this approach is that it treats all edges of the graph *G* equally without distinguishing between edges corresponding to high and low similarities.

2.2.2 SP-STAR

SP-STAR (Pevzner *et al*. 2000) treats every *l*-mer that appears in the input sequences as a potential consensus motif. For each l-mer in the sample it finds its best instance in each sequence and collects these instances to form an initial motif model. This model is improved heuristically by a local improvement approach. The aim is to minimize the sum-of-pairs score of the model: SPscore (m_1, m_2… m_n) = $\Sigma_{i,j}$ $\delta(m_i, m_j)$. For any length-*l*

motif in the model: a) find the best match motif in each of the sequences; b) extract the consensus motif by choosing the most frequently appearing nucleotide in each position among these *n* length-*l* motifs. SP-STAR repeats this local improvement procedure until SPscore cannot be further improved. The time complexity of finding the best potential sample consensus, which has the minimal sum-of-pairs score, is *O* $((nm)^2)$. The time of local improvement is *O* (*Mnml*). *M* is the number of iteration needed to converge. This number is unpredictable. Also, this algorithm may converge to a local optimum.

2.2.3 COPIA

COPIA (Consensus Pattern Identification and Analysis) (Liang 2001) is a software for finding consensus pattern in the sequences. The algorithm assesses every *r* (*r* is a constant less than or equal to *n*) subsequences of length *l* (each in a different sequence) and extract the consensus motif choosing the most frequently appearing nucleotide in each position among the *r* subsequences. The consensus motif is then used to find its closest pattern in each of the *n* sequences and these form a motif model. The output is the new consensus motif that produces the total minimal distance score to its instances. The time complexity of this algorithm is O $((nm)^{r+1}l)$ when $r \geq 3$.

2.2.4 Random Projection Approach

Buhler and Tompa designed the PROJECTION algorithm (Buhler *et al*. 2001) that can find good starting points for consensus motif models. It is designed for the (*l*, *d*)-motif model, and it selects a projection by selecting *k* out *l* positions at random. Then each length-*l* string can be *hashed* into buckets based on these *k* positions. Within the bucket all the instances have the same nucleotides in these *k* positions. For *k* too large the number of motif instances that form a family under the projection are small. The random projection algorithm can be run multiple times and the best motif from these runs will be

selected. The time complexity of this approach is hard to predict, but usually linear time complexity in the size of data set can be achieved.

2.2.5 Tree-based Approaches

Tree-based approaches (Eskin *et al*. 2002, Sagot 1998) search for the (*l, d*) motif model. They use a suffix tree data structure or some of its variations. The idea is that the consensus motif will be one of the *l*-mers in the data set or their neighborhood. These methods can find conserved patterns by traversing the suffix tree. They can find all valid motif models through one search. However, due to very exhaustive searching of the *e*-neighborhood, they are quite time consuming and grossly impractical.

2.2.6 Weeder

Weeder (Pavesi *et al*. 2004) is an algorithm that uses concept of consensus and enumerates exhaustively all the k-mers up to a maximum pre-specified length. It collects motif occurrences from the input sequences and evaluates each motif according to number of sequences in which it appears and how well conserved it is in each sequence. This conservation is estimated with respect to expected values derived from the k-mer frequency analysis. Different combinations of 'canonical' motif parameters are automatically tried by the algorithm in different runs. These parameters are derived from the analysis of known instances of yeast TFBSs. Weeder analyzes and compares the top-scoring motifs in each run using a simple clustering method. The aim here is to detect which ones are likely to correspond to TFBSs. Best instances of each motif are selected using a weight matrix built with sites found by consensus-based algorithm.

The website of the algorithm is:

http://159.149.109.16/Tool/ind.php

2.3 Discussion

Several comparative studies (Brazma *et al*. 1998, Brejova *et al*. 2000, Rigoutsos *et al*. 2000, Sinha *et al*. 2000, Sinha *et al*. 2003, Tompa *et al*. 2005) about these approaches have been reported. On simulated samples (*e.g.* on the samples containing simulated (*l, d*) motifs), consensus approaches are shown to perform better than statistical profile based approaches, because consensus approaches take more information about the *d* mismatch than statistical approaches. However there are still no experiments to show that the consensus approaches are significantly better on real biological data. One of the observed common drawbacks of the statistical Profile Model approaches is their frequent trapping at a local optimum. Moreover, in practice, enumerative consensus approaches take much longer time. That is why statistical approaches are still the most popular choice for biologists.

Chapter 3 Simulated Annealing in Motif Discovery

3.1 Overview of Simulated Annealing

The idea for Simulated Annealing based optimization (Kirkpatrick *et al* 1983) relies on the principles of thermodynamics and mimics the process in which a solid material is first melted and then allowed to cool by slowly reducing temperature. This approach is very suitable for discrete combinatorial optimization problems, such as our problem.

Simulated Annealing makes use of the definition of neighborhood. The algorithm simulates a walk through the solution space that is obtained by iteratively moving from the current solution to its neighbor, which is randomly selected from the current solution's neighborhood. The detail definition of a neighbor will be discussed in the next section. Improved moves are always accepted, while deteriorating moves are only accepted with a certain probability. This acceptance probability is controlled by a parameter which is called *temperature* T and a function $F(S_0, S_1, T)$ which calculates the probability value. The algorithm stops when the solution converges or the *temperature* is small enough.

In Simulated Annealing, for each solution S, there is a *cost* associated with the defined *quality* based on the criterion. Usually, the smaller this cost, the better. The function $F(S_0, S_1, T)$ is constant, which is chosen such that solutions corresponding to a large increase in cost will have a small probability of being accepted, and solutions corresponding to small increases in cost will have a larger probability of being accepted. There is no limitation on the size of the deterioration with respect to its acceptance. A sample function given by standard Simulated Annealing is shown in equation (3.1).

$$F(S_0, S_1, T): P_T(\text{accept } S_1 \text{ from } S_0) = \begin{cases} 1 & \text{If } C(S_1) \le C(S_0) \\ \exp\left(\frac{C(S_1) - C(S_0)}{T}\right) & \text{If } C(S_1) > C(S_0) \end{cases} \quad (3.1)$$

The *temperature* is the most important parameter in Simulated Annealing. It is a non-increasing sequence of numbers which tend to zero during the search process. In the beginning of the search, when *temperature* is large enough, large deteriorations are accepted and the algorithm tends to accept any move; as *temperature* decreases, only small deteriorations are accepted; finally, close to the end of computation, as *temperature* approaches zero, the probability of accepting a worse solution is very small, almost no deterioration is accepted, and the search simply attempts to find the local minimum similar to greedy iterative algorithms. The speed of *temperature* reduction plays an important role in the convergence of the algorithm and hence has a great influence to the result quality. However, to determine a function for temperature reduction is still a major unresolved problem in Simulated Annealing algorithm. Slowing down the "cooling schedule" will increase the probability of finding global optimum solution, so we have to make a trade off. A simple temperature reduction function is given in the equivalent (3.2).

$$T \leftarrow kT \ (0< k < 1, k \text{ is always chosen to be greater than 0.95 in practice}) \quad (3.2)$$

From the above description, we can see that Simulated Annealing has the ability to escape from local minima. This is achieved by jumping out of them before the solution is too close to local minima. That means finding the global minimum is not related to the initial condition which is the initial solution in our case. Another advantage is its very simple implementation. However, the selection of Simulated Annealing parameters is a bit subjective and this represents the main disadvantage. Simulated Annealing is

sometimes called a “biased random walk”. This due to the fact that iteration steps are made randomly and they do not contain an intelligent move as most of the other optimization techniques. One of the characteristics of Simulated Annealing algorithm is that it does not require the knowledge of the search space. This can be an either advantage or a disadvantage depending on conditions of application and problem in question.

Let us define some annotations below, which are used in the pseudocode.

r = a random number between 0 and 1;

T_0 = initial “temperature” in SA;

T = variable analog to “temperature” in SA;

T_{stop} = temperature threshold; the algorithm will stop when T is smaller than this parameter.

The pseudocode of standard Simulated Annealing algorithm is given in Figure 3.1

ALGORITHM SIMULATED_ANNEALING

1. Generate an initial schedule S_0 and temperature T_0
2. Set the initial best schedule $S = S_0$
3. Compute cost of S_0: $C(S_0) = C(S)$
4. Set the temperature $T = T_0$
5. While T is not small enough do:
5.1 Select a random neighbor S_1 to the current schedule, $(S_1 \in N(S_0))$
5.2 Set $\Delta(C) = C(S_1) - C(S_0)$.
5.3 If $(\Delta(C) \leq 0)$ {downhill move}
5.3.1 Set $S_0 = S_1$.
5.3.2 If $C(S_1) < C(S)$ then set $S = S_1$
5.4 If $(\Delta(C) > 0)$ {uphill move}
5.4.1 Choose a random number r uniformly from [0, 1].
5.4.2 If $r < e^{-\Delta(C)/T}$ then set $S_0 = S_1$
5.5 Reduce (or update) temperature T
6. Return the schedule S

Figure 3.1 Pseudocode of Standard Simulated Annealing

3.2 Neighborhood Generating Mechanisms

Standard heuristic Algorithms, such as Simulated Annealing, look for the global optimum by iteratively moving from the current solution to the next. With this feature, they require the concept of neighbors of the current solution in order to ensure an improvement of the current solution. It is necessary to search neighboring solutions before the improved one can be found. In our problem of motif discovery, three types of neighborhood generating mechanisms are used, namely *basic* Neighborhood Generating Mechanism, *In-file* Neighborhood Generating Mechanism and *nearest* Neighborhood Generating Mechanism. They are used in this way: either basic Neighborhood Generating Mechanism or In-file Neighborhood Generating Mechanism is first used in the algorithm to look for a neighbor; if none of the neighbors is acceptable because of the algorithm restrictions, *nearest* Neighborhood Generating Mechanism will be used to definitely provide a valid neighbor for the algorithm to continue searching.

3.2.1 Basic Neighborhood Generating Mechanism

Let a motif S_0 represent a current solution. A *basic* neighbor of S_0 is defined as:

> **$S_1 \in N(S_0)$, where S_1 can be obtained from S_0 by changing a nucleotide in any position to any other allowed nucleotide (A, C, G, T).**

The operation is illustrated in Figure 3.2.

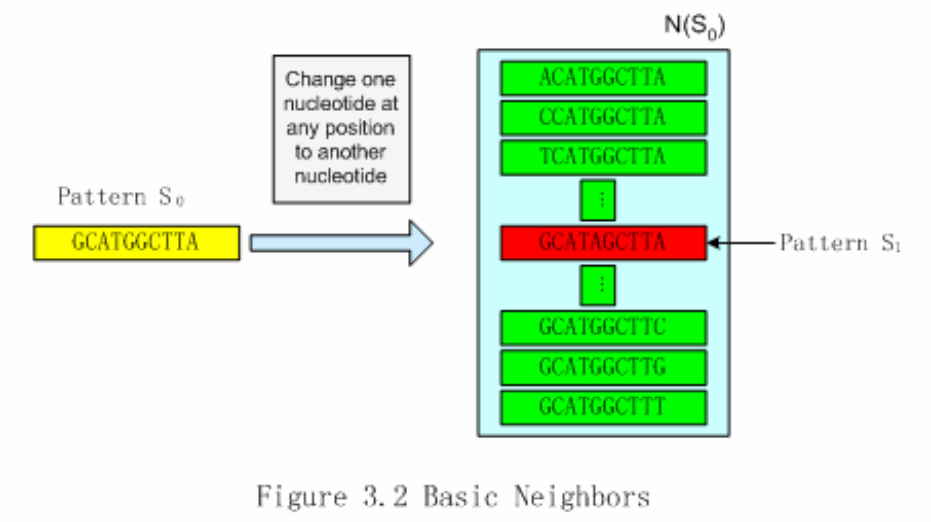

Figure 3.2 Basic Neighbors

Based on the algorithm, we have 3*l* neighbors for the current solution. The search space of this neighborhood generating algorithm includes all the l^4 possibilities. With this neighborhood definition, the search space is continuous, which is very helpful for the algorithm to search the whole solution space thoroughly. However some solutions (motifs) found by using this neighborhood definition may not appear in the input sequences, because the sequences most likely do not contain each of the l^4 motifs. In the case of $l^4 > n(m-l+1)$, there are at least $l^4-n(m-l+1)$ number of motifs found are not in the input sequences.

3.2.2 In-file Neighborhood Generating Mechanism

The constraint that the consensus motif must appear in the input sequences suggests a different strategy. As the name suggests, all the neighbors found by using this mechanism must come from the input sequences.

Let a pattern S_0 represent a current solution. An In-file neighbor of S_0 is defined as:

$S_1 \in N(S_0)$, where S_1 can be obtained from the input sequences such that the similarity between S_0 and S_1 is greater than the threshold.

The operation to get an *In-file neighbor* is illustrated in Figure 3.3.

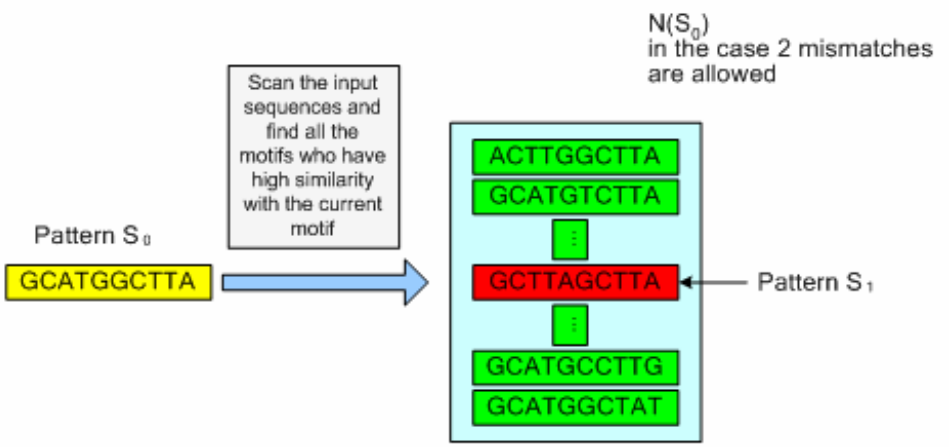

Figure 3.3 In-file Neighbors

Based on this algorithm, the search space contains only those motifs appearing in the input sequences. The maximum number of different motifs is min ((m-l+1)n, 4^l). In practice, the number of different motifs is much smaller than this number. However, to extract these in-file neighbors, we have to do the following: a) scan the whole input file, which takes O (nm) time; b) do comparison between the current solution (motif) and the substring pattern from the input sequences, which takes O (l) time.

The advantage of using this neighborhood definition is the ability to reduce the search space, which is very important in solving those *NP*-hard problems (like our problem). However, there is a serious draw back of this neighborhood definition, i.e. it results in a non-continuous search space. There may be a huge gap between some sections of the solution space (those solution motifs locating in different sections differ in too many positions). If the starting solution (motif) is in one of such sections, the search will never reach the other sections of the solution space. Without the ability of jumping among these sections, the algorithm can only search for the local optimal. To overcome this draw back, *in-file* neighborhood generating mechanism has to be used together with *nearest* Neighborhood Generating Mechanism. This will be explained in the next section.

3.2.3 Nearest Neighborhood Generating Mechanism

There might be some situations, in which either *basic* Neighborhood Generating Mechanism or *In-file* Neighborhood generating mechanism cannot find acceptable neighbors because of the following reasons: a) the consensus motif must appear in the input sequences; b) the forbiddance of Tabu solutions, which is explained in detail in Tabu Search chapter. However, we have to get a neighbor to continue searching. A motif of the same length, which has the least mismatches compared with the current solution will be chosen from the input sequences as the initial solution in the next search iteration. This neighbor can be found at the same time of searching for *in-file* neighbors.

3.3 Implementation of Simulated Annealing Algorithm

The Simulated Annealing algorithm we use to solve our problem belongs to the consensus motifs approaches. The idea of this approach is: it tries to look for a consensus motif first; then the consensus motif is used to scan the sequences to look for those similar motifs to build the motif model.

Starting from the initial motif, Simulated Annealing iteratively searches the neighborhood of the current solution in order to get closer to the global optima. In our problem, to look for the neighbor of the current solution, both *basic* neighborhood generating mechanism and *In-file* neighborhood generating mechanism are implemented. If no valid *basic* neighbor or *In-file* neighbor can be found, *Nearest* neighbor will be used. For each solution, we have a cost associated with it based on which of the three criteria is selected as described in the former content. The algorithm stops when the *temperature* is smaller than some defined threshold.

3.3.1 Outline of Simulated Annealing Implementation

In the implementation, there are three parts of the solution: CONTROLLER, RESTART and SA. Every time, CONTROLLER calls RESTART to get an initial solution and passes it to SA. SA runs the initial solution until the *temperature* becomes very low and the solution reaches some local minimum. Then CONTROLLER calls RESTART again. Figure 3.4 shows the flowchart of the implementation.

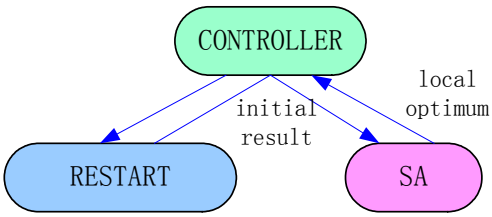

Figure 3.4 Flowchart of SA Implementation

3.3.2 Detail Algorithm of SA Using Basic Neighbors

We can use the relation $r < e^{-\Delta(C)/T}$ to decide whether or not to accept the move from the current solution to its neighbor. In the beginning, when T is large enough, these moves are similar to randomly selecting a position and changing the nucleotide on this position to another. When T becomes very small, only a downhill move (a move which will reduce the value of the criterion function) can be accepted. We have to find such a downhill move. So, here, we have two problems to solve:

(a) It is a necessity to have some move order, so that when T is very small we can scan all the possibilities to find a downhill move. Finding a downhill move randomly is not reasonable when T is very small.

(b) As T is large in the beginning, it is likely to accept most moves no matter whether they are downhill or uphill. Since we are using SA, we have to make the move random enough. We cannot always change the nucleotide at some particular position.

To solve the above two problems, we follow the following 3 steps every time we are looking for a neighbor solution (assume the motif we are looking for has 10 nucleotides):

Step1: Randomly reorder all the positions: R_1 (4, 10, 9, 1, 5, 6, 3, 2, 7, 8).

Step2: Randomly reorder all the nucleotides: R_2 (G, T, A, C)

Step3: For each position in R_1, change the nucleotide on that position to another nucleotide in the order R_2.

These three steps allow us to have an order to scan all possible moves.

Every time, we update the current solution, we compare it with the best solution we found so far. If the current solution is better, update will be done. To ensure that the final solution motif we get appears in the input sequences, we have to check the existence every time we update the best solution.

The flowchart of data processing within SA block is shown in Figure 3.5:

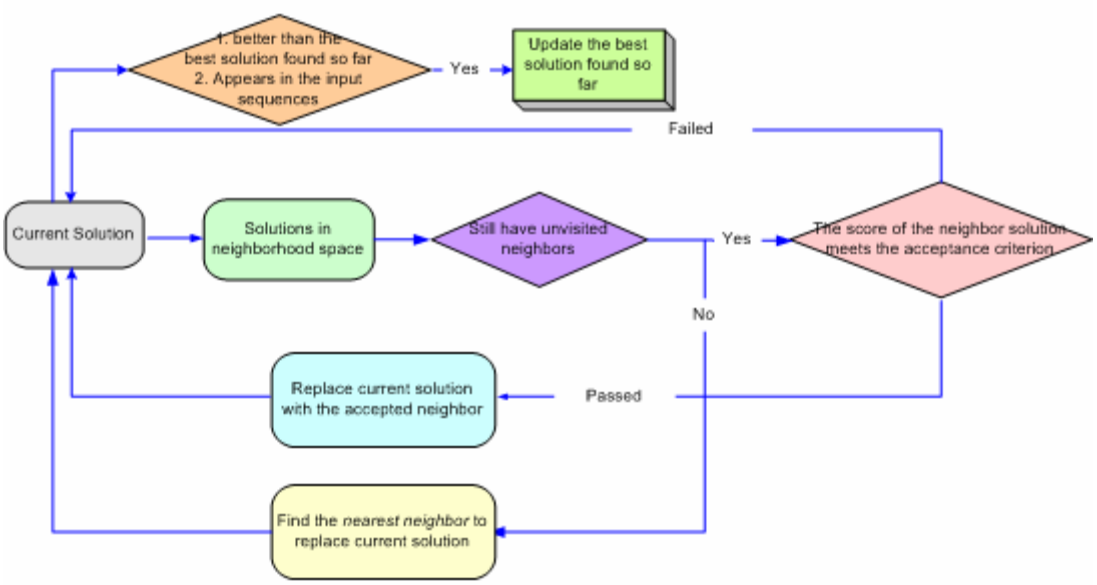

Figure 3.5: Flowchart of SA Using *Basic* Neighbors

The pseudocode of the algorithm is described in Figure 3.6:

Steps 1 to 4 are doing the initialization.

Step 5 reorders both position list and nucleotides list to ensure the research thorough and random.

Step 6 is the stop condition of the algorithm.

Step 6.1 is the stop condition for each iteration. It checks whether the Basic Neighborhood Generating mechanism can find an acceptable solution.

Steps 6.1.3 to 6.1.3.2 accept any downhill move.

Steps 6.1.4 to 6.1.4.2 accept the uphill move with certain probability.

Steps 6.2 to 6.2.2 find the *nearest* neighbor in the situation when no acceptable basic neighbor can be found.

Step 6.3 updates the temperature every n iteration move from the current solution to its neighbor.

ALGORITHM SIMULATED_ANNEALING

---

1. Generate an initial solution pattern S_0 and temperature T_0
2. Set the initial best solution $S = S_0$
3. Compute the cost of S_0: $C(S_0) = C(S)$
4. Set the temperature $T = T_0$
5. Reorder the positions and nucleotides to get R_1 and R_2
6. While T is not small enough do:
6.1 While move has not been accepted AND R_1 has not reached the end
6.1.1 Select a neighbor S_1 to the current solution according to R_1 and R_2, ($S_1 \in N(S_0)$)
6.1.2 Set $\Delta(C) = C(S_1) - C(S_0)$.
6.1.3 If ($\Delta(C) \leq 0$) downhill move
6.1.3.1 If ($C(S_1) < C(S)$ AND S_1 appears in the input sequences, then set $S = S_1$
6.1.3.2 Set $S_0 = S_1$ and return to Step 6
6.1.4 ELSE uphill move
6.1.4.1 Choose a random number r uniformly from [0, 1].
6.1.4.2 If $r < e^{-\Delta(C)/T}$ then set $S_0 = S_1$ and return to Step 6
6.2 If no neighbor is accepted
6.2.1 Let S_{near} be the neighbor generated by using *Nearest Neighborhood* Generating Mechanism
6.2.2 Set $S_0 = S_{near}$ and return to Step 6
6.3 Every n iterations, reduce (or update) temperature T: $T \leftarrow kT$
7. Return the solution motif S

Figure 3.6 Pseudocode of the SA Using *Basic* Neighborhood Generating Mechanism

3.3.3 Detail Algorithm of SA Using In-file Neighbors

When Simulated Annealing uses *in-file neighborhood* generating mechanism, those in-file neighbors have to be extracted from the input sequences in the beginning of each iteration. To ensure that the selection of neighbor is random enough and thorough as discussed in the previous section, we simply reorder the collection of those in-file neighbors and select the neighbor according to this order.

Since the solution motifs we visited under this definition of neighbor are all from the input sequences, no check is needed before updating the best solution.

3.4 Time complexity of Simulated Annealing Algorithm

3.4.1 Time Complexity of SA Using Basic Neighbors

1. For two random reorderings, the time complexity is O(*l*) and O(1).

2. For neighbourhood move operation, each position has three possible changes, so there are O(l) neighbors for every solution.
3. To evaluate each neighbor solution, we have to compare it with all the patterns in the input file. For one sequence, there are m-l+1 number of patterns, so there is in total (m-l+1)*n patterns in the input file. For each comparison, l nucleotides are compared. So the time complexity is O ((m-l+1)*n*l) = O (m*n*l) (since in most of the time l is much smaller than m).
4. For the worst case, all those standard neighbors cannot meet the requirement, *nearest* neighbor have to be found. This takes O (m*n*l) time and only one neighbor is returned.
5. From 2, 3 and 4, we can get the time complexity for each iteration is O(l) * O(mnl) + O(mnl) + O(mnl) = O (mnl^2).
6. The algorithm takes i iterations to converge. Here, i depends on the "cooling schedule", which is the equation (4.2), and the stop threshold. It can be calculated by using the in-equation $T_0 ki < T_{stop}$. Hence the total time complexity of this algorithm is O ($mnl^2(\log(T_{stop}/T_0)/\log(k))$).

3.4.2 Time Complexity of SA Using In-file Neighbors

1. For the random reordering, the time complexity is O(l).
2. For nighborhood move operation, the number of neighbors U is an unpredictable value.
3. The evaluation of the solution is the same as in SA using *basic* neighborhood generation. The time complexity is O ((m-l+1)*n*l) = O (m*n*l).
4. To find the *nearest* neighbor, it takes O (m*n*l) time and only one neighbor is returned.

5. From 2, 3 and 4, we can get the time complexity for each iteration is O(U) * O(mnl) + O(mnl) + O(mnl) = O ($Umnl$).

6. Hence the total time complexity of this algorithm is O ($Umnl$(log(T_{stop}/T)/log(k))).

3.5 Conclusion

Simulated Annealing algorithm using in-file neighbors converge much faster, because of the smaller solution space. However, the quality of the solution is not that good as what we get from the same algorithm but using basic neighborhood definition. It is because the solution space under in-file neighborhood definition is non-continuous. In Simulated Annealing, no memory is used for recording the visited solutions which will be forbidden in the later iterations, and the only acceptance criterion is the function (3.1). Original idea of Simulated Annealing tries to jump out of the local optimum at the time when the *temperature* is high enough, but now the algorithm accepts almost every in-file neighbor similar to each other when the temperature is high, which results in being trapped in the same islanded section of the solution space. Although with the help of *nearest neighbors*, which can only be happen in the situation that no in-file neighbor is acceptable when the *temperature* is very small, the search can jump out of the section and break away from the local optimum, it already loses the best feature of Simulated Annealing algorithm.

Chapter 4 Tabu Search in Motif Discovery

4.1 Overview of Tabu Search

Tabu Search, which is a meta-heuristic approach, is a neighborhood search method introduced by Glover (1986). Many computational experiments have shown that Tabu Search can be applied in many combinatorial problems. Because of its flexibility, Tabu Search has already beat many classical procedures.

Tabu Search is also an iterative procedure just like Simulated Annealing. It explores the solution space by moving from the current solution to the best solution in N*(S), which is a subset of its neighborhood N(S) at each iteration. The initial solution is typically created with some cheapest insertion heuristic or sometime even randomly. When the initial solution is created the algorithm tries to improve it by using local search with one or more neighborhood structures and a best-accept strategy. Unfortunately, unlike the classical descent methods, the current solution may deteriorate from one iteration to the next and the risk of visiting again a solution and more generally of cycling is present. Thus, to avoid cycling, those *tabu solutions* are not allowed to be chosen. Here, *tabu solutions* are those solutions containing *tabu-active* elements, which are the selected attributes of recently explored solutions. The duration that an attribute remains tabu is called *tabu-tenure*, and can vary over different intervals of time.

Without memories, Tabu Search just selects the best neighbor solution for the next iteration. In order to avoid being trapped into local optimal and to improve the efficiency of the exploration process, the algorithm proposes the technique of tabu, which keeps track some information along the exploration process (like those solutions visited in the previous iterations). With the help of this memory technique, certain solutions are

prevented from N*(S) and hence from being revisited. This systematic use of memory is an essential feature of Tabu Search. One role of the memory is to restrict the choice to some subset of the neighbor solutions. It means even for the same solution, they may have different neighbors if they are encountered in the different stage of the search process. That is why Tabu Search is called a dynamic neighborhood search algorithms.

4.1.1 Two Types of Memory

There are two types of memory in Tabu Search. One is short-term memory, and the other is long-term memory. The effect of both types of memory can be viewed from modifying the neighborhood N(S) of the current solution.

The short-term memory component is the starting point for many Tabu Search implementations. Under the control of short-term memory, N*(S) is a subset of N(S) by using tabu condition to prevent a particular solution, or set of solutions, from being chosen as the outcome of the next move. The most important short-term memory to determine the solutions in N(S) is the use of tabu list. In the simplest form, a tabu list contains the tabu solutions that have been visited in the recent past (less than tabu-tenure number of iterations). N*(S) is got from N(S) by excluding those solutions in tabu list. Other tabu list structures can contain those *tabu-active* attributes or prevent certain moves. Those attributes whose tenure expires are removed from the tabu list at the time some new attributes are added. Tabu lists containing attributes are much more effective, although they raise a new problem. With forbidding an attribute as *tabu-active* element, typically more than one solution is declared as tabu solutions. Some of these solutions that must now be avoided might be of excellent quality and have not yet been visited. To overcome this problem, the tabu status can be overridden if certain conditions are met;

this is called *the aspiration criterion* and it happens, for example, when a tabu solution is better than any previously found solution.

In the simplest case, long-term memory is used to restart the search. Once the search by short term-memory gets stuck, long-term memory determines the new starting point. One example of the use of long-term memory is the elite solutions list, which records those elite solutions found so far. Members in the elite solutions list are determined by setting a threshold that is connected to the objective function value of the best solution found during the search. With the analysis on those elite solutions, we can tell the common attributes or the difference among them. These common attributes and the difference will be used in Intensification restart and Diversification restart, which is discussed in Section 4.1.3.

4.1.2 Use of Memories

Generally, those memory structures in Tabu Search operate in four dimensions: recency, frequency, quality, and influence.

Recency-based memory is the most commonly used short-term memory. It keeps track of solution attributes that have changed during the recent past and assigns them as tabu-active elements. This prevents certain solutions of recent past from belonging to the admissible neighborhood of the current solution and hence from being revisited.

Generally, frequency-based memory is a type of long-term memory, which stores frequency of searching in each area and it provides complements the information provided by recency-based memory. It may store the number of times an attribute enters or leaves the solutions or the number of iterations during which an attribute belongs the

solutions generated. Like other type of long-term memories, frequency-based memory is applied only to those elite solutions.

Quality-based memory is also called elitist memory, which is a long-term memory. It records the *n* best candidates found so far. An example of quality-based memory is the elite solutions list which has been described in the former context. The neighborhood of those solutions in the elite memory can be used to extend the search neighborhood.

Influence-based memory considers the effects of the different choices on both quality and structure during the search. It records the information about the influence of choices on particular solution elements.

Using these four types of memories we can realize two key strategies which are highly important components of Tabu Search: intensification and diversification.

4.1.3 Intensification and Diversification Strategies

Intensification strategy aims to identify solutions attributes that are common to good solutions and to encourage the Tabu Search to seek solutions with these common attributes. This strategy enables a more thorough search of the solution space. Diversification strategy is a complement of this. It aims to encourage the search process to search unexplored regions and to generate solutions that differ in various significant manners from those already used. This can be done by forbidding those attributes that are common to those good solutions. This strategy can radically shift searching area to different section of the solution space.

4.1.4 Flowchart and Pseudocode of the Standard Tabu Search

The prime Tabu Search flow is shown in Figure 4.1

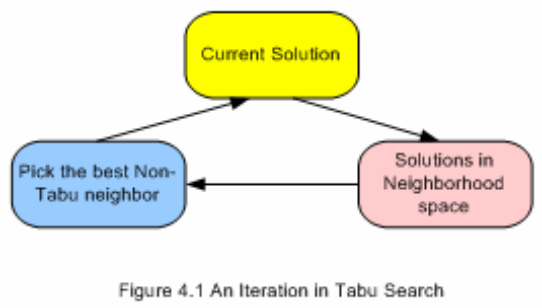

Figure 4.1 An Iteration in Tabu Search

From this figure, we see that there are two important elements in the iteration: one is defining a neighborhood and the other is checking whether a move is a tabu move or not. We use the *basic* Neighborhood Generating Mechanism and *In-file* Neighborhood Generating Mechanism as what we did in SA. To check whether a move is a tabu move, we have to have one or more tabu lists, which is a recency-base memory. With the help of these *tabu-active* elements, we should be able to jump out the cycles.

In Tabu Search, the stop condition can be defined in any way that is suitable for the problem. Here are some possible immediate stopping conditions:

a) $N^*(S) = \Phi$;

b) Maximum number of iterations are reached;

c) The number of iterations since the last improvement of the solution is larger than some defined number (a threshold);

d) Some evidence can be given that an global optimum solution has been obtained.

The following are the pseudocode of standard Tabu Search:

```
ALGORITHM TABU_SEARCH
1.      Generate an initial solution S0
        Set the current solution S = S0;
        Set the final solution Sf = S0;
        Set globalMin = MAX
2.      WHILE stopCondition has NOT be hit, DO
2.1         Set neighborMin = MAX; Set tempNeighbor = S
2.2         For each neighbor S* ∈ N(S)
2.2.1           IF C(S*) < neighborMin AND S* is not a tabu solution
2.2.1.1             Set neighborMin = C(S*);
                    Set tempNeighbor = S*
2.3         IF neighborMin < globalMin
2.3.1           Set globalMin = neighborMin;
                Set Sf = tempNeighbor
2.4         Set S = tempNeighbor AND update Tabu List
3.      return Sf
```

Figure 4.2 Pseudocode of Standard Tabu Search

4.2 Neighborhood Generating Mechanisms

Tabu Search is also a standard heuristic algorithm. Like Simulated Annealing, it explores the solution space by looking for the next solution from the current one. The same three neighborhood definitions are used in Tabu Search as what we have described in the chapter of Simulated Annealing.

4.3 Implementation of Tabu Search Algorithm

This Tabu Search algorithm we use to solve our problem also belongs to the approaches for consensus motifs. Like other consensus motifs approach, Tabu Search tries to look for a consensus motif first, and then uses this consensus motif to scan the sequences to look for those similar motifs to build the motif model.

Tabu Search is also a neighborhood local search algorithm like Simulated Annealing. The search steps are very similar with that of Simulated Annealing, but the algorithm of

choosing the neighbor strictly follows the strategy of Tabu Search, which is totally different from that of Simulated Annealing. Starting from the initial solution (motif), Tabu Search iteratively searches the neighborhood of the current solution in order to get closer to the global optimum. In the problem, to look for the neighbor of the current solution, both *basic* neighborhood generating mechanism and *In-file neighborhood* generating mechanism are tried. If by the use of these two neighborhood definitions no acceptable neighbor is found, *nearest Neighbor* will be used. For each solution, we have a cost associated with it based on which criterion is selected as described in the section of problem definition. The algorithm stops if the number of iterations since the last improvement of the solution is larger than some defined number.

4.3.1 Outline of Tabu Search Implementation

There are five parts in Tabu Search algorithm: CONTROLLER, INTENSIFICATION, DIVERSIFICATION, RESTART and TS. Every time, CONTROLLER passes the *elite solutions list* to INTENSIFICATION and DIVERSIFICATION, which then analyze those elite solutions to get some constrains for restart. Restart receives and follows those constrains to generate one intense solution and one diverse solution. These two solutions are returned to CONTROLLER and will be used as initial solutions in TS, which is the complete Tabu Search algorithm. TS searches the solution space starting from these two initial solutions until some stop criterion is met, and generates a list of elite solutions (the function value is within some tolerance comparing with the best solution found so far). Lastly, TS returns this *elite solutions list* to Controller. Figure 4.3 shows the flowchart of Tabu Search algorithm.

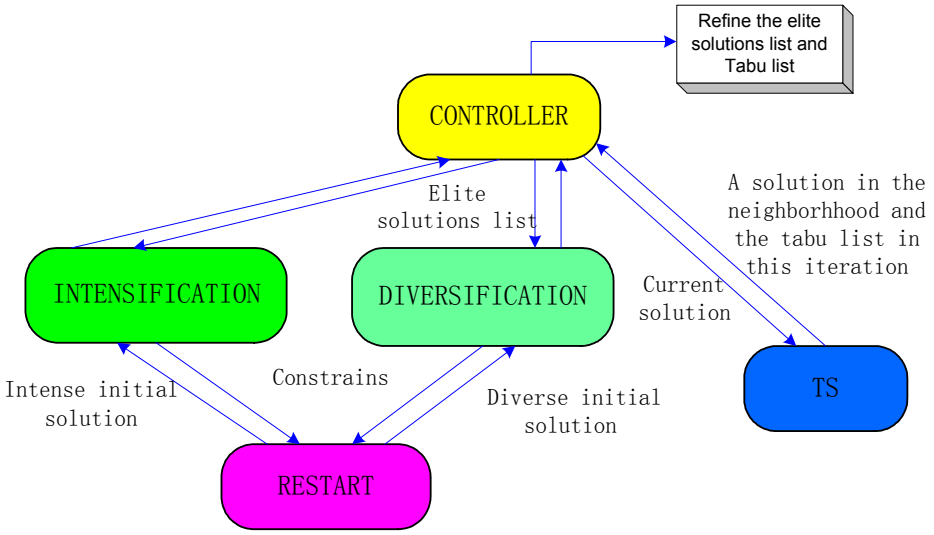

Figure 4.3 Flowchart of TS Implementation

4.3.2 Detail Algorithm of Tabu Search Using Basic Neighbors

As described in the previous section, there are several parts in the implementation of this algorithm, and they are explained separately.

CONTROL

Controller guides TS to find a good solution more efficiently. It passes information between the Intensification/Diversification part and the TS part.

INTENSIFICATION

This part finds the common elements of those elite solutions. Here, the common elements are those which represent the same nucleotides at the same positions in different elite solutions (motifs). To get an intensification result, we still make use of these common elements.

In the implementation, for each position, if one nucleotide is used more than 75%, the intensification result will keep this nucleotide in this position. An example is shown in Figure 4.4

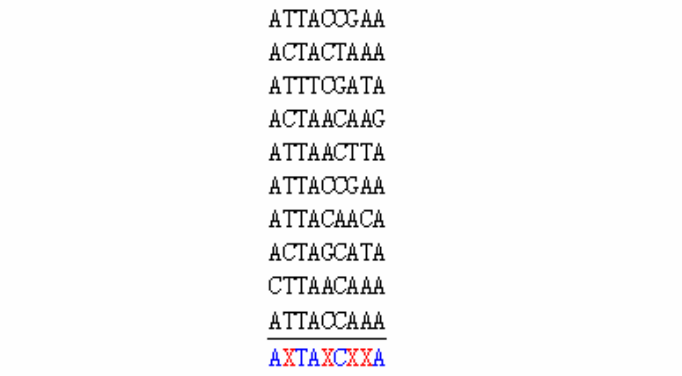

Figure 4.4 Partial Solution Generated by Intensification Strategy

This partial solution is sent to Restart, where those “X” nucleotides can be determined.

DIVERSIFICATION

This part finds the common elements of the elite solutions, which will not be allowed to use in the diversification result.

In the implementation, for each position, if one nucleotide is used more than 75%, the diversification result will forbid this nucleotide in this position. We randomly choose any other nucleotide for this position. An example is shown in Figure 4.5

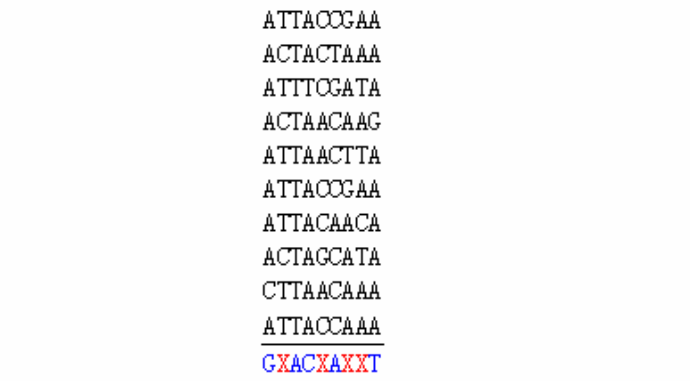

Figure 4.5 Partial Solution Generated by Diversification Strategy

RESTART

This component receives the partial solution from the Intensification and Diversification components to complete it. Here we use random assignment for non-common positions.

TS

The most important part in Tabu Search is the definition of the neighborhood. Like in Simulated Annealing, both *basic* neighborhood generating mechanism and *In-file* neighborhood generating mechanism are implemented. In this section, the implementation of Tabu Search using *in-file* neighbors is talked about.

As the feature of Tabu Search, which is different from SA, we need to search all possible neighbors of the current solution and to choose the best non-tabu solution. That means we need a tabu list to record those *tabu-active* elements. Here, the idea of looking for a non-tabu *in-file* neighbor and updating the tabu list is illustrated in Figure 4.6 (Tabu with format (3, G, 2) means that the nucleotide in the 3rd position cannot be changed to G in the solution motif within the next 2 iterations).

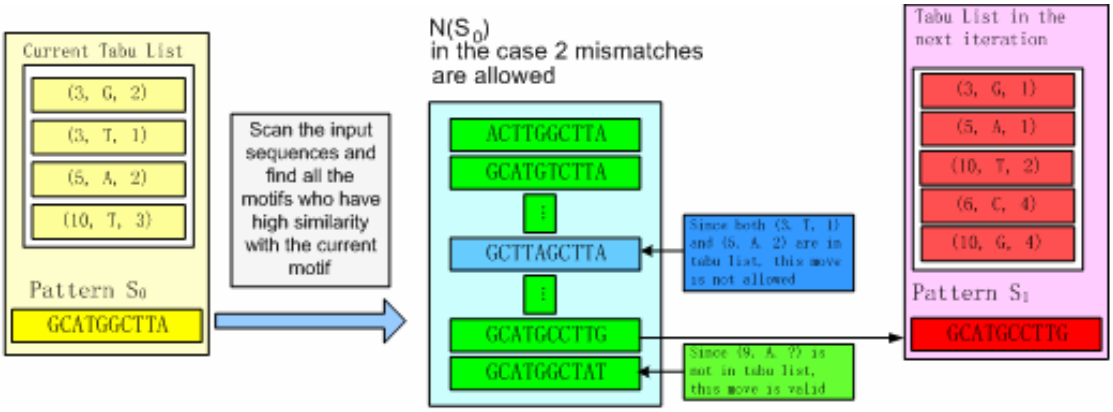

Figure 4.6: Choose a Neighbor for the Next Iteration in TS

In TS, the improved-best aspiration criterion is also used. As described, Intensification and Diversification restart strategies are very important components here, we need memories to record all the elite solutions that are within some defined tolerance to the best solution found so far.

The flowchart of TS using the *in-file* neighborhood generating mechanism is given in Figure 4.7.

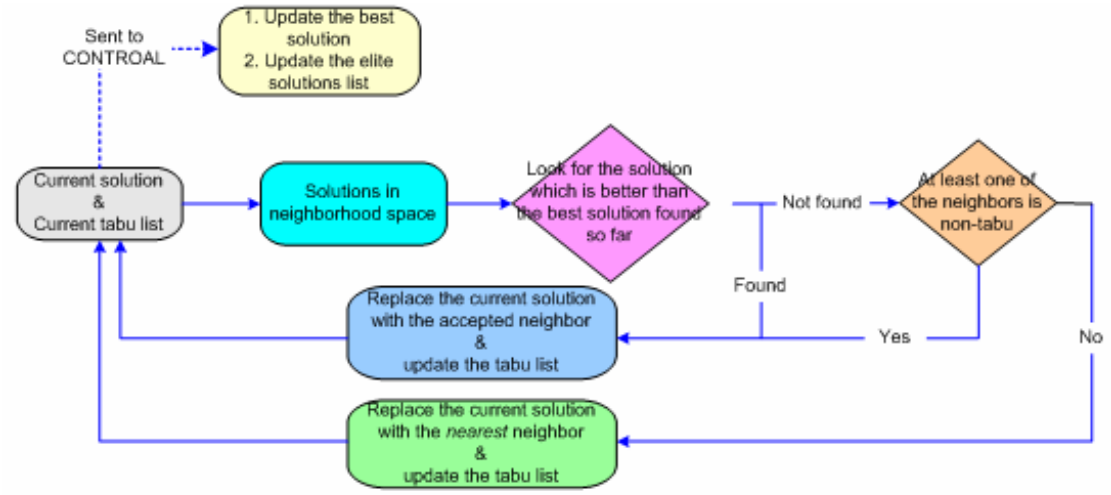

Figure 4.7 Flowchart of TS

The corresponding pseudocode is illustrated in Figure 4.8.

Step 1 does some initializations for global variables

Step 2.1 resets some local variables for each iteration

Steps 2.2 to 2.2.2.1 look for the *best solution* and the *best non-tabu solution* in the current solution's neighborhood.

Steps 2.3 to 2.4 do some update. If the *best solution* is better than any other solution found so far, this solution is selected as the initial solution for the next iteration; otherwise, the best non-tabu solution is selected.

```
ALGORITHM TABU_SEARCH
1.      Generate an initial solution S0
        Set the current solution S = S0;
        Set the final solution Sf = S0;
        Set globalMin = MAX
2.      WHILE stopCondition has NOT be hit, DO
2.1         Set neighborMin = MAX;
            Set nonTabuMin = MAX;
            Set tempNeighbor = S;
            Set nonTabuTempNeighbor = S
2.2         For each neighbor S* ∈ N(S)
2.2.1           IF C(S*) < neighborMin
2.2.1.1             Set neighborMin = C(S*);
                    Set tempNeighbor = S*
2.2.2           IF C(S*) < nonTabuMin AND S* is not a tabu solution
2.2.2.1             Set nonTabuMin = C(S*);
                    Set nonTabuTempNeighbor = S*
2.3         IF neighborMin < globalMin
2.3.1           Set globalMin = neighborMin
                Set Sf = tempNeighbor
                Set S = tempNeighbor
2.3.2       ELSE IF nonTabuTempNeighbor exists
                Set S = nonTabuTempNeighbor
2.3.3       ELSE
                Set S = nearest neighbor
2.4         Update Tabu List
3.      return Sf
```

Figure 4.8 Pseudocode of Our Tabu Search Implementation

4.3.3 Detail Algorithm of Tabu Search Using In-file Neighbors

The algorithm is quite similar as that of using basic neighbors except the following two differences:

a) As the difference of the neighborhood definition, neighbors are generated differently. Current solution S_0 is used to scan the input sequences. All those patterns from the input sequences, which have the same length as the current solution and differ from the current solution within some defined threshold, will be selected as the current solution's neighbors.

b) As the solution patterns visited are all from input sequences, there is no need to do any check before updating the best solution and the elite solutions list.

4.4 Time Complexity of Tabu Search Algorithm

4.4.1 Time Complexity of TS Using Basic Neighbors

Let T = *tabu-tenure* (the number of iterations that an attribute remains tabu)

This T must be less than 4*l*, otherwise after 4*l* iterations all possible moves will be *tabu moves*.

1. For neighbourhood move operation, each position has three possible changes, so there are O(*l*) neighbors for every solution.
2. To evaluate each neighbor solution, we have to compare it with all the patterns in the input file. For one sequence, there are *m*-*l*+1 number of patterns, so there is in total (*m*-*l*+1)***n* patterns in the input file. For each comparison, *l* nucleotides are compared. So the time complexity is O ((*m*-*l*+1)***n***l*) = O (*m***n***l*) (since in most of the time *l* is much smaller than *m*).

3. To check whether the neighbor is a *tabu-solution*, we have to check whether the changed position is inside the tabu list or not. If the number of iterations that every tabu remains inside tabu list is T, there will be T tabus inside the list. Here we ignore the situation of using Neareast neighbor, otherwise the number of tabus inside the list will be greater than T. Hence the time complexity is O (T);
4. For the worst case, all those standard neighbors cannot meet the requirement, *nearest* neighbor have to be found. This takes O (*m***n***l*) time and only one neighbor will be returned.
5. From 2, 3 and 4, we can get the time complexity for each iteration is O(*l*) * (O(*mnl*) + O (T)) + O(*mnl*) + O(*mnl*) = O (*mnl*^{2}).
6. The algorithm takes *i* iterations to converge. Here, *i* depends on the stop criterion, and is unpredictable. Hence the total time complexity of this algorithm is O (*mnl*^{2}*i*).

4.4.2 Time Complexity of TS Using In-file Neighbors

The algorithm is quite similar as that of using basic neighbors, so the time complexity is also similar except the number of neighbors and the number of tabus inside the tabu list.

1. For neighborhood move operation, we cannot know the number of neighbors accurately, which is denoted as *Q* here.
2. Since each neighborhood move involves several positions in this algorithm, mutiple tabus are added into the tabu list in each iteration. We cannot tell the number of tabus inside the list. However, it is confirmly less than 4*l*.

Based on the discussion of the time complexity of TS using Standard Neighbors, we can tell the time complexity for each iteration of TS using In-file neighbors as:

O(*Q*) * (O(*mnl*) + O (4*l*)) + O(*mnl*) + O(*mnl*) = O (*mnlQ*)

The algorithm takes *i'* iterations to converge. Here, *i'* depends on the stop criterion, and is unpredictable. But from the experiments and the discussion from the section 4.3.4, we know that the *i'* is much smaller than *i*. The total time complexity of this algorithm is O (*mnlQi'*).

4.5 Conclusion

In Tabu Search, algorithm using In-file neighbors performs better. Similar to Simulated Annealing, Tabu Search algorithm using in-file neighbors converges much faster because of the smaller solution space. In contrast to Simulated Annealing, the quality of the solution is as good as what we get from the same algorithm but using basic neighborhood generating mechanism. It is because Tabu Search has the ability to forbid re-visiting the same solutions with the help of the memories. Although the solution space under in-file neighborhood definition is non-continuous, Tabu Search can jump out of the current solution space section whenever it realizes that the most of the solutions in this section have already been visited. So with the help of the *nearest* neighbor, non-continuous solution space has no influence to the Tabu Search result.

Chapter 5 Genetic Algorithm in Motif Discovery

5.1 Overview of Genetic Algorithm

Genetic Algorithm are based on population generation heuristics that borrows its ideas from the natural genetic evolution and diversification. The algorithms are pioneered by John Holland in the 1960s. The algorithm is inspired by the evolutionary ideas of natural selection and evolvement. The essential concepts of Genetic Algorithm aim to simulate processes of survival of fittest as they evolve in natural system. They provide a technical solution suitable for hard optimization problems.

Generally, there are two main components in genetic algorithm: the problem encoding and the evaluation function. Genetic Algorithm can be applied to almost all problems that have large search space. However, these problems may be in quite different styles. To apply Genetic Algorithm on these problems, we have to encode them to the style of gene evolution. Also, to evaluate the goodness of a solution, we have to find a function. That takes the solution as the input and returns a score value.

Genetic Algorithm makes use of a number of current solutions and combines them together to generate new solutions by imitating the genetic process of reproduction. Three fundamental principles are used in GA to create new population, and they are: *Selection*, *Crossover* and *Mutation*. Only gene patterns which are most fit will reproduce and create a new population. This is performed in the Crossover step. The idea behind is that "good" sections of the parents are combined to produce even more fit children. Although many of the children created in this way will not be sufficiently successful to survive the next selection, some will.

Given the input data, we first randomly generate a certain number of solutions, say, S_1, S_2 ... S_n. In the GA terminology, each sample solution is called a *chromosome*. The set of chromosomes is designated as a *population*.

Selection

To each chromosome, we have a *fitness* value from the evaluation function. Stronger individuals, which are those chromosomes with higher *fitness* values, will have greater chance to survive and to reproduce offspring than weaker individuals which will tend to perish. In other words, the algorithm tends to keep good solutions in the population and discard the bad ones.

Crossing over and Mutation

The most important step in Genetic Algorithm is reproduction, which includes “crossing over” and “mutation”.

The content of the two chromosomes participating in reproduction are merged together to form a new chromosome. This heuristic step provides a possible method to get a better child solution from two good parent individuals, which is an evolution. Figure 5.1 gives the idea of “Cross over”.

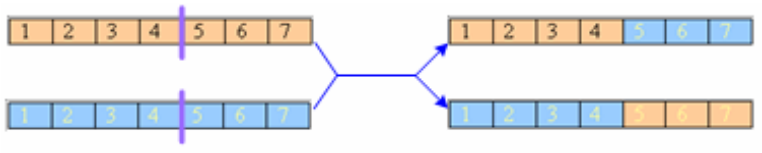

Figure 5.1: Crossover

If we repeat “Selection” and “Cross over”, no new solution space can be explored since these two steps make use of the known domains. This may result in convergence to a local minimum instead of the global minimum. To ensure deviation from the known

domains, the Mutation step is necessary. After the child solution is generated, every fraction of the chromosome is allowed to mutate with a very small possibility (typically 0.001), which is a kind of parthenogenesis in biology. This mutation strategy allows us to explore the whole solution space. With Mutation, new features not known before to the population are generated and they may or may not be beneficial to individuals in the population. However, we hope that in a large population some of these mutations will be beneficial. An example of mutation is shown in Figure 5.2.

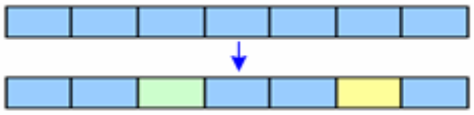

Figure 5.2 Mutation

Figure 5.3 shows the flowchart of standard Genetic Algorithm.

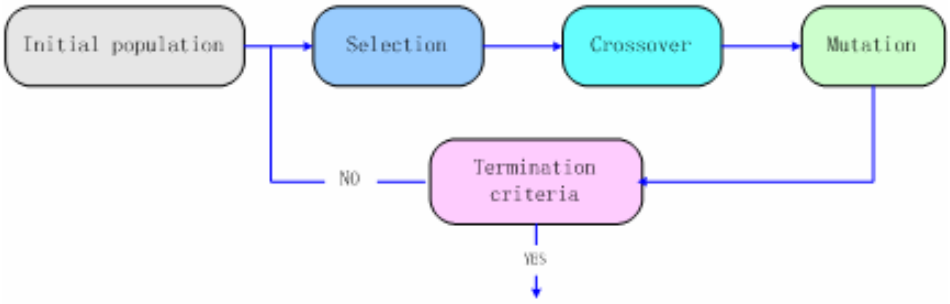

Figure 5.3: Genetic Algorithms Flowchart

Pseudocode of Genetic Algorithm is given in Figure 5.4.

```
ALGORITHM  Genetic_Algorithm
P_ini = random population;
P_current = P_ini;
fitness [P_current] = evaluate (P_current);
While (fitness[P_current] < T) {                  //fitness has not reached desired level
        S_current = select (P_current);           //select fittest chromosomes
        CO_current = crossover (S_current);       //perform crossover on selected pairs
        M_current = mutate (CO_current);          //mutate resulting chromosomes
        P_next = S_current + M_current;           //produce next generation
        P_current = P_next;
        Fitness [P_current] = evaluate (P_current);
}
```

Figure 5.4: Pseudocode of Genetic Algorithm

The Genetic Algorithm uses several criteria to decide when to stop, which are listed below:

a) The number of generations reaches a specified number;

b) The running time of the algorithm reaches some specified amount;

c) The number of generations with no improvement in the fitness function reaches some specified number;

d) The running time of the algorithm with no improvement in the fitness function reaches some specified amount.

e) The fitness of the best solution reaches some defined threshold.

f) The fitness of the population reaches some defined threshold.

5.2 Implementation of Genetic Algorithm

This Genetic Algorithm we are using to solve our problem also belongs to the approaches for consensus motifs. Like other consensus motifs approach, Genetic Algorithm tries to look for a consensus motif first, and then use this consensus motif to scan the sequences to look for those similar motifs to build the motif model.

In contrast to Tabu Search and Simulated Annealing, Genetic Algorithm is a population heuristic algorithm. It needs a number of initial solutions to iteratively generate next generations. The algorithm keeps on merging two parent solutions and mutating the child individuals in order to get better offspring and to approach to the global optima. For each solution, we have a cost associated with it based on which of the three criteria is selected as described in the former content. The goal of the algorithm is to continually improve the fitness of the best solution, as well as the average population fitness. The algorithm stops if the number of generations with no improvement in the fitness function reaches some specified number.

Initial Population: The *initial population* of gene patterns is created randomly from the input sequences. First, we randomly select a sequence and a position in the sequence. A solution pattern can be generated by extracting the substring of the sequence starting from that position with the length equals to the length of the pattern searching for.

Selection: This block extracts a subset of patterns (solutions) from the existing population of patterns, according to the defined fitness. Selection can be performed as described below:

Consider the population where each pattern (solution) has associated fitness. The more fit the pattern, the higher its fitness score. The fitness function is determined by different motif selection criterion. We calculate the mean-fitness of the population. Every individual pattern will be copied to the new population, at frequency proportional to its fitness (relative to the average fitness). For example, when “Any repetition per sequence” is used as the selection criteria, the number of appearance of a pattern will be its fitness. For all the solutions in the population, if the average number of the

appearance is 8.5 (average fitness), and the number of appearance of an individual pattern is 20, we have 20/8.5 ≈ 2.35. This individual pattern will be copied 2 times and also it will have probability of 0.35 to have one more copy in the new population. In our implementation, the size of the population changes dynamically, but it should not differ too much between iterations.

Crossover: The example described in the overview section uses one-point crossover. In our implementation we use a two-point crossover, where we randomly select two positions in parent patterns, cut the parent patterns into three segments and create two children by swapping the segments between the two cutting points. Let *S* be the number of solutions in the population, then *S*/4 solutions are selected out to do the crossover. Figure 5.5 illustrates this strategy.

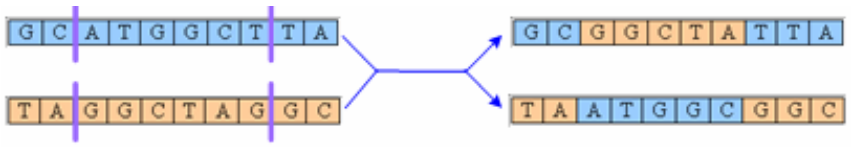

Figure 5.5: Two Points Crossover

Mutation: The last step is the Mutation where we use probability P to change any nucleotide in a pattern to another nucleotide. If the pattern has a length K, the probability of changing to another pattern is $1-(1-P)^K$ (in our implementation we let this value ≈ 0.02).

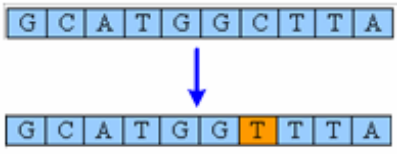

Figure 5.6 Mutation

For this Genetic Algorithm, we did not implement too many extra functions. The flowchart and the pseudocode are similar to the standard ones given in the previous section.

5.3 Time Complexity of Genetic Algorithm

Let S = number of solutions in the initial population.

During the search, in each iteration, the numbers of solutions are not differ too much, so we can assume the number are O (Sl).

a) To generate the initial population, O(S) time is needed;

b) In the step of Selection, the fitness of every solution needs to be calculated. The time complexity of this step is O($S*m*n*l$), because, as discussed in the previous chapter, to evaluate the fitness of each solution, we have to compare it with all the patterns in the input file, which take O ($m*n*l$) times.

c) In the step of Crossover, since O(S) solutions are taken out to do the crossover, O(Sl) time is used.

d) Mutation will be applied to every solution, so this step takes O(Sl) time.

Hence the total time complexity of our Genetic Algorithm is O($RSmnl$), where O(R) is the time to converge and it is unpredictable.

5.4 Conclusion

Genetic algorithm operates on entire populations of candidate solutions in parallel, which is one of the main strengths of the genetic approach. This parallel operation feature implies that Genetic Algorithm is much more likely to locate the global optima than the traditional techniques, which iteratively refine a single solution, because they are much less likely to get stuck at local optima. Moreover, due to the parallel operation, the performance of the algorithm is much less sensitive to the initial conditions. The

performance of Genetic Algorithm is at least as good as a purely random search, because it makes hundreds, or sometimes even thousands, of initial guesses. Hence, when the search space is large, complex or poorly understood, Genetic Algorithm could be very useful and frequently sufficiently efficient.

The convergence of a GA is usually slower than traditional techniques. In fact, with a good initial guess close to the global optimum, traditional techniques are much faster and more accurate than a genetic search. Another problem of Genetic Algorithm is that although the solutions found more likely to estimate the global optimum, most of the time they are only estimate. From the discussion above, we know that Genetic Algorithm never searches the small neighborhood around the current solution like what Tabu Search and Simulated Annealing do. Users must realize that Genetic Algorithm only finds an exact optimum by chance, whereas traditional algorithms (such as Tabu Search and Simulated Annealing) find it exactly.

Genetic algorithm can be used in a way that enhances and complements the traditional methods. Genetic approaches, in particular, are now available to optimize difficult, NP-hard objective functions. Furthermore, these genetic approaches are often simple to design and easy to code, and can be used in concert with traditional methods to greatly increase the probability of finding the true global optimum.

Chapter 6 Dragon Motif Builder

Dragon Motif Builder (DMB) is a motif search tool. This application is for detecting short DNA motifs from a set of unaligned DNA sequences. This tool is developed based on four different heuristic algorithms in the format of web-based application. Simulated Annealing, Tabu Search and Genetic Algorithm have already been discussed in the previous chapters. The other algorithm is Expectation Maximization, which is based on the idea of maximum likelihood estimation.

A public server is free for academic users, and can be found at http://sdmc.i2r.a-star.edu.sg/DRAGON/Motif_Search/ (Yang *et al*. 2005).

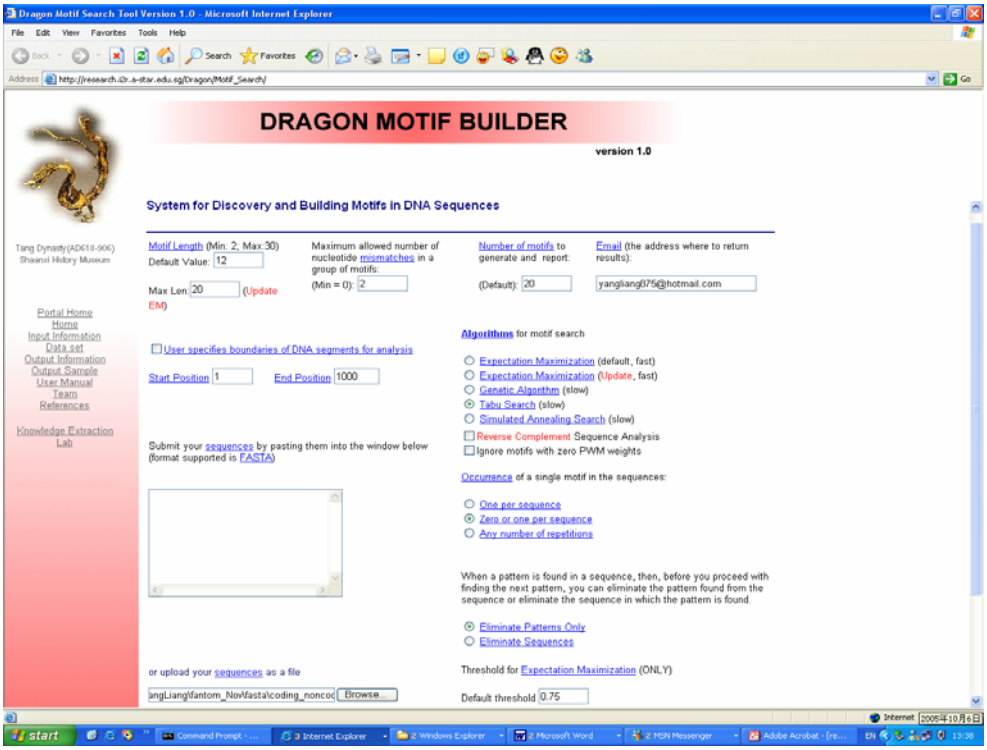

Figure 6.1 Screen Print of Dragon Motif Builder

DMB aims to provide a free-access tool for the biologists to analyze the biological sequences. Figure 6.1 shows the main page of this web-based software tool. It allows users to submit the sequences to the UNIX server for analysis by using web browser, and to receive the result report through Email. Figure 6.2 explains the processing and system platforms of DMB.

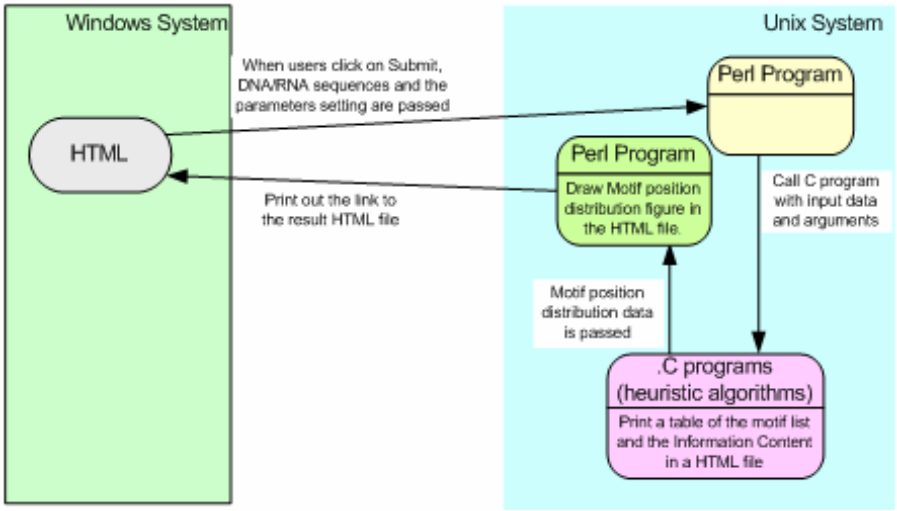

Figure 6.2 DMB Processing

6.1 Functions in DMB

In the following content of this section, we will see how those functions in the software can be used through different parameter setting.

Input File

In order to use the tool, users should provide a set of DNA sequences in the FASTA format, either in ACGT or acgt alphabet. These sequences can be either pasted to the main sub-window provided, or browsed from the disk.

The maximum number of sequences DMB can support is 10,000 and the length of the individual sequence must be less than 3,500 nucleotides. Currently, Dragon Motif Builder only can detect input sequences in the FASTA format:

```
>sequence name
sequence as one / more lines
```

Motif Length

Motif length indicates the length of DNA subsequence you expect to represent the motif. The default is 8 nucleotides, and the range is from 4 to 30 nucleotides.

Maximum allowed number of nucleotide mismatches in a group of motifs

This number indicates the maximum number of nucleotides different from the consensus motif which the algorithm can tolerate while grouping motifs.

Number of motifs to generate and report

This is the number of motifs that users want to be discovered and extracted from the input data. The default value is 1.

Email address

Users' Email addresses are required in order to receive the analysis results, because this kind of problem is quite time consuming and users may not be patient enough to wait for the result. The submissions will be stored on the server and processed, and it takes some time to search for the best groups of motifs with the heuristic algorithms. In this case, E-mail address is required so that the result file could be sent back to the users even the browser has been closed. Without this email information, the analysis will not start.

Algorithms for motif search

You can select any one of the heuristic algorithms you like: Tabu Search, Simulated Annealing or Genetic Algorithm.

Occurrence of a single motif in the sequences

This is the criteria described in the problem definition section. When “One per sequence” is chosen, it means that each sequence must contain ONE similar motif, and only the best motif is chosen. When “Zero or one per sequence” is chosen, it means that the motif may or may not appear in the sequences. Only ONE best motif can be picked out from one sequence. “Any number of repetitions” means that the motifs may or may not appear in the sequences. Any number of motifs can be picked out from one sequence.

Eliminating the selected motifs

When the user wants more than one motif in the search report, these are two methods of eliminating the selected Motifs from the sequences. When “Eliminate Patterns Only” is selected, it means that the patterns, which were chosen for the previous motifs, will be excluded in the next search iteration. When the option of “Eliminate Sequences” is selected, it means that those sequences with the previous selected motifs appeared will be excluded.

Analysis specifies boundaries of DNA segments

If the sequences are aligned, it is possible to select the segment for submitted sequences to be analyzed. To use this feature, users need to check the square box before the “User specifies segment for analysis” and then specify the start and end positions of sequences for the analysis. One checkbox to induce the double-stranded search for all the algorithms

The rest options and parameters not mentioned here are for Expectation Maximization Algorithm, and not included in this thesis.

After pressing the 'submit' key the file or pasted sequences will be transmitted to the server and further processed.

6.2 Motif Report

In the result page, the motifs are identified by groups. One group for each iteration. Figure 6.2 shows the motif report for the 2nd iteration. The report has three parts, which are a) a list of motifs in the group; b) analysis of the group; c) and a graph representation of the position distribution.

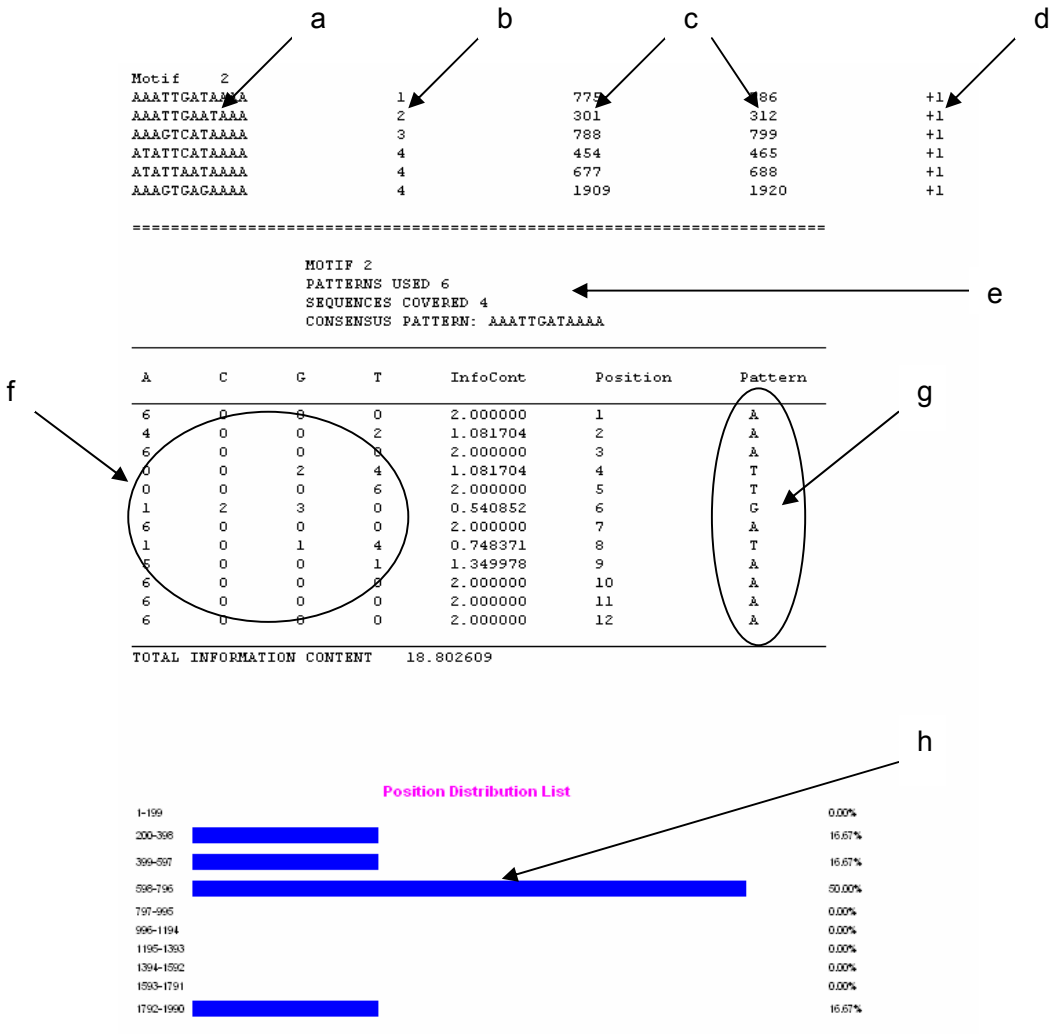

Figure 6.3 An Example of Motif Report

An example of one motif report is given in Figure 6.2. Inside the figure:

a: Similar specific motif patterns, which have been grouped together

b: The specific sequence, from which the motif selected

c: The start and end position of the motif in the sequence

d: The appearance of the motif in the sequence. +1 for forward pattern and -1 for reverse compliment pattern.

e: Some description of the motif group, such as: iteration number, the number of motifs in the group, the number of sequences those motifs selected from and the consensus motif.

f: The summary of the motif group in term of position weight matrix

g: Consensus motif of the group.

h: The percentage of each position range

Chapter 7 Experiment Result

To test how accurate our algorithms predict the Transcription Factor Binding Sites (TFBS), we use the benchmark data set provided by Tompa (2005), which can be downloaded from the website http://bio.cs.washington.edu/assessment/. We compare the motifs found based on our motif search algorithms, which are Simulated Annealing, Tabu Search, Genetic Algorithm and those existing algorithms, which are MEME, QuickScore, Weeder, GLAM, Improbizer, Consensus and AlignACE. These algorithms already have test results in Tompa's paper. With the comparison, we will verify the algorithm we proposed. We will assess how well our algorithms (SA, TS and GA) recognize such known biological sequences as compared to those well-known motif discovery algorithms.

7.1 Experiment Data Set

The benchmark data sets we are going to use to test the algorithms are from three types: a) Real genomic promoter sequences that are used as the 'real' data set. These sequences contain annotated real TFBSs. There is unfortunately a drawback of using this data set since we do not know what other real TFBSs are located in the promoters. Thus, if the algorithm correctly predicts those non-annotated binding sites, it will be penalized. b) The 'Markov' data set contains the sequences that are randomly generated using Markov chain and we assume that we already know the complete correct answer. c) The randomly chosen sequences from the same genome represent the 'generic' data.

7.2 Parameters Setting in DMB

a) Motif Length: 12 (SA and TS) and 8 (GA)

TFBSs are generally known to have a length within the range of 10-15 nucleotides. Hence a motif length of 12 should be a good length to be chosen for searching motifs. Another reason for us to choose this number instead of a smaller one is the definition of "correct prediction of site" in our experiment (also defined in Tompa's paper). A predicted site is considered overlapping a known site only if they overlap by at least ¼ the length of the known site. And from the benchmark data set given, some of the TFBSs are very long up to 30, so to predict such long motifs, the minimum length of the motifs is 8 with no mismatch allowed. That is why we choose a longer motif.

Unfortunately, our GA implementation has restriction to search for motifs of length of maximally 8 nucleotides. It is because when the motif is long, the initial population will be exponentially large. Although, we will definitely miss some of the correct predictions, the result is still quite good, which is shown in the next section.

b) Maximum allowed number of nucleotide mismatches in a group of motifs: 2
Eliminating the selected Motifs: choose "Eliminate Patterns Only".
In the data sets, most of the files contain only few sequences (less than 10).

c) Number of motifs to generate and report: 10
The number of motifs generated by most of the existing algorithms (used to compare with our algorithms) is within the range from 5 to 10. We choose to extract 10 motifs from our input sequences. This is because the length of our input sequences is within the range of 1500 to 2000 nucleotides. We set the size length of motif search as 12 nts (nucleotides/residues). Hence a coverage of 120+ nucleotides (12 * 10) including the distance between the motifs was reasonable area to be covered for finding significant motifs.

d) Algorithms for motif search: We try SA, TS and GA

e) Occurrence of a single motif in the sequences: Any number of repetitions.

As described above, most of the date files contain few sequences (some of them only have 1 or 2 sequences). It is not the case that each sequence comes from the different species and contains the same TFBSs, so “Any number of repetitions” should be a reasonable choice.

f) Choice of the result motifs:

We choose the whole group of motifs instead of choosing the consensus motif, and no post-process will be done to filter out those false predicted solution motifs.

7.3 Comparison Result

Here, we use the same comparison rule as what described in Tompa’s paper (2005). For each system that we denote as T (tool) and each data set that we denote as D, the accuracy of T on D can be assessed at the nucleotide level and at the site level. For the assessment at the nucleotide level we denote by:

nTP be the number of nucleotide positions in both known sites and predicted sites,

nFN be the number of nucleotide positions in known sites but not in predicted sites,

nFP be the number of nucleotide positions not in known sites but in predicted sites, and

nTN be the number of nucleotide positions in neither known sites nor predicted sites.

One can say that a predicted site overlaps a known site if they overlap by at least 1/4 the length of the known site. At the site level we can thus define:

sTP be the number of known sites overlapped by predicted sites,

sFN be the number of known sites not overlapped by predicted sites, and

sFP be the number of predicted sites not overlapped by known sites.

At either the nucleotide (x=n) or site (x=s) level one can then define

Sensitivity: xSn = xTP / (xTP + xFN), and

Positive Predictive Value: xPPV = xTP / (xTP + xFP).

The sensitivity gives the fraction (probability) of known sites (or site nucleotides) that are predicted, and the positive predictive value gives the fraction of predicted sites (or site nucleotides) that are known.

At the nucleotide level one can also define

Specificity: nSp = nTN / (nTN + nFP).

Finally, various single statistics can be considered that in a sense average (some of) the score measures mentioned. We define (Tompa 2005) the (nucleotide level) performance coefficient as

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

the (nucleotide level) correlation coefficient as

nCC = (nTP nTN - nFN nFP) / √((nTP+nFN)(nTN+nFP)(nTP+nFP)(nTN+nFN)) ,

and the (site level) average site performance as

sASP = (sSn + sPPV) / 2.

The test results can be found in Appendix A (Simulated Annealing) and Appendix B (Tabu Search). Figure 7.1 to 7.5 show us the analysis result of Sensitivity, Positive Predictive value, Performance/Correlation Coefficient value, Average performance value

and Specificity of different algorithms, and all of these analysis values are already been normalized.

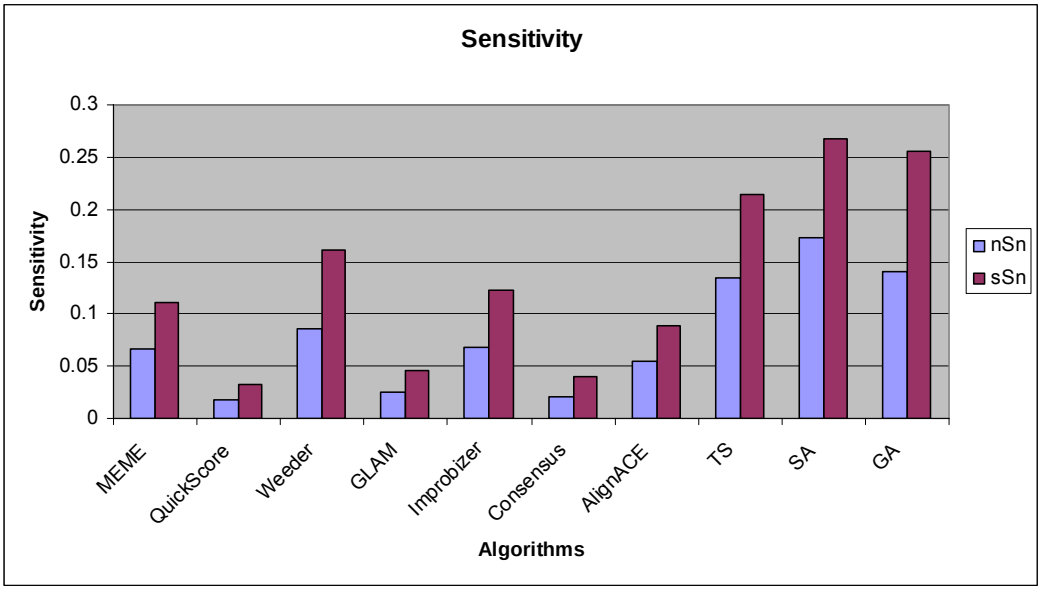

Figure 7.1 Sensitivity of Different Algorithms

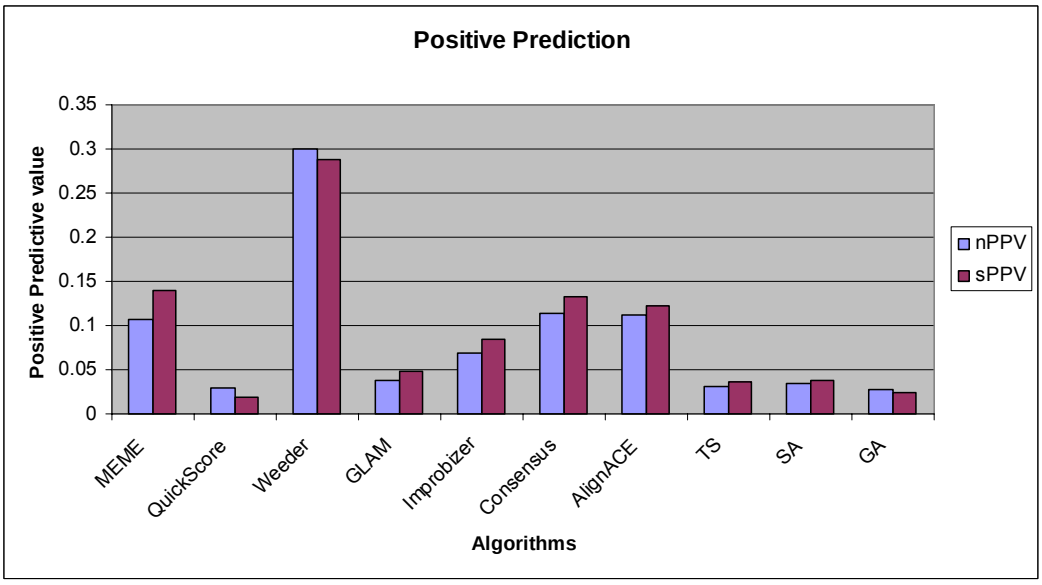

Figure 7.2 Positive Predictive Values of Different Algorithms

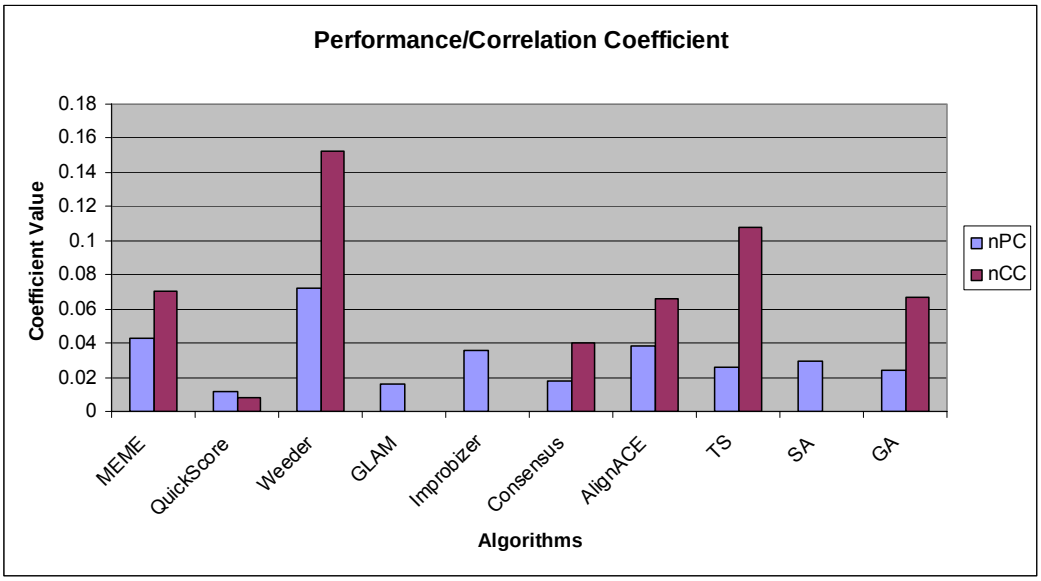

Figure 7.3 Performance/Correlation Coefficient of Different Algorithms

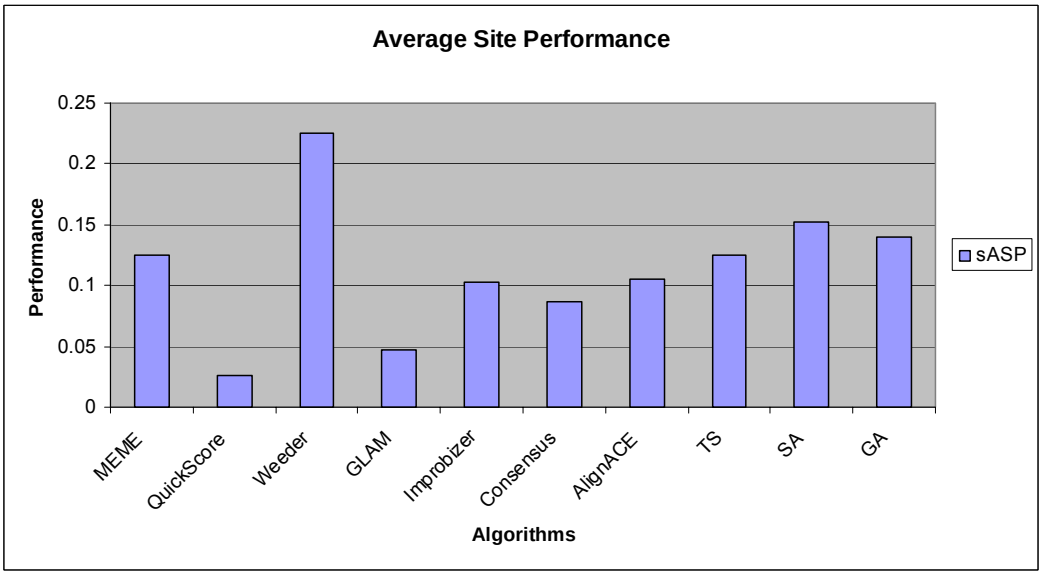

Figure 7.4 Average Site Performance of Different Algorithms

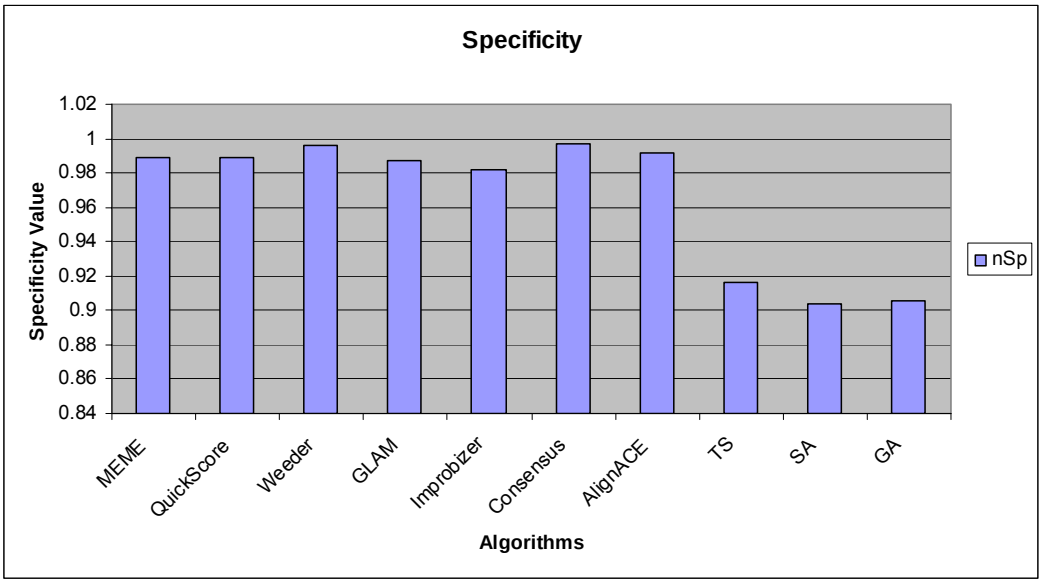

Figure 7.5 Specificity of Different Algorithms

7.4 Discussion

7.4.1 Performance of Simulated Annealing, Tabu Search and Genetic Algorithm

From the 5 analyzed figures in section 7.3, we can see that our algorithms (SA, TS and GA) performance quite good on the benchmark data set. They have the highest sensitivity. They can correctly predict more TFBSs compared with other well-known motif search algorithms.

However, our algorithms all have very small Positive Predictive values, and they are only larger than that of QuickScore. This means that our algorithms reports proportionally more noisy results (false TFBSs). As described in the previous section (Parameters setting in DMB), we choose the whole group of motifs instead of choosing the consensus motif. Because it is very hard to know which motif from the group is the correctly predicted TFBS and if we choose the wrong one, we will miss the correct 'answer' even

we really did find it. On the other hand, it is not possible that all the motifs in the group represent correct 'answer' (in fact, only a very small portion of solution TFBSs in the group are correct), and that is why our results have a high false prediction rate. While in Tompa's paper, all the experiment results were provided by the search algorithms' authors themselves and many algorithms did some post-process to filter out those false predicted solution motifs. In parameters setting of the experiments in their website, it is said that the final results were selected from the result motif group by eyes based on their biology knowledge, such as for MEME, QuickScore, GLAM etc. It is thus impossible to compare the results of automated analysis as ours with theirs. To improve the Positive Prediction, some work can be done, which is discussed in the "further work" section in the next Chapter.

Our algorithms all have middle ranged Performance Coefficient. They are comparable with that of others. TS has the second highest Correlation Coefficient, however SA has a very low Correlation Coefficient, which is almost 0.

SA, GA and TS have the second, third and the fourth highest Average Site Performance value respectively. This means that they have picked up the significant portion of the real TFBSs.

Our algorithms all have slightly lower Specificity values than all other programs. The Specificity of our algorithm is nTN/(nTN+nFP). However, some algorithms did the manual elimination of predictions they considered not good. If *y* false predictions are filtered, their Specificity becomes (nTN-*y*)/(nTN-*y*+nFP), which is greater than that of our algorithms'.

However, as explained before, some of the authors of Tompa's study have done manual elimination of predictions they considered not good. This process is not explained in any detail and thus makes not possible to compare our 'raw' results with theirs. We believe that if such manual cleaning of predictions is made, our results could well be somewhere in the upper group of better performing predictors. As it stands now, the results are definitely better than at least one of the existing algorithms, which is QuickScore.

7.4.2 Comparison between Approaches for Consensus Motifs and Approaches for Profile Motifs

Generally, from the analyzed figures, approaches for consensus motifs (Weeder, SA, TS and GA) perform much better than approaches for profile motifs (MEME, GLAM, CONSENSUS etc.) in term of sensitivity.

After taking a close look at the difference between the two groups of algorithms, we found that the accuracy difference due to the different form in which they express the similarity between the two motifs. For example, MEME uses Expectation Maximization algorithm. As discussed in Chapter 2, the basic idea is that MEME will return those motifs that have high scores by comparing with the Position Weight Matrix (PWM), and the consensus motif is the one with the highest score. The similarity requirement in this case is very loose. They look at PWM as the background, and two motifs are similar and can be grouped together if both of them have high score by comparing with the same PWM. Since they only look at the information content, that is why motifs in the same group found by approaches for Profile Motifs can have a large difference. However, it is not possible that the binding site motifs mutate a lot.

On the other hand, in our similarity definition, we compare two motifs directly. We say that two motifs are similar and can be grouped together if the similarity score between them, which comes from by calculating the number of matched nucleotides, is higher than our defined threshold. That is why the motifs in one group found by approaches for consensus motifs are much more compact than that found by approaches for profile motifs.

Chapter 8 Conclusions and Future Work

We provide three heuristic algorithms, which are based on Simulated Annealing, Tabu Search and Genetic Algorithm, for this motif discovery problem. All of these algorithms have the ability to escape from the local optimum and search for the global optimal solutions. The algorithms and the program structures have been presented in detail in the thesis. At the same time, a web-accessible Motif Search tool is also implemented based on our three heuristic algorithms. It is free for academic users and can be found at http://sdmc.i2r.a-star.edu.sg/DRAGON/Motif_Search/.

From the comparison result discussed in the previous chapter, we conclude that SA, TS and GA are very useful algorithms to the motif discovery problem. They perform much better than those existing algorithms in terms of sensitivity; they also perform better than several other programs based on some other measures of prediction success. Our algorithms can correctly predict more TFBSs compared with other motif search algorithms. However we get low positive predictive values for our algorithms. As explained in section 7.4.1, it is because we do not do any manual elimination of false prediction to our algorithms as some of the authors of Tompa's study do. As it stands now, the results are definitely better than at least one of the existing algorithms, which is QuickScore. However, we still have to do much more experiments before we can draw any definite conclusion.

In practice, motif discovery algorithms have to take into account characteristics of the input data. These include: the length of the unknown motifs; corrupted samples (some sequence may not contain a motif); invaded sample (some sequence may contain more

than one instance); multiple patterns (some sequence may contain more than a single common pattern); what strand the given sequence lies on; etc.

To increase the accuracy of finding the TFBSs, we can introduce the biological features into our algorithms. Here is an example. In fact, our software (DMB) has many other functions. To increase the accuracy of finding the TFBSs, we provided Reverse Complement Sequence Analysis. We know that DNA has a double-stranded structure, where one strand is considered ‘direct’ and the other ‘complementary’ one. These two strands are complementing each other. So, for example, when we use motif TATACCG as a consensus motif to find the similar motifs in the input sequences, it is also necessary to look for those motifs which are similar to ATATGGC, which is a complementary motif for the first one. We look at these two groups of motifs as having the same weight.

Here are some other ideas that can be introduced into our algorithms to improve the performance:

a) The specific binding sites could be located in the same region in promoters. For example, some binding sites may always appear in the position -500 to -200 realtive to TSS.

b) The binding sites in the sequences could be located in the same order. We can try to find the Maximum Weight Common Subsequence for filtering.

c) If the motifs in one group share a great similarity with the consensus motif, this group should have high Information Content (IC). Then we can choose in motif selection between different iterations those that increase the information content.

Bibliography

[1] Ao W., Gaudet J., Kent W.J., Muttumu S. and Mango S.E. Environmentally induced foregut remodeling by PHA-4/FoxA and DAF-12/NHR. *Science* 305, 1743-1746, 2004

[2] Bailey T.L and Elkan C. Fitting a mixture model by expectation maximization to discover motifs in biopolymers', *Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology* (ISMB'94), pp. 28-36, AAAI Press, Menlo Park, California, August, 1994.

[3] Bailey T.L. and Elkan C. The value of prior knowledge in discovering motifs with MEME. *Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology*. 21-29 (AAAI Press, Menlo Park, CA, 1995). 1995a

[4] Bailey T.L. and Elkan C. Unsupervised learning of multiple motifs in biopolymers using expectation maximization, *Machine Learning*, 21:51-80, 1995b

[5] Bajic V.B., Huang E, Yang L, Modeling methodology for detection of regulatory motifs in DNA/RNA and proteins, *Int.J.Comp.Syst.Signals,* (accepted) 2004

[6] Barbulescu L, Watson J.P. and Whitley L.D., Dynamic Representations and Escaping Local Optima: Improving Genetic Algorithms and Local Search. *Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, pp: 879-884. 2000

[7] Booker L, Improving Search in Genetic Algorithms. *Genetic Algorithms and Simulating Annealing*, Davis L, ed. Morgan Kaufman, pp. 61-73. 1987

[8] Brazma A, Jonassen I, Eidhammer I, and Gilbert D. Approaches to the automatic discovery of patterns in biosequences. Journal of Computational Biology, 5:279-305, 1998.

[9] Brejova B., Marco C.Di, Vinaf T. and Romero S. *Finding patterns in biological sequences.* Research report, 2000

[10] Buhler J. and Tompa M. Finding motifs using random projections. In Proceedings of *the 5th Annual International Conference on Computational Molecular Biology* (RECOMB01), pages 69-76, 2001.

[11] Casella G and George E.I., "Explaining the Gibbs sampler", *The American Statistician*, 46(2):167-174, 1992.

[12] Davis L.D., Handbook of Genetic Algorithms. *Van Nostrand Reinhold*. 1991

[13] Denning P.J. "Genetic Algorithm," *American Scientist*, Vol. 80, pp. 12-14. 1982

[14] Dowsland K. Simulated annealing, in *Modem Heuristic Techniques for Combinatorial Problems* (C Reeves, editor), New York, John Wiley & Sons. 1993

[15] Eglese R.W. Simulated annealing: a tool for operational research, *European Journal of operational Research*, Vol. 46, No. 3. June 15, pp. 271 – 281. 1990

[16] Eskin E and Pevzner P, "Finding composite regulatory pattern in DNA sequence", Bioinformatics, 18:354-363, 2002.

[17] Favorov A.V., Gelfand M.S., Gerasimova A.V., Mironov A.A. and Makeev V.J. Gibbs sampler for identification of symmetrically structured, spaced DNA motifs with improved estimation of the signal length and its validation on the ArcA binding sites. *Proceedings of BGRS 2004* (BGRS, Novosibirsk, 2004)

[18] Fleischer M, Simulated annealing: past, present, and future, pages: 155 – 161, *ACM Press*, New York, NY, USA 1995

[19] Frith M.C., Hansen U., Spouge J.L. and Weng Z. Finding functional sequence elements by multiple local alignment. *Nucleic Acids Res.* 32, 189-200, 2004

[20] Glover F, Tabu Search - Part I. *ORSA Journal on Computing*, 1: 190-206, 1989

[21] Glover F, Tabu Search - Part II. *ORSA Journal on Computing*, 2: 4-32, 1990

[22] Glover F and Laguna M. Modern Heuristic Techniques for Combinatorial Problems, *Colin R. Reeves* (Ed.), 70-150, Blackwell Scientific Publications, Oxford, 1993.

[23] Glover F and Laguna M, *Tabu Search*, Kluwer Academic Publisher, 1997

[24] Goldberg D.E. Genetic Algorithms in Search, Optimization & Machine Learning**,** Addison-Wesley. 1989

[25] Hertz G.Z., Stormo G.D. Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. *Bioinformatics*. 1999 Jul-Aug; 15(7-8): 563-77. 1999

[26] http://bioinformatics.org/faq/

[27] Huang E, Yang L, Chowdhary R, Kassim A, Bajic V.B., An algorithm for *ab initio* DNA motif detection, Chapter 4 in *Information Processing and Living Systems*, *World Scientific*, 611-614, 2005

[28] Hughes, J.D., *et al*. Computational identification of *cis*-regulatory elements associated with functionally coherent groups of genes in *Saccharomyces cerevisiae*. *J. Mol. Biol*. 296, 1205-1214, 2000

[29] Ingber L., "Simulated annealing: Practice versus theory," *Computer Modelling*, vol. 18, pp. 29–57, 1993.

[30] Ingber L., "Adaptive simulated annealing (ASA): Lessons learned," *Control and Cybernetics*, vol. 25, pp. 33–54, 1996.

[31] Johnson D.S., Aragon C.R., McGeoch L.A., Schevon C. (1989) Optimization by Simulated Annealing: An Experimental Evaluation. *Operations Research*, 37(6): 865-892. 1989

[32] Kirkpatrick S, Gelatt C, Vecchi M. Optimization by Simulated Annealing. *Science*, 220(4598): 671-680. 1983

[33] Koza, J.R. Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press. 1992

[34] Koza, J.R. Genetic Programming II: Automatic Discovery of Reusable Programs. MIT Press. 1994

[35] Krishnan SPT, E Huang, L Yang, V B Bajic, Statistical Properties of region around PolyA sites in Human, *5th HUGO Pacific meeting and 6th Asia Pacific meeting on Human genetics, 17-20 November 2004, Singapore.* 2004

[36] Latchman D, Gene Regulation: a eukaryotic perspective. Nelson Thornes Ltd, Fourth edition, 2002.

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

[38] Lawrence C.E. and Reilly A.A. An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences. *Proteins*. 1990; 7(1):41-51.

[39] Liang C. COPIA: A new software for finding consensus patterns in unaligned protein sequences. *Master thesis, University of Waterloo*, 2001.

[40] Liu J, A Combinatorial Approach for Motif Discovery in Unaligned DNA Sequences. *PhD thesis University of Waterloo*, 2004

[41] Mitchell M. An introduction to Genetic Algorithms. *MIT Press*. Cambridge, Massachusetts. London, England. 1996

[42] Mühlenbein H, How genetic algorithms really work: I. Mutation and Hillclimbing, Parallel Problem Solving from Nature -2-, R. Männer and B. Manderick, eds. *North Halland*. 1992

[43] Pavesi, G. *et al*. Weeder Web: discovery of transcription factor binding sites in a set of sequences from co-regulated genes. Nucleic Acids Res. 32, W199-W203, 2004

[44] Pevzner P.A. and Sze S. Combinatorial approaches to finding subtle signals in DNA sequences. In Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology, pages 269-278, 2000.

[45] Randall M. and Abramson D., "A general parallel Tabu Search algorithm for combinatorial optimization problems," Part '99: Proceedings of the 6th Australasian Conference on Parallel and Real Time Systems, Cheng W. and Sajeev A.(Eds.), *Springer-Verlag*, Singapore, 1999, pp. 68–79.

[46] Reeves C.R. Genetic algorithms. In *Modern Heuristic Techniques for Combinatorial Problems*, 151-196. *Oxford: Blackwell Scientific Publications*. 1993

[47] Reeves C.R. Genetic algorithms for the Operations Researcher. *INFORMS J. Comp*. 9: 231-250, 1997

[48] Régnier M. and Denise A. Rare events and conditional events on random strings. *Discrete Math. Theor. Comput. Sci*. 6, 191-214, 2004

[49] Rigoutsos *et al*, "The Emergence of Pattern Discovery Techniques in Computational Biology." *Metabolic Engineering*. 2(3):159-177, July 2000

[50] Sagot M.F. Spelling approximate repeated or common motifs using a suffix tree. LATIN'98: Theoretical Informatics, pages 111-127, 1998. Lecture Notes in Computer Science.

[51] Sinha S and Tompa M, "A statistical method for finding transcription factor binding sites". *In Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology*, AAAI Press, pages 344-354, 2000.

[52] Sinha S and Tompa M, "Performance comparison of algorithms for finding transcription factor binding sites". *In 3rd IEEE Symposium on Bioinformatics and Bioengineering*, IEEE Press, pages 214-220, 2003.

[53] Stormo G.D. DNA binding sites: representation and discovery. Bioinformatics, 16:16-23, 2000.

[54] Thijs G. *et al*. A higher-order background model improves the detection of promoter regulatory elements by Gibbs sampling. *Bioinformatics* 17, 1113-1122, 2001.

[55] Tompa M, Li N, Bailey T.L., Church G.M, Moor B.D *et al*. Assessing Computational Tools for the Discovery of Transcription Factor Binding Sites. Nature Biotechnology, Vol. 23, no. 1, 137-144.

[56] Tovey C. Simulated annealing. *American Journal of Mathematical and Management Sciences*, 8(3&4): 389-407. 1988

[57] Werner T. Models for prediction and recognition of eukaryotic promoters. *Mamm Genome*. Feb; 10(2):168-75. 1999

[58] Yang L, Huang E, Bajic V.B., Some implementation issues of heuristic methods for motif extraction from DNA sequences, *Int.J.Comp.Syst.Signals*, 5(2) (in print) (2005)

Appendix A: Test Result of Simulated Annealing

Data set	nTP	nFP	nFN	nTN	sTP	sFP	sFN		nSn	nPPV	nSp	nPC	nCC	sSn	sPPV	sASP
dm01g	21	735	104	5140	2	61	5		0.168	0.0277778	0.8748936	0.0244186	0.0184612	0.2857143	0.031746	0.1587302
dm02r	0	360	49	1591	0	30	5		0	0	0.8154792	0	-0.0742504	0	0	0
dm03m	0	552	104	5344	0	46	9		0	0	0.9063772	0	-0.0422755	0	0	0
dm04g	7	833	128	7032	1	69	8		0.0518519	0.0083333	0.8940877	0.0072314	-0.0227143	0.1111111	0.0142857	0.0626984
dm05g	1	671	159	6669	0	56	14		0.00625	0.0014881	0.9085831	0.0012034	-0.0430871	0	0	0
dm06r	3	405	95	2497	0	34	7		0.0306122	0.0073529	0.8604411	0.0059642	-0.0564974	0	0	0
dm07m	0	480	0	4020	0	40	0		NaN	0	0.8933333	0	NaN	NaN	0	NaN
dm08m	0	636	0	5364	0	53	0		NaN	0	0.894	0	NaN	NaN	0	NaN
hm01g	0	3144	236	32620	0	262	16		0	0	0.9120904	0	-0.0251285	0	0	0
hm02r	40	812	217	7931	3	67	8		0.155642	0.0469484	0.9071257	0.0374181	0.0357098	0.2727273	0.0428571	0.1577922
hm03r	48	1116	360	13476	4	93	11		0.1176471	0.0412371	0.9235197	0.0314961	0.0250295	0.2666667	0.0412371	0.1539519
hm04m	2	1390	166	24442	1	115	10		0.0119048	0.0014368	0.9461908	0.0012837	-0.0149154	0.0909091	0.0086207	0.0497649
hm05r	32	496	161	2311	2	40	9		0.1658031	0.0606061	0.8232989	0.0464441	-0.0070211	0.1818182	0.047619	0.1147186
hm06g	24	540	51	3885	2	45	7		0.32	0.0425532	0.8779661	0.0390244	0.076544	0.2222222	0.0425532	0.1323877
hm07m	1	599	126	4274	0	50	6		0.007874	0.0016667	0.8770778	0.0013774	-0.0557029	0	0	0
hm08m	52	644	138	6666	6	53	7		0.2736842	0.0747126	0.9119015	0.0623501	0.1005063	0.4615385	0.1016949	0.2816167
hm09g	0	1200	160	13640	0	100	10		0	0	0.9191375	0	-0.0306192	0	0	0
hm10m	13	383	76	2528	2	31	9		0.1460674	0.0328283	0.8684301	0.0275424	0.0072668	0.1818182	0.0606061	0.1212121
hm11g	19	881	250	6850	1	74	18		0.070632	0.0211111	0.8860432	0.0165217	-0.0247161	0.0526316	0.0133333	0.0329825
hm12r	7	257	63	673	1	21	4		0.1	0.0265152	0.7236559	0.0214067	-0.1020729	0.2	0.0454545	0.1227273
hm13r	0	600	164	5236	0	50	9		0	0	0.8971899	0	-0.0558783	0	0	0
hm14r	12	336	70	1582	1	28	3		0.1463415	0.0344828	0.8248175	0.0287081	-0.0150851	0.25	0.0344828	0.1422414
hm15r	0	876	90	7034	0	73	4		0	0	0.8892541	0	-0.0374044	0	0	0
hm16g	0	2028	164	18808	0	169	7		0	0	0.9026685	0	-0.0290063	0	0	0
hm17g	79	533	66	4822	7	44	3		0.5448276	0.129085	0.9004669	0.1165192	0.2268662	0.7	0.1372549	0.4186275
hm18m	0	1068	88	13844	0	89	7		0	0	0.9283798	0	-0.0212693	0	0	0
hm19g	12	516	75	1897	1	43	3		0.137931	0.0227273	0.7861583	0.0199005	-0.0340856	0.25	0.0227273	0.1363636
hm20r	339	4989	967	63705	33	410	43		0.2595712	0.0636261	0.9273736	0.0538523	0.0953903	0.4342105	0.0744921	0.2543513
hm21g	2	634	91	4273	0	53	7		0.0215054	0.0031447	0.8707968	0.002751	-0.0436702	0	0	0

hm22m	43	401	63	2493	3	32	2		0.4056604	0.0968468	0.8614375	0.0848126	0.1388675	0.6	0.0857143	0.3428571
hm23r	20	292	123	1565	2	24	3		0.1398601	0.0641026	0.8427571	0.045977	-0.0123432	0.4	0.0769231	0.2384615
hm24m	33	411	59	3497	4	32	4		0.3586957	0.0743243	0.8948311	0.0656064	0.1209825	0.5	0.1111111	0.3055556
hm25g	8	220	62	710	1	18	4		0.1142857	0.0350877	0.7634409	0.0275862	-0.0743612	0.2	0.0526316	0.1263158
hm26m	33	735	214	8018	4	60	6		0.1336032	0.0429688	0.9160288	0.0336049	0.0290239	0.4	0.0625	0.23125
mus01r	9	267	76	1148	1	22	5		0.1058824	0.0326087	0.8113074	0.0255682	-0.0494113	0.1666667	0.0434783	0.1050725
mus02r	24	876	208	7892	2	73	10		0.1034483	0.0266667	0.9000912	0.0216606	0.0018697	0.1666667	0.0266667	0.0966667
mus03g	21	363	121	1995	2	30	7		0.1478873	0.0546875	0.846056	0.0415842	-0.003888	0.2222222	0.0625	0.1423611
mus04m	24	480	240	6256	2	40	12		0.0909091	0.047619	0.9287411	0.0322581	0.0144821	0.1428571	0.047619	0.0952381
mus05r	18	306	70	1606	2	25	4		0.2045455	0.0555556	0.8399582	0.0456853	0.0247726	0.3333333	0.0740741	0.2037037
mus06g	23	277	44	1156	4	22	1		0.3432836	0.0766667	0.8066992	0.0668605	0.0774552	0.8	0.1538462	0.4769231
mus07g	8	928	92	4972	1	77	3		0.08	0.008547	0.8427119	0.0077821	-0.027268	0.25	0.0128205	0.1314103
mus08m	0	432	41	4027	0	36	3		0	0	0.9031173	0	-0.0312482	0	0	0
mus09r	24	240	17	719	1	20	1		0.5853659	0.0909091	0.7497393	0.0854093	0.1507445	0.5	0.047619	0.2738095
mus10g	84	912	139	11865	7	75	8		0.3766816	0.0843373	0.9286217	0.0740088	0.149041	0.4666667	0.0853659	0.2760163
mus11m	93	447	118	5342	10	36	5		0.4407583	0.1722222	0.9227846	0.1413374	0.2339944	0.6666667	0.2173913	0.442029
mus12m	38	250	107	1105	3	20	4		0.262069	0.1319444	0.8154982	0.0962025	0.0581947	0.4285714	0.1304348	0.2795031
yst01g	3	837	119	8041	0	70	7		0.0245902	0.0035714	0.905722	0.0031283	-0.0277019	0	0	0
yst02g	36	360	72	1532	3	29	2		0.3333333	0.0909091	0.8097252	0.0769231	0.0811402	0.6	0.09375	0.346875
yst03m	43	449	104	3404	5	36	13		0.292517	0.0873984	0.8834674	0.0721477	0.1008138	0.2777778	0.1219512	0.1998645
yst04r	20	640	87	5253	2	53	4		0.1869159	0.030303	0.8913966	0.0267738	0.0331244	0.3333333	0.0363636	0.1848485
yst05r	35	289	37	1139	3	23	1		0.4861111	0.1080247	0.797619	0.0969529	0.1473872	0.75	0.1153846	0.4326923
yst06g	62	526	98	2814	5	43	2		0.3875	0.1054422	0.842515	0.090379	0.1285005	0.7142857	0.1041667	0.4092262
yst07m	0	432	0	2568	0	36	0		NaN	0	0.856	0	NaN	NaN	0	NaN
yst08r	75	1281	204	9440	5	108	9		0.2688172	0.0553097	0.8805149	0.0480769	0.0714191	0.3571429	0.0442478	0.2006953
yst09g	0	1224	215	14561	0	102	13		0	0	0.922458	0	-0.0335899	0	0	0
yst10m	0	588	0	4412	0	49	0		NaN	0	0.8824	0	NaN	NaN	0	NaN
Fly	32	4672	639	37657	3	389	48		0.04769	0.0068027	0.8896265	0.0059891	-0.0248899	0.0588235	0.0076531	0.0332383
Human	819	25101	4300	256780	78	2076	220		0.1599922	0.0315972	0.9109518	0.0271013	NaN	0.261745	0.0362117	0.1489783
Mouse	366	5778	1273	48083	35	476	63		0.2233069	0.0595703	0.8927239	0.0493461	0.0626043	0.3571429	0.0684932	0.212818
Yeast	274	6626	936	53164	23	549	51		0.2264463	0.0397101	0.8891788	0.0349668	0.0509021	0.3108108	0.0402098	0.1755103
Total	1491	42177	7148	395684	139	3490	382		0.1725894	0.034144	0.9036749	0.0293412	NaN	0.2667946	0.0383026	0.1525486

Appendix B: Test Result of Tabu Search

Data set	nTP	nFP	nFN	nTN	sTP	sFP	sFN		nSn	nPPV	nSp	nPC	nCC	sSn	sPPV	sASP
dm01g	1	647	124	5228	0	54	7		0.008	0.0015432	0.8898723	0.0012953	-0.0469956	0	0	0
dm02r	0	288	49	1663	0	24	5		0	0	0.8523834	0	-0.0650001	0	0	0
dm03m	11	445	93	5451	1	37	8		0.1057692	0.0241228	0.9245251	0.0200364	0.0149198	0.1111111	0.0263158	0.0687135
dm04g	0	732	135	7133	0	61	9		0	0	0.9069294	0	-0.0415782	0	0	0
dm05g	11	577	149	6763	2	47	12		0.06875	0.0187075	0.9213896	0.0149254	-0.0053004	0.1428571	0.0408163	0.0918367
dm06r	3	381	95	2521	0	32	7		0.0306122	0.0078125	0.8687112	0.006263	-0.053568	0	0	0
dm07m	0	408	0	4092	0	34	0		NaN	0	0.9093333	0	NaN	NaN	0	NaN
dm08m	0	516	0	5484	0	43	0		NaN	0	0.914	0	NaN	NaN	0	NaN
hm01g	0	2904	236	32860	0	242	16		0	0	0.918801	0	-0.0240627	0	0	0
hm02r	5	667	252	8076	0	56	11		0.0194553	0.0074405	0.9237104	0.0054113	-0.0360125	0	0	0
hm03r	44	1000	364	13592	4	83	11		0.1078431	0.0421456	0.9314693	0.03125	0.0251297	0.2666667	0.045977	0.1563218
hm04m	0	1260	168	24572	0	105	11		0	0	0.9512233	0	-0.0181996	0	0	0
hm05r	30	450	163	2357	2	37	9		0.1554404	0.0625	0.8396865	0.0466563	-0.0032612	0.1818182	0.0512821	0.1165501
hm06g	34	470	41	3955	4	38	5		0.4533333	0.0674603	0.8937853	0.0623853	0.1409085	0.4444444	0.0952381	0.2698413
hm07m	10	482	117	4391	0	41	6		0.0787402	0.0203252	0.9010876	0.0164204	-0.0106556	0	0	0
hm08m	0	636	190	6674	0	53	13		0	0	0.9129959	0	-0.0490747	0	0	0
hm09g	0	1032	160	13808	0	86	10		0	0	0.9304582	0	-0.0282238	0	0	0
hm10m	14	310	75	2601	2	25	9		0.1573034	0.0432099	0.8935074	0.0350877	0.0277751	0.1818182	0.0740741	0.1279461
hm11g	16	800	253	6931	1	67	18		0.0594796	0.0196078	0.8965205	0.0149673	-0.026207	0.0526316	0.0147059	0.0336687
hm12r	4	224	66	706	1	18	4		0.0571429	0.0175439	0.7591398	0.0136054	-0.1117287	0.2	0.0526316	0.1263158
hm13r	18	510	146	5326	2	42	7		0.1097561	0.0340909	0.9126114	0.0267062	0.0128738	0.2222222	0.0454545	0.1338384
hm14r	0	300	82	1618	0	25	4		0	0	0.8435871	0	-0.0868598	0	0	0
hm15r	0	696	90	7214	0	58	4		0	0	0.9120101	0	-0.0329274	0	0	0
hm16g	0	1788	164	19048	0	149	7		0	0	0.914187	0	-0.0270653	0	0	0
hm17g	78	462	67	4893	7	38	3		0.537931	0.1444444	0.9137255	0.1285008	0.2431836	0.7	0.1555556	0.4277778
hm18m	0	888	88	14024	0	74	7		0	0	0.9404506	0	-0.0192702	0	0	0
hm19g	17	415	70	1998	1	35	3		0.1954023	0.0393519	0.8280149	0.0338645	0.0113516	0.25	0.0277778	0.1388889
hm20r	229	4463	1077	64231	24	368	52		0.1753446	0.0488065	0.9350307	0.0396949	0.0597229	0.3157895	0.0612245	0.188507
hm21g	2	562	91	4345	0	47	7		0.0215054	0.0035461	0.8854697	0.0030534	-0.0397295	0	0	0

hm22m	13	371	93	2523	2	30	3		0.1226415	0.0338542	0.8718037	0.0272537	-0.0030696	0.4	0.0625	0.23125
hm23r	8	268	135	1589	1	22	4		0.0559441	0.0289855	0.8556812	0.0194647	-0.0660205	0.2	0.0434783	0.1217391
hm24m	31	365	61	3543	4	29	4		0.3369565	0.0782828	0.9066018	0.0678337	0.1222459	0.5	0.1212121	0.3106061
hm25g	0	168	70	762	0	14	5		0	0	0.8193548	0	-0.1232822	0	0	0
hm26m	25	659	222	8094	3	54	7		0.1012146	0.0365497	0.9247115	0.0275938	0.0159838	0.3	0.0526316	0.1763158
mus01r	13	227	72	1188	2	18	4		0.1529412	0.0541667	0.839576	0.0416667	-0.0047192	0.3333333	0.1	0.2166667
mus02r	52	728	180	8040	5	59	7		0.2241379	0.0666667	0.9169708	0.0541667	0.0794813	0.4166667	0.078125	0.2473958
mus03g	23	289	119	2069	3	23	6		0.1619718	0.0737179	0.8774385	0.0533643	0.027601	0.3333333	0.1153846	0.224359
mus04m	13	395	251	6341	1	33	13		0.0492424	0.0318627	0.9413599	0.0197269	-0.0076416	0.0714286	0.0294118	0.0504202
mus05r	21	195	67	1717	2	16	4		0.2386364	0.0972222	0.8980126	0.0742049	0.0902957	0.3333333	0.1111111	0.2222222
mus06g	7	257	60	1176	1	21	4		0.1044776	0.0265152	0.820656	0.0216049	-0.0406103	0.2	0.0454545	0.1227273
mus07g	2	646	98	5254	0	54	4		0.02	0.0030864	0.8905085	0.002681	-0.0369116	0	0	0
mus08m	2	346	39	4113	0	29	3		0.0487805	0.0057471	0.9224041	0.005168	-0.0102498	0	0	0
mus09r	24	204	17	755	1	17	1		0.5853659	0.1052632	0.7872784	0.0979592	0.1761244	0.5	0.0555556	0.2777778
mus10g	56	760	167	12017	5	63	10		0.2511211	0.0686275	0.9405181	0.0569685	0.1025915	0.3333333	0.0735294	0.2034314
mus11m	83	385	128	5404	9	31	6		0.3933649	0.1773504	0.9334946	0.1392617	0.2245123	0.6	0.225	0.4125
mus12m	38	226	107	1129	3	18	4		0.262069	0.1439394	0.8332103	0.1024259	0.0739335	0.4285714	0.1428571	0.2857143
yst01g	0	780	122	8098	0	65	7		0	0	0.9121424	0	-0.0361105	0	0	0
yst02g	21	327	87	1565	2	27	3		0.1944444	0.0603448	0.827167	0.0482759	0.0128843	0.4	0.0689655	0.2344828
yst03m	7	389	140	3464	1	32	17		0.047619	0.0176768	0.8990397	0.0130597	-0.0336033	0.0555556	0.030303	0.0429293
yst04r	40	536	67	5357	4	44	2		0.3738318	0.0694444	0.9090446	0.0622084	0.1270825	0.6666667	0.0833333	0.375
yst05r	22	254	50	1174	2	21	2		0.3055556	0.0797101	0.8221289	0.0674847	0.0704406	0.5	0.0869565	0.2934783
yst06g	60	504	100	2836	5	42	2		0.375	0.106383	0.8491018	0.0903614	0.1273096	0.7142857	0.106383	0.4103343
yst07m	0	336	0	2664	0	28	0		NaN	0	0.888	0	NaN	NaN	0	NaN
yst08r	59	1129	220	9592	4	95	10		0.2114695	0.0496633	0.8946927	0.0419034	0.0537778	0.2857143	0.040404	0.1630592
yst09g	11	1069	204	14716	1	88	12		0.0511628	0.0101852	0.9322775	0.008567	-0.0075997	0.0769231	0.011236	0.0440795
yst10m	0	528	0	4472	0	44	0		NaN	0	0.8944	0	NaN	NaN	0	NaN
Fly	26	3994	645	38335	3	332	48		0.0387481	0.0064677	0.9056439	0.0055734	-0.0236746	0.0588235	0.0089552	0.0338894
Human	578	22150	4541	259731	58	1836	240		0.1129127	0.0254312	0.9214207	0.0211962	0.0168281	0.1946309	0.030623	0.1126269
Mouse	334	4658	1305	49203	32	382	66		0.2037828	0.0669071	0.9135181	0.0530411	0.0694082	0.3265306	0.0772947	0.2019126
Yeast	220	5852	990	53938	19	486	55		0.1818182	0.0362319	0.9021241	0.0311526	0.0390953	0.2567568	0.0376238	0.1471903
Total	1158	36654	7481	401207	112	3036	409		0.1340433	0.0306252	0.9162885	0.0255669	0.107878	0.2149712	0.0355781	0.1252747

Appendix C: Test Result of Genetic Algorithm

Data set	nTP	nFP	nFN	nTN	sTP	sFP	sFN		nSn	nPPV	nSp	nPC	nCC	sSn	sPPV	sASP
dm01g	16	544	109	5331	2	68	5		0.128	0.0285714	0.9074043	0.0239163	0.0173828	0.2857143	0.0285714	0.1571429
dm03m	8	384	96	5512	1	48	8		0.0769231	0.0204082	0.9348711	0.0163934	0.006229	0.1111111	0.0204082	0.0657596
dm04g	0	680	135	7185	0	85	9		0	0	0.913541	0	-0.0399316	0	0	0
dm05g	27	349	133	6991	4	43	10		0.16875	0.0718085	0.9524523	0.0530452	0.0802533	0.2857143	0.0851064	0.1854103
dm07m	0	264	0	4236	0	33	0		NaN	0	0.9413333	0	NaN	NaN	0	NaN
dm08m	0	400	0	5600	0	50	0		NaN	0	0.9333333	0	NaN	NaN	0	NaN
hm01g	12	2996	224	32768	2	374	14		0.0508475	0.0039894	0.9162286	0.0037129	-0.0096017	0.125	0.0053191	0.0651596
hm02r	24	1008	233	7735	3	126	8		0.0933852	0.0232558	0.8847078	0.0189723	-0.0114516	0.2727273	0.0232558	0.1479915
hm03r	41	1159	367	13433	1	148	14		0.1004902	0.0341667	0.9205729	0.0261646	0.0126293	0.0666667	0.0067114	0.036689
hm04m	16	2248	152	23584	1	282	10		0.0952381	0.0070671	0.9129762	0.0066225	0.0023343	0.0909091	0.0035336	0.0472213
hm05r	20	292	173	2515	2	37	9		0.1036269	0.0641026	0.8959743	0.0412371	-0.0003205	0.1818182	0.0512821	0.1165501
hm06g	7	633	68	3792	1	79	8		0.0933333	0.0109375	0.8569492	0.009887	-0.0182227	0.1111111	0.0125	0.0618056
hm07m	12	516	115	4357	0	66	6		0.0944882	0.0227273	0.8941104	0.0186625	-0.005837	0	0	0
hm08m	2	1062	188	6248	0	133	13		0.0105263	0.0018797	0.8547196	0.0015974	-0.0606874	0	0	0
hm09g	3	1317	157	13523	0	165	10		0.01875	0.0022727	0.9112534	0.0020311	-0.0253819	0	0	0
hm10m	6	410	83	2501	1	51	10		0.0674157	0.0144231	0.8591549	0.012024	-0.036049	0.0909091	0.0192308	0.0550699
hm11g	26	926	243	6805	3	115	16		0.0966543	0.0273109	0.8802225	0.0217573	-0.0128733	0.1578947	0.0254237	0.0916592
hm12r	7	89	63	841	1	11	4		0.1	0.0729167	0.9043011	0.0440252	0.0037252	0.2	0.0833333	0.1416667
hm13r	25	551	139	5285	3	69	6		0.152439	0.0434028	0.905586	0.034965	0.0321162	0.3333333	0.0416667	0.1875
hm14r	17	143	65	1775	2	18	2		0.2073171	0.10625	0.9254432	0.0755556	0.0970354	0.5	0.1	0.3
hm15r	0	576	90	7334	0	72	4		0	0	0.9271808	0	-0.0297115	0	0	0
hm16g	16	1528	148	19308	1	192	6		0.097561	0.0103627	0.9266654	0.0094563	0.0081708	0.1428571	0.0051813	0.0740192
hm17g	73	679	72	4676	8	84	2		0.5034483	0.0970745	0.8732026	0.0885922	0.1756461	0.8	0.0869565	0.4434783
hm18m	1	735	87	14177	0	92	7		0.0113636	0.0013587	0.9507108	0.0012151	-0.0134085	0	0	0
hm19g	24	448	63	1965	3	56	1		0.2758621	0.0508475	0.814339	0.0448598	0.042242	0.75	0.0508475	0.4004237
hm20r	251	5741	1055	62953	27	715	49		0.1921899	0.0418892	0.9164265	0.035618	0.0525319	0.3552632	0.0363881	0.1958256
hm21g	2	622	91	4285	0	78	7		0.0215054	0.0032051	0.8732423	0.0027972	-0.0430279	0	0	0
hm22m	44	436	62	2458	4	53	1		0.4150943	0.0916667	0.8493435	0.0811808	0.1331695	0.8	0.0701754	0.4350877
hm23r	19	221	124	1636	1	29	4		0.1328671	0.0791667	0.8809908	0.0521978	0.0109878	0.2	0.0333333	0.1166667
hm24m	30	506	62	3402	4	62	4		0.326087	0.0559701	0.870522	0.0501672	0.0865173	0.5	0.0606061	0.280303

hm25g	0	88	70	842	0	11	5		0	0	0.9053763	0	-0.0852219	0	0	0
hm26m	70	1010	177	7743	7	125	3		0.2834008	0.0648148	0.884611	0.0556881	0.084468	0.7	0.0530303	0.3765152
mus01r	18	190	67	1225	3	23	3		0.2117647	0.0865385	0.8657244	0.0654545	0.05184	0.5	0.1153846	0.3076923
mus02r	32	920	200	7848	4	115	8		0.137931	0.0336134	0.895073	0.0277778	0.0170059	0.3333333	0.0336134	0.1834734
mus03g	17	279	125	2079	2	34	7		0.1197183	0.0574324	0.8816794	0.04038	0.0010013	0.2222222	0.0555556	0.1388889
mus04m	31	433	233	6303	3	54	11		0.1174242	0.0668103	0.9357185	0.0444763	0.0406942	0.2142857	0.0526316	0.1334586
mus05r	28	244	60	1668	3	30	3		0.3181818	0.1029412	0.8723849	0.0843373	0.1140187	0.5	0.0909091	0.2954545
mus06g	11	189	56	1244	2	23	3		0.1641791	0.055	0.8681089	0.0429688	0.0196207	0.4	0.08	0.24
mus07g	5	579	95	5321	1	72	3		0.05	0.0085616	0.9018644	0.0073638	-0.0207896	0.25	0.0136986	0.1318493
mus08m	4	260	37	4199	1	32	2		0.097561	0.0151515	0.941691	0.013289	0.0158705	0.3333333	0.030303	0.1818182
mus09r	0	104	41	855	0	13	2		0	0	0.8915537	0	-0.0704442	0	0	0
mus10g	53	1283	170	11494	7	160	8		0.2376682	0.0396707	0.8995852	0.0351926	0.0586898	0.4666667	0.0419162	0.2542914
mus11m	52	676	159	5113	7	85	8		0.2464455	0.0714286	0.8832268	0.0586246	0.0731538	0.4666667	0.076087	0.2713768
mus12m	23	209	122	1146	2	27	5		0.1586207	0.0991379	0.8457565	0.0649718	0.0035772	0.2857143	0.0689655	0.1773399
yst01g	2	1110	120	7768	0	139	7		0.0163934	0.0017986	0.8749718	0.0016234	-0.0381742	0	0	0
yst02g	5	259	103	1633	0	33	5		0.0462963	0.0189394	0.8631078	0.013624	-0.0604929	0	0	0
yst03m	20	572	127	3281	4	70	14		0.1360544	0.0337838	0.8515443	0.0278164	-0.0065708	0.2222222	0.0540541	0.1381381
yst04r	11	781	96	5112	1	98	5		0.1028037	0.0138889	0.8674699	0.0123874	-0.0116226	0.1666667	0.010101	0.0883838
yst05r	17	239	55	1189	2	29	2		0.2361111	0.0664062	0.8326331	0.0546624	0.0390603	0.5	0.0645161	0.2822581
yst06g	55	601	105	2739	6	75	1		0.34375	0.0838415	0.8200599	0.0722733	0.0876711	0.8571429	0.0740741	0.4656085
yst07m	0	488	0	2512	0	61	0		NaN	0	0.8373333	0	NaN	NaN	0	NaN
yst08r	11	1509	268	9212	0	190	14		0.0394265	0.0072368	0.8592482	0.0061521	-0.0461649	0	0	0
yst09g	2	798	213	14987	0	100	13		0.0093023	0.0025	0.9494457	0.0019743	-0.0217931	0	0	0
yst10m	0	608	0	4392	0	76	0		NaN	0	0.8784	0	NaN	NaN	0	NaN
Fly	51	2621	473	34855	7	327	32		0.0973282	0.0190868	0.9300619	0.0162162	0.0124928	0.1794872	0.0209581	0.1002226
Human	748	25940	4371	255941	75	3243	223		0.1461223	0.0280276	0.9079754	0.0240832	NaN	0.2516779	0.022604	0.1371409
Mouse	274	5366	1365	48495	35	668	63		0.1671751	0.0485816	0.9003732	0.0391149	0.0378465	0.3571429	0.0497866	0.2034647
Yeast	123	6965	1087	52825	13	871	61		0.1016529	0.0173533	0.8835089	0.0150459	-0.0064563	0.1756757	0.0147059	0.0951908
Total	1196	40892	7296	392116	130	5109	379		0.1408384	0.0284167	0.9055629	0.0242184	0.0670603	0.2554028	0.0248139	0.1401083