8-6-2009

A Combined Motif Discovery Method

Daming Lu
University of New Orleans

A Combined Motif Discovery Method

A Thesis

Submitted to the Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirements for the degree of

Master of Science
in
Computer Science
Bioinformatics

by

Daming Lu

B.E. Dalian University of Technology, 2007

August 2009

Acknowledgment

I would like to express my great gratitude to Dr. Stephen Winters-Hilt. As my advisor, he has helped me tremendously to understand the importance of academic research. He gave me the enthusiasm to begin research in the field of Bioinformatics, which I previously did not know anything about. I aspire to keep this enthusiasm in the future endeavors, as he has after years of research. His intelligence and accomplishments make him an excellent mentor yet he still knows how to balance work and fun.

Table of Contents

List of Illustrations

Abstract

A central problem in the bioinformatics is to find the binding sites for regulatory motifs. This is a challenging problem that leads us to a platform to apply a variety of data mining methods.

In the efforts described here, a combined motif discovery method that uses mutual information and Gibbs sampling was developed. A new scoring schema was introduced with mutual information and joint information content involved. Simulated tempering was embedded into classic Gibbs sampling to avoid local optima.

This method was applied to the 18 pieces DNA sequences containing CRP binding sites validated by Stormo and the results were compared with Bioprospector. Based on the results, the new scoring schema can get over the defect that the basic model PWM only contains single positioin information. Simulated tempering proved to be an adaptive adjustment of the search strategy and showed a much increased resistance to local optima.

Keywords:

Transcription Factor Binding Site

Gibbs Sampling

Mutual Information

Information Content

Simulated Tempering

Scoring Function

Chapter 1 Introduction

Uncovering the hidden mechanism of gene transcription control is a huge effort in the post genomic era. Various methods have been invented to decipher the information encoded in DNA sequences. The approaches come from two ways: the biological experimental way or computational biology way.

Biology experiment is accurate in locating the functional DNA subsequences in the genome sequences, but is time and labour consuming. Conversely, the computational way is high throughput and time saving, but needs a large amount of DNA sequences as prerequisite and is not very accurate.

Motif discovery by computer programs, however, became feasible as the publicly available biosequences databases grow in site and high performance computers become cheaply available. Consequently, many fundamental computational methods to discover functional biosequences have been developed. Those methods include the Gibbs sampling method introduced by Lawrence [1] and EM method used by Elkan [2]. Although these methods achieve some degree of success, and many computer programs have been developed based on them, the problem of motif discovery from DNA sequences still remains difficult because of its complex nature.

In addition, the search strategy differs largely also. Some basic algorithms like consensus [3], EM [4] and Gibbs sampler [5] brought solutions to this problem, but the result was not satisfactory enough. The enhanced computer programs based on them such as MEME [6], AlignAce [7], and Bioprospector [8] are more powerful in dealing with true data, since these programs are enhanced by using more complex models and considering more parameters. After considering the above algorithms, we found a varied Gibbs sampling method similar to Bioprospector with some advantages. A new scoring schema was introduced with further incorporation of a novel mutual information motif finder to strengthen the overall method. Simulated tempering was also embedded into classic Gibbs sampling to avoid local optima.

Chapter 2 Gibbs Sampling

Gibbs sampling is a Markov chain Monte Carlo method for joint distribution estimation when the full conditional distributions of all the relevant random variables are available. The Gibbs sampling procedure iteratively draws samples from the full conditional distributions. The samples collected in this way are guaranteed to converge to the true joint distribution as long as there is no zero-probability in the target joint distribution.

Gibbs sampling strategy has been applied to Bayesian hierarchical models in bioinformatics. The first introduction of the methodology is its application to the motif discovering problem in DNA sequence analysis [5].

This chapter serves as a brief review for the applications of Gibbs sampling in the field of bioinformatics. The working mechanism of Gibbs sampling was discussed and some essential concepts needed for understanding this method was introduced.

2.1 Introduction to Gibbs Sampling

Gibbs sampling is a technique to draw samples from a join distribution based on the full conditional distributions of all the associated random variables. Though the idea goes back to the work of Hasting (1970) [9], whose focus was on its Markov chain Monte Carlo (MCMC) nature, the Gibbs sampler was first formally introduced by Geman and Geman [10] to the field of image processing. The work caught the attention of the statistics society (especially boosted by the thesis of Gelfand and Smith (1992) [11]).

Since then, the applications of Gibbs sampling have covered both the Bayesian world and the world of classical statistics. In the former case, Gibbs sampling is often used to estimate posterior distributions, and in the latter, it is often applied to likelihood estimation [12]. In particular, Gibbs sampling has become a popular alternative to the expectation-maximization (EM) for solving the incomplete-data problem in the Bayesian

context, where the associated random variables of interest include both the hidden variables (i.e., the missing data) and the parameters of the model that describe the complete data.

To provide answers to this type of questions, EM is a numerical maximization procedure that climbs in the likelihood landscape aiming to find the model parameters and the hidden variables that maximize the likelihood function. In contrast, Gibbs sampling provides the means to estimate the target joint distribution of the hidden variables and the model parameters as a whole, and leave the estimation of the random variables for later (i.e. after the samples are drawn), where maximum a posterior (MAP) estimates are often used. Thus, Gibbs sampling suffers less from the problem of local maxima than EM. This property makes Gibbs sampling a suitable candidate for solving the model-based problems in bioinformatics, where the likelihood function usually consists of a large amount of modes due to the high complexity of the data.

In the remainder of this chapter, the applications of Gibbs sampling to the hierarchical Bayesian models were shown that address an important problem in systems biology. The goal is to discover regulation mechanism of genes. A typical framework by means of computational biology for this kind of study is composed of two steps. In the first step groups of genes that share similar expression profiles (which measured by the microarray technology) are found. (These genes are called to be coexpressed). This is done by performing clustering algorithms to the gene expression profiles (i.e., microarray data). The second step is based on the general assumption that coexpression implies coregulation. For each group of genes found in the first step, the DNA sequences that are related to the regulation of these genes are extracted and common patterns of these sequences (called motifs) are seeked. The positions of these conserved motifs are likely to be the binding sites of transcription factors, which are the executors of the gene regulation mechanism. We show in this thesis that the Gibbs sampling strategy can be applied to both the motif finding problem of DNA sequences and other bioinformatics problems such as the clustering of microarray.

We will first review the working mechanism of Gibbs sampling. Then some basic biological concepts for understanding the biological problems of interest are introduced.

2.2 Explanation in Mathematical Terms

2.2.1 Parameters

The first requirement for the Gibbs sampling is the observable data. The observed data will be denoted Y. In the general case of the Gibbs sampling, the observed data remains constant throughout. Gibbs sampling requires a vector of parameters of interest that are initially unknown.

These parameters will be denoted by the vector Φ. Nuisance parameters, Θ, are also initially unknown. The goal of Gibbs sampling is to find estimates for the parameters of interest in order to determine how well the observable data fits the model of interest, and also whether or not data independent of the observed data fits the model described by the observed data. Gibbs sampling requires an initial starting point for the parameters. In our situation, this is set randomly. Then, one at a time, a value for each parameter of interest is sampled given values for the other parameters and data.

Once all of the parameters of interest have been sampled, the nuisance parameters are sampled given the parameters of interest and the observed data. At this point, the process is started over. The power of Gibbs sampling is that the joint distribution of the parameters will converge to the joint probability of the parameters given the observed data.

The Gibbs sampler requires a random starting point of parameters of interest, Φ, and nuisance parameters, Θ, with observed data Y, from which a converging distribution can be found. For the sampler, there is an initial starting point.

$(\Theta_1^{(0)}, \Theta_2^{(0)}, ..., \Theta_D^{(0)}, \Phi^{(0)})$,

Steps a-d are then repeatedly run.

a) Sample $\Theta_1^{(i+1)}$ from $p(\Theta_1 \mid \Theta_2^{(i)}, ..., \Theta_D^{(i)}, \Phi^{(i)}, Y)$

b) Sample $\Theta_2^{(i+1)}$ from $p(\Theta_2 \mid \Theta_1^{(i+1)}, \Theta_3^{(i)}, ..., \Theta_D^{(i)}, \Phi^{(i)}, Y)$

……

……

c) Sample $\Theta_D^{(i+1)}$ from $p(\Theta_D \mid \Theta_1^{(i+1)}, ..., \Theta_{D-1}^{(i+1)}, \Phi^{(i)}, Y)$

d) Sample $\Phi^{(i+1)}$ from $p(\Phi \mid \Theta_1^{(i+1)}, ..., \Theta_D^{(i+1)}, Y)$

2.2.2 ParametersMultiple Alignments

One application of Gibbs sampling useful in computational molecular biology is the detection and alignment of locally conserved regions (motifs) in sequences of amino acids or nucleic acids assuming no prior information in the patterns or motifs. Gibbs sampling strategies claim to be fast and sensitive, avoiding the problem that EM algorithms fall into as far as getting trapped by local optima.

2.3 Algorithm Scheme

First the basic multiple alignment strategy is examined where a single motif is desired. The most basic implementation, known as a site sampler, assumes that there is exactly one motif element located within each sequence.

2.3.1 Notation

- *N:* number of sequences
- $S_1 ... S_n$: set of sequences
- *W*: width of motif to be found in the sequences
- *J*: the number of residues in the alphabet. J = 4 for nucleic acid sequences and 20 for amino acid sequences.

• $c_{i,j,k}$: Observed counts of residue j in position i of motif k . j ranges from 1…J. i ranges from 0..W where $c_{0,j}$ contains the counts of residue j in the background. If it is assumed that only a single motif is searched for, the k term can drop out.

• $q_{i,j}$: frequency of residue j occurring in position i of the motif. i ranges from 0..W as above.

• a_k : vector of starting positions of the motifs within the sequences. k ranges from 1. .N .

• b_j : pseudocounts for each residue – needed according to Bayesian statistical rules to eliminate problems with zero counts.

• B : The total number of pseudocounts. $B = \sum_j b_j$.

2.3.2 Initialization

Once the sequences are known, the counts for each residue can calculated. Initially, $c_{0,j}$ will contain the total counts of residue j within all of the sequences and $c_{i,j}$ is initialized to 0 for all other values of i. This is a summary observed data. The site sampler is then initialized by randomly selecting a position for the motif within each sequence and recording these positions in a_k . The counts are updated according to this initial alignment. After the observed counts are set, $q_{i,j}$ can be calculated.

$$q_{i,j} = \frac{c_{i,j} + b_j}{N - 1 + B}$$

Equation 1: Motif Residue Frequencies

$$q_{0,j} = \frac{c_{o,j} + b_j}{\sum_{k=1}^{j} c_{0,k} + B}$$

Equation 2 : Background Residue Frequencies

2.3.3 Predictive Update Step

The first step, known as the predictive update step, selects one of the sequences and places the motif within that sequence in the background and updates the residue counts. One of the N sequences, z, is chosen. The motif in sequence z is taken from the model and placed in the background. The observed counts $c_{i,j}$ are updated as are the frequencies $q_{i,j}$. The selection of z can be random or in a specified order.

2.3.4 Sampling Step

In the sampling step, a new motif position for the selected sequence is determined by sampling according to a weight distribution. All of the possible segments of width *W.* within sequence *z* are considered. For each of these segments x, a weight A_x is calculated according to the ratio $A_x = \frac{Q_x}{P_x}$ where $Q_x = \prod_{i=1}^{W} q_{i,r_i}$ is the model residue frequency according to equation 1 if segment x is in the motif model, and $P_x = \prod_{i=1}^{W} q_{0,r_i}$ is the background residue frequency according to equation 2. r_i refers to the residue located at position i of segment x . Once A_x is calculated for every possible x, a new position a_z is chosen by randomly sampling over the set of weights A_x. Thus, possible starting positions with higher weights will be more likely to be chosen as the new motif position than those positions with lower weights. Since this is a stochastic process, the starting position with the highest weight is not guaranteed to be chosen. Once the iterative predictive update and sampling steps have been performed for all of the sequences, a probable alignment is present. For this alignment, a maximum posteriori (MAP) estimate can be calculated using equation 3:

$$F = \sum_{i=1}^{W} \sum_{j=1}^{J} c_{i,j} \log \frac{q_{i,j}}{q_{0,j}}$$

Equation 3: Alignment conditional log-likelihood

2.3.5 Explanation

The idea is that the more accurate the predictive update step is, the more accurate the sampling step will be since the background will be more distinguished from the motif description. Given random positions a_k in the sampling step, the pattern description $q_{i,j}$ will not favor any particular segment. Once some correct a_k have been selected by chance, the $q_{i,j}$ begins to favor a particular motif.

```
globalMaxAlignmentProb = 0
For Iteration = 1 to N
  Initialize Random alignment
  localMaxAlignmentProb = 0;
      while (not in local maximum
                and
              innerloop < MAXLOOP)
        do
         for each sequence do{
           Predictive Update
           Sample
         }
       calculate AlignmentProb
       if(AlignmentProb
          >localMaxAlignmentProb){

    localMaxAlignmentProb=AlignmentProb;
             not in local maximum=true;
       }
       Innerloop++;
}
If(localMaxAlignmentProb
  ==globalMaxAlignmentProb)
  exit -> max found twice

else if (localMaxAlignmentProb >
globalMaxAlignmentProb)
 globalMaxAlignmentProb=
                  localMaxAlignmentProb
}
```

Fig 2.1 Gibbs Sampling Algorithm Sketch

Chapter 3 Simulated Tempering

3.1 Simulated Annealing

Simulated annealing is a generalization of a Monte Carlo method for examining the equations of state and frozen states of n-body systems [13]. The concept is based on the manner in which liquids freeze or metals recrystalize in the process of annealing. In an annealing process a melt, initially at high temperature and disordered, is slowly cooled so that the system at any time is approximately in thermodynamic equilibrium. As cooling proceeds, the system becomes more ordered and approaches a "frozen" ground state at T=0. Hence the process can be thought of as an adiabatic approach to the lowest energy state. If the initial temperature of the system is too low or cooling is done insufficiently slowly the system may become quenched forming defects or freezing out in metastable states (ie. trapped in a local minimum energy state).

The original Metropolis scheme was that an initial state of a thermodynamic system was chosen at energy *E* and temperature T, holding T constant the initial configuration is perturbed and the change in energy *dE* is computed. If the change in energy is negative the new configuration is accepted. If the change in energy is positive it is accepted with a probability given by the Boltzmann factor *exp -(dE/T)*. This processes is then repeated sufficient times to give good sampling statistics for the current temperature, and then the temperature is decremented and the entire process repeated until a frozen state is achieved at T=0.

By analogy the generalization of this Monte Carlo approach to combinatorial problems is straightforward [14, 15]. The current state of the thermodynamic system is analogous to the current solution to the combinatorial problem, the energy equation for the thermodynamic system is analogous to at the objective function, and ground state is analogous to the global minimum. The major difficulty (art) in implementation of the algorithm is that there is no obvious analogy for the temperature T with respect to a free parameter in the combinatorial problem. Furthermore, avoidance of entrainment in local

minima (quenching) is dependent on the "annealing schedule", the choice of initial temperature, how many iterations are performed at each temperature, and how much the temperature is decremented at each step as cooling proceeds.

There are certain optimization problems that become unmanageable using combinatorial methods as the number of objects becomes large. A typical example is the traveling salesman problem, which belongs to the NP-complete class of problems. For these problems, there is a very effective practical algorithm called simulated annealing (thus named because it mimics the process undergone by misplaced atoms in a metal when it's heated and then slowly cooled). While this technique is unlikely to find the *optimum* solution, it can often find a very good solution, even in the presence of noisy data.

The traveling salesman problem can be used as an example application of simulated annealing. In this problem, a salesman must visit some large number of cities while minimizing the total mileage traveled. If the salesman starts with a random itinerary, he can then pairwise trade the order of visits to cities, hoping to reduce the mileage with each exchange. The difficulty with this approach is that while it rapidly finds a local minimum, it cannot get from there to the global minimum.

Simulated annealing improves this strategy through the introduction of two tricks. The first is the so-called "Metropolis algorithm" [16], in which some trades that do not lower the mileage are accepted when they serve to allow the solver to "explore" more of the possible space of solutions. Such "bad" trades are allowed using the criterion that

$$e^{-\Delta D/T} > R(0,1)$$

where ΔD is the change of distance implied by the trade (negative for a "good" trade; positive for a "bad" trade), T is a "synthetic temperature," and $R(0,1)$ is a random number in the interval $[0,1]$. D is called a "cost function," and corresponds to the free energy in the case of annealing a metal (in which case the temperature parameter would actually be the kT, where k is Boltzmann's Constant and T is the physical temperature, in the Kelvin

absolute temperature scale). If *T* is large, many "bad" trades are accepted, and a large part of solution space is accessed. Objects to be traded are generally chosen randomly, though more sophisticated techniques can be used.

The second trick is, again by analogy with annealing of a metal, to lower the "temperature." After making many trades and observing that the cost function declines only slowly, one lowers the temperature, and thus limits the size of allowed "bad" trades. After lowering the temperature several times to a low value, one may then "quench" the process by accepting only "good" trades in order to find the local minimum of the cost function. There are various "annealing schedules" for lowering the temperature, but the results are generally not very sensitive to the details.

There is another faster strategy called threshold acceptance [17]. In this strategy, all good trades are accepted, as are any bad trades that raise the cost function by less than a fixed threshold. The threshold is then periodically lowered, just as the temperature is lowered in annealing. This eliminates exponentiation and random number generation in the Boltzmann criterion. As a result, this approach can be faster in computer simulations.

3.2 Simulated Tempering

To alleviate the vulnerability of Gibbs sampling to local optima trapping, we propose to combine a thermodynamic method, called simulated tempering, with Gibbs sampling. The combined method was validated using synthetic data and actual promoter sequences extracted from CRP binding site of *E.Coli*. It is noteworthy that the marked improvement of the efficiency presented here is attributable solely to the improvement of the search method.

Simulated tempering is an accelerated version of simulated annealing and has two main features. First, the temperature of the system is continuously adjusted during the optimization process and may be increased as well as decreased. Second, the adjustment of temperature is performed without detailed analysis of the potential landscape. Temperature control is performed by introducing a second Markov chain.

In this section, we demonstrate that simulated tempering (ST) [18], which is one of many proposals from the field of thermodynamics for the systematic avoidance of local optima in multivariate optimization problems, is quite useful for reducing the vulnerability of Gibbs sampling to local optima. The application of ST to a genetics problem has already been reported [19]. SA and potential deformation [20,21], which has already succeeded in other problems of bioinformatics, are also rooted in the field of thermodynamics. ST and SA employ a temperature parameter T, the introduction of which into a local alignment problem has already been reported [22].

The novelty of ST is that it attempts to adjust the value of adaptively to the current score of alignments. By changing T, ST adopts continuously changing search methods ranging from a fast deterministic-like search to a random-like search, reducing the possibility of being trapped in local optima. This principal is schematically shown in Fig. 1. In the present work, we implemented and tested an ST-enhanced Gibbs sampling algorithm for TFBS discovery, which we call GibbsST. The validation of our algorithm is also presented on synthetic test data and promoter sequences of Saccharomyces cerevisiae.

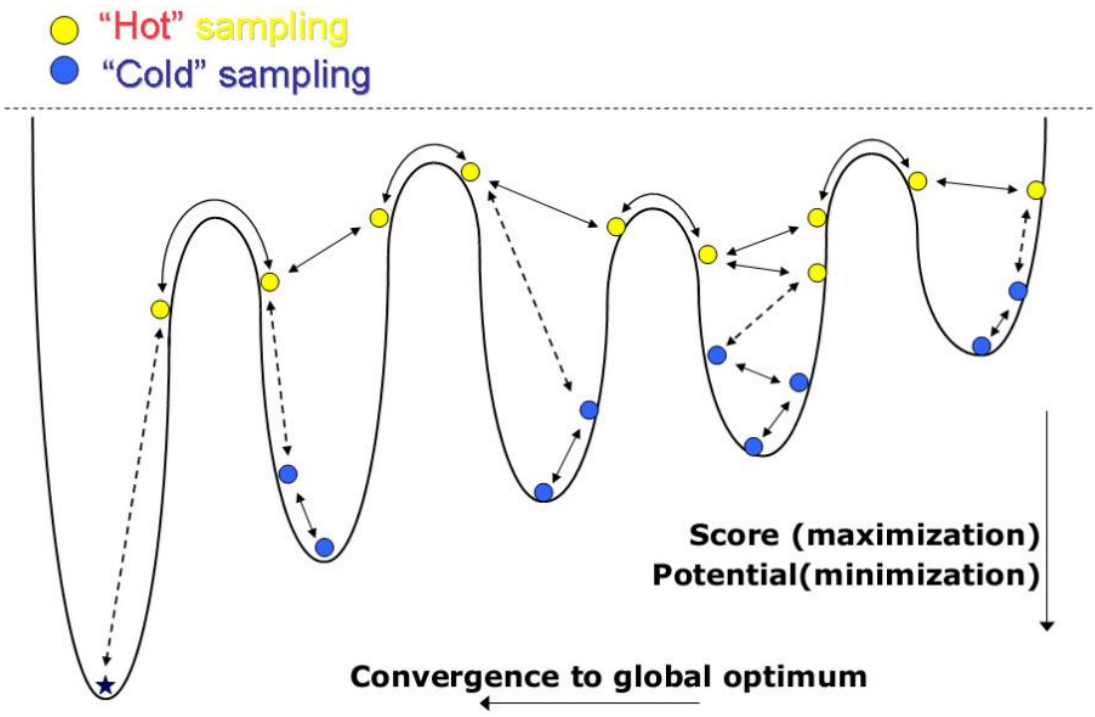


Fig 3.1 Avoid local optimum via simulated tempering

3.3 Gibbs Sampling with Simulated Tempering

3.3.1 Gibbs sampling with temperature

In this section, we introduce a temperature, *T*, into the "classic" Gibbs sampling algorithm proposed by Lawrence et al. The details of the algorithm (row selection order, pseudocount, etc.) will be introduced later along with the implementation of our algorithm. For simplicity, it is assumed that all *N* of input sequences have exactly one occurrence (the OOPS-model) of the pattern, which is always W_m bp long, and negative strands are not considered.

The algorithm holds a current local alignment, *A*, and a current PWM (Position Weight Matrix), $q_{i,j}$, which are iteratively updated as a Markov chain until the convergence to a pattern. The alignment *A* is represented by the starting points of aligned segments, x_k, which form a gapless sequence block. The first half of an iterative step is the recalculation of elements of the current PWM according to the current alignment, excluding the k-th row. Then in the second half of a step, the k-th row of the current alignment is updated by sampling a new value of x_k according to weights derived from $q_{i,j}$. Let *l*(1), *l*(2), ... denote the entire sequence of the row to be updated. We set the probability of the new starting point being x proportional to $(\frac{Q_x}{P_x})^{\beta}$, $\beta = 1/T$

where $Q_x = \prod_{i=0}^{W_m-1} q_{l(x+i),i}$ is the likelihood that the x-th substring

(x ~ x - 1 + W_m-th letters) of the k-th input sequence comes from the probabilistic model represented by the current PWM, and $P_x = \prod_{i=0}^{W_m-1} p_{l(x+i)}$ is the likelihood that the same subsequence comes from a totally random sequence of the base composition observed for the entire input, $p_{0,1,2,3}$ (that is, $p_{G,A,C,T}$). The *T* is a positive value which is the "temperature" of the system. Note that the computational complexity of the single step of the optimization is not changed by introducing the temperature.

It is easy to see that the above introduced iteration step maximizes $\prod_{i=0}^{W_m-1}(q_{l(x+i),i} / p_{l(x+i)})^{\beta}$ unless T is extremely large. Since k circulates all N of input sequences, this is a maximization of $\beta\sum\sum q_{i,j}\log(q_{i,j} / p_i)$ after all. Hence, the Gibbs sampling introduced here has the relative entropy of the pattern PWM against the background model as its objective-function (or score) to be maximized, and so does our algorithm. Following the convention of statistical physics, however, we refer to TFBS discovery as a minimization of the potential U, which is currently (negative relative entropy). Because we are not proposing a new definition of U, we do not evaluate the sensitivity and specificity of our new algorithm. In principle, the sensitivity and specificity must be independent from the search method in the limit of large step number.

When $T = ß = 1$, the method is reduced to the classic Gibbs sampling without the idea of temperature. In this case, there always is a finite probability of selection of non-optimal x, which gives rise to the escape from the local minima. However, the magnitude of the escape probability may not be sufficient for deep local minima, because the probability is ultimately limited by the pseudocount. The temperature strongly affects the behavior of the optimization algorithm. It is easy to see that when T is large enough, the x selection is almost random ($T \rightarrow \infty$ means that the probabilities of all x are 1), and the algorithm is very inefficient despite the high immunity to the local minima problem. When $T \rightarrow 0$, on the other hand, a very quick convergence to local minima only results, because the movement in the solution space is a "steepest-descent" movement. In simulated annealing, the temperature is initially set to an ideally large value, T_h, where essentially no barrier exists in the potential landscape, and then slowly lowered. There is a theoretical guarantee that SA converges to the global minimum when the temperature decreases slowly enough [23]. However, it is frequently unrealistic to follow the theory because of the large number of iterations required for annealing.

3.3.2 Temperature scheduling

Simulated tempering is an accelerated version of simulated annealing and has two main features. First, the temperature of the system is continuously adjusted during the optimization process and may be increased as well as decreased. Second, the adjustment of temperature is performed without detailed analysis of the potential landscape. Temperature control is performed by introducing a second Markov chain (i.e. a random walk along the temperature axis) that is coupled with U.

In simulated tempering, the temperature of the system takes one of the N_T temperature levels, $T_0 < T_1 < T_2 ... < T_{N_T-1}$ (usually, it is required that $T_{N_T-1} \sim T_h$).During the optimization, the temperature is updated accordingly to the transition rates, *R*, given by a Metropolis-Hastings-like formula:

$$R(T_i \to T_{i+1}) \propto 1/(1+S_+)$$
$$R(T_i \to T_{i-1}) \propto S_- /(1+S_-)$$

where $S_\pm$ is given by

$$\frac{\frac{\exp(-\frac{U}{T_i})}{Z_i}}{\frac{\exp(-\frac{U}{T_{i\pm1}})}{Z_{i\pm1}}}$$

Z_i is a normalizing factor usually called the partition function of the system, defined as

$$Z_i = \sum \exp(-\frac{U}{T_i}).$$

How should the temperature levels be decided in ST? Unlike the case of simulated annealing, no conclusive theory or rule is known for the decision of algorithmic

parameters of simulated tempering, except for the requirement of small temperature intervals. According to the equations above, the equilibrium distributions of U defined for neighboring values of T_i must be overlapped to ensure finite transition rates between these temperature levels. This mainly requires small temperature intervals.

The temperature levels must be decided empirically, which leaves us a vast combination of T_i to explore. However, considering the success of classic Gibbs sampling (and our preliminary test, whose data are not shown), we can safely assume that $T_h \sim 1$ for the current problem.

Moreover, a good starting point has already been pointed out by Frith et al. [7]. In their thesis, they introduced temperature in a manner similar to ours, and reported that a slight improvement of performance was observed only when they fixed the temperature to slightly lower than 1.
So, in this thesis, we chose the result from their work.

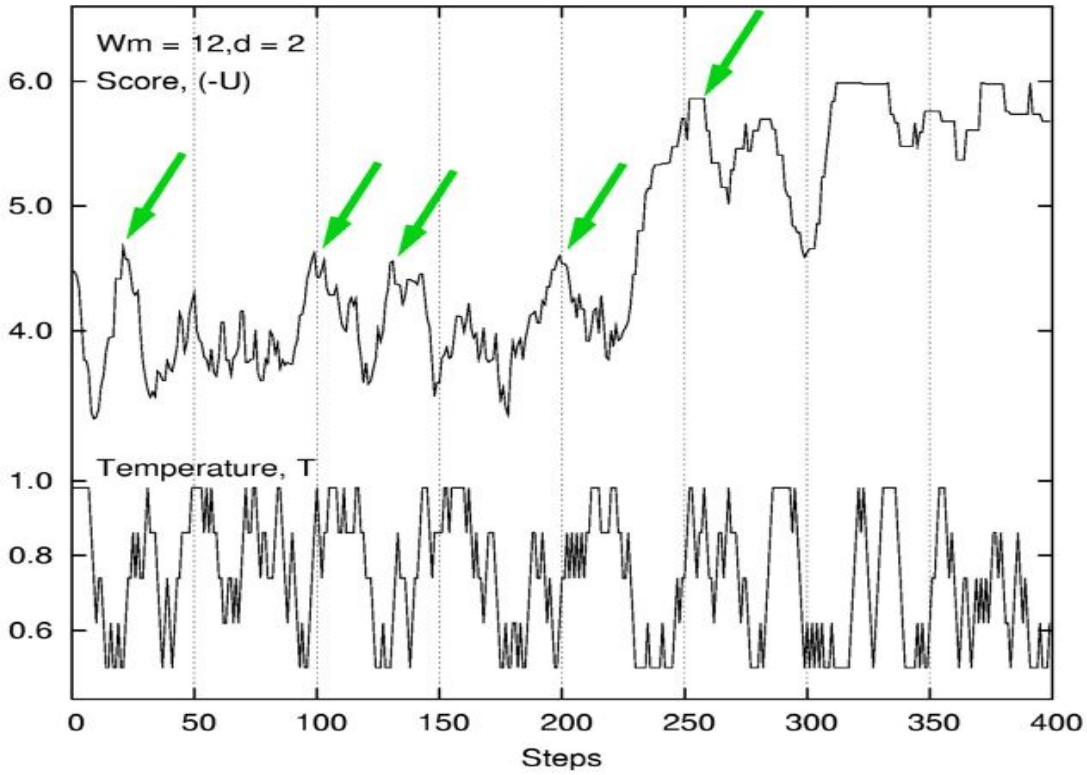


Fig 3.2 TLC 5 = 0.50, 0.62, 0.74, 0.86, 0.98

Chapter 4 Mutual Information & Joint Information

4.1 Mutual Information

The concept of entropy is very important in information theory. It is characterized by the quantity of a random process' uncertainty. If the entropy of the source is less than the capacity of the channel, then asymptotically error free communication can be achieved. The entropy of a discrete random variable X with a frequency p(x) is defined by:

$$H(X) = -\sum_{x} p(x) \log_2 p(x)$$

The joint entropy of two discrete random variables X and Y with frequency p(x) and p(y), respectively, is defined by:

$$H(X,Y) = -\sum_{x,y} p(x,y) \log_2 p(x,y)$$

Conditional entropy H (X|Y) is the entropy of a random variable X, given another random variable Y, which is def ined by:

$$H(X \mid Y) = -\sum_{x,y} p(x,y) \log_2 p(x \mid y)$$

The relative entropy *D*(p||q) is a measure of the distance between two distributions. The relative entropy (or Kullback Leibler distance) between two frequency p(x) and q(x) is defined as

$$D(p \,\|\, q) = \sum p(x) \log_2 \frac{p(x)}{q(x)}$$

The relative entropy is always non-negative and is zero if and only if p = q. However, it is not a true distance between distributions since it is not symmetric and does not satisfythe triangle inequality.

The reduction in uncertainty X due to the knowledge of random variable Y is called the mutual information. For two random variables X and Y, this reduction is:

$$I(X;Y) = \sum_{x,y} p(x,y) \log_2 \frac{p(x,y)}{p(x)p(y)}$$

Where p(x, y) is the joint frequency, p(x) and p(y) are marginal frequency of x and y, respectively, and I(X; Y) is a measure of the dependence between the two random variables. It is symmetric in X and Y and is always non-negative.

Therefore, a recursive style mutual information concept was proposed. The main purpose is to capture more information given more joint frequency. Thus, for a third random variable Z, the accumulative mutual information is defined as:

$$I(X,Y;Z) = \sum_{x,y,z} p(x,y,z) \log_2 \frac{p(x,y,z)}{p(x,y)p(z)}$$

The meaning of accumulative mutual information is that given a single random varable and joint frequency of a group of random varables, it can calculate the intense of linkage between them.

4.2 Scoring Schema

As mentioned in previous section, one of the important problems in motif discovery area is finding the known TFBSs in a given DNA sequence or promoter region (known motif prediction). In this section we focus on this problem and at first, some definitions and notations further used in this thesis are introduced.

Let $N = \{A, C, G, T\}$ be the four nucleotide letters' of which DNA sequences are composed. We have the DNA sequence $D = d_1, ..., d_n$ (a promoter region) on N, and let us suppose that we have t known TFBSs of the length l which are represented by a matrix $B_{t \times l}$ for a given TF, and we intend to investigate by B, where D possesses a motif instance or transcription factor binding site corresponding to the given TF. For finding the position of this motif instance in D, we first create a position weight matrix W of B, and then we scan all subsequences $R = d_i, ..., d_{i+l-1}$ for $i = 1, ..., n-l+1$ of D, and align position weight matrix W with each R. All the subsequences which score is greater than a cutoff are reported as motif instances. The creation of position weight matrix W from TFBSs and calculating the score of alignment W with a subsequence are called scoring schema.

The accuracy of the solution in this search problem depends on how we design the scoring schema, and how the position weight matrix is constructed. In this section we first discuss two existing scoring schemas which are employed for ranking known motifs and predicting TFBSs, later a new scoring schema is presented.

4.2.1 Independent scoring schema

The first scoring schema is a conventional method and is employed in many theses. In this scoring schema, it is assumed that all positions in a given motif are completely independent. This scoring schema is defined as follows.

Suppose we have a promoter region D and a TFBS matrix B of some known motifs. Assume that $F(b, j)$ ($b \in N$ and $1 \le j \le l$) shows the occurrences of nucleotide b in column j of the matrix B. Employing this function, a frequency P is made as follows:

$$P(b, j) = \frac{F(b, j)}{t} + a(b) \quad b \in N \ \wedge \ 1 \le j \le l,$$

where $a(b)$ is the smoothing parameter ($a(b) = 0.01$). Later, a position weight matrix $W_{4\times l}$ is made as follows:

$$W_{b,j} = \log \frac{P(b, j)}{p(b)} \quad b \in N \ \wedge \ 1 \le j \le l,$$

where each $p(b)$ shows the occurrence frequency of nucleotide b (independent of nucleotides in the other position) in a random sequence (obviously $p(b) = 0.25$ for every $b \in N$).

Now, let R be a DNA subsequence with the length l of a promoter region D ($R = r_1,...,r_l$ and $r_i \in N$ for $1 \le i \le l$). For computing the score of R, we align position weight matrix W with R and calculate $Score_1(R)$ as follows:

$$Score_1(R) = \sum_{i=1}^{l} W_{r_i,i}$$

This score can be normalized as follows:

$$NScore_1(R) = \frac{Score_1(R) - MinScore_1}{MaxScore_1 - MinScore_1},$$

where $MaxScore_1$ and $MinScore_1$ are calculated as follows:

$$MaxScore_1 = \sum_{j=1}^{l} \max_{b\in N}\{W_{b,j}\}, \text{ and } MinScore_1 = \sum_{j=1}^{l} \min_{b\in N}\{W_{b,j}\}.$$

4.2.2 Dependent scoring schema

The second scoring schema was first introduced in [24]. In this scoring schema, dependency between some positions in a given TFBS is assumed. This method uses a statistical approach to find dependent positions in a set of known TFBSs. Therefore, if the dependent positions of a set of TFBSs are available, then this scoring schema is defined as follows.

Similar to the previous definition, we have a promoter region D and t binding sites of the length l which are represented by a matrix $B_{t\times l}$ for a given TF. Also, assume that $F([b_1,...,b_m],[j_1,...,j_m])$ shows the occurrences of bases $b_1,...,b_m$ ($b_i \in N$ $for 1\le i\le m$) in dependent positions $j_1,...,j_m$ in the matrix B (positions $j_1,...,j_m$ are determined by statistical approaches [24]). As an example, $F([A,C,A,T],[3,4,8,11])$ represents the number of occurrences of A, C, A, and T in the positions 3, 4, 8, and 11 in a given matrix B. It should be noted that the positions $j_1,...,j_m$ are dependent and not necessarily consecutive.

The corrected frequency for the bases $b_1,...,b_m$ in positions $j_1,...,j_m$ is defined as:

$$P([b_1,...,b_m],[j_1,...,j_m])=\frac{F([b_1,...,b_m],[j_1,...,j_m])}{t}+a(b_1,...,b_m),$$

where $a(b_1,...,b_m)$ is a smoothing parameter and can be calculated as follows:

$$a(b_1,...,b_m)=a(b_1)\times...\times a(b_m).$$

Now, the position weight matrix W corresponding to the binding sites is calculated as:

$$W_{[b_1,...,b_m],[j_1,...,j_m]}=\log_2\frac{P([b_1,...,b_m],[j_1,...,j_m])}{p(b_1)\times...\,p(b_m)}$$

Finally, for a given subsequence $R = r_1,...,r_l$ ($r_i \in N$ and $1 \le i \le l$) of D, we align position weight matrix W with R and calculate $Score_2(R)$ as follows:

$$Score_2(R) = \sum_{i=1}^{k_1} W_{[r_{j_i}],[j_i]} + \sum_{i=1}^{k_2} W_{[r_{j_i}, r_{j_{i+1}}],[j_i, j_{i+1}]} + ... + \sum_{i=1}^{k_m} W_{[r_{j_i},...,r_{j_{i+m-1}}],[j_i, ..., j_{i+m-1}]}$$

where k_1 is the number of independent positions, k_2 is the number of dependent positions order 2 (nucleotides at positions j_i and j_{i+1}) and k_m the number of dependent positions order m (nucleotides at positions $j_i, j_{i+1},..., j_{i+m-1}$).

The normalized version of $Score_2(R)$ can be defined as:

$$NScore_2(R) = \frac{Score_2(R) - MinScore_2}{MaxScore_2 - MinScore_2},$$

where $MaxScore_2$ and $MinScore_2$ can be calculated as follows:

$$MaxScore_2 = \sum_{i=1}^{k_1} \max_{b \in N} W_{b,j_i} + \sum_{i=1}^{k_2} \max_{[b_1,b_2] \in (N \times N)} W_{[b_1,b_2],[j_i, j_{i+1}]} + ... + \sum_{i=1}^{k_m} \max_{[b_1,...,b_m] \in (N \times ... \times N)} W_{[b_1,...,b_m],[j_{i,...,} j_{i+m}]}$$

and

$$MinScore_2 = \sum_{i=1}^{k_1} \min_{b \in N} W_{b,j_i} + \sum_{i=1}^{k_2} \min_{[b_1,b_2] \in (N \times N)} W_{[b_1,b_2],[j_i, j_{i+1}]} + ... + \sum_{i=1}^{k_m} \min_{[b_1,...,b_m] \in (N \times ... \times N)} W_{[b_1,...,b_m],[j_{i,...,} j_{i+m}]}$$

4.2.3 New scoring schema

In the previous subsections we presented two scoring schemas. In the first, nucleotides in all positions in a given TFBS are considered as independent, but this may not be true in all cases because it is shown that dependency between some positions are important [25,26]. In the second, dependency between some positions in a TFBS are considered, but this model has also two problems: first, calculation of dependency between positions is sophisticated, and second, final score is obtained by summation of all the scorings obtained by each order dependent positions, which are not in the same range.

As mentioned, all positions in TFBSs may be dependent, because the length of TFBSs are short, therefore all positions in TFBS may be involved in the interaction with a factor and dependency between all positions are important. TFBSs are short regions in promoter region that TFs can be bonded to them to provide initial conditions for gene transcription. By mutual comparison of TFBS corresponding to a specific TF, we see that some positions in TFBS are mutated and some other ones are conserved.

Since the length of a TFBS is short, therefore it seems that both mutated and conserved positions play an important role in binding of TF and TFBS. During a transcription process, TFBS region constructs structure by hydrogen bonds and this causes the attraction of TF to this region. Thus, with respect to the above feature of this process, it seems that the conserved positions and mutated positions cause this attraction. Also, with respect to that, the average specific free energy of binding to all binding sites play an important role in this attraction, and by considering that this energy is directly related to the information content of the preferred binding sites [26], we use the information content for TFBS scoring. We also illustrate the original motif discovering via mutual information in Appendix A.4.

Similar to the previous subsection, suppose that we have a promoter region D and binding site matrix $B_{t\times l}$ for a given TF. Employing information theory, we compute the information content (IC) of a set of TFBSs which are represented by the matrix B with position independency as follows:

$$IC = \sum_{j=1}^{l} \sum_{b\in N} \frac{F(b, j)}{t} \log \frac{F(b, j)}{t \times p(b)},$$

where F and p are computed similar to independent scoring schema. From this formula, we have $0 \le IC \le 2l$. Now, we assume that positions are mutually dependent, and $F([b_1, b_2], [j_1, j_2])$ shows the number of the occurrence of nucleotides b_1 and b_2 in positions j_1 and j_2 in the given matrix B. As an example, $P([A, T], [3, 8])$ represents the frequency of the occurrence of the pair A and T in the positions 3 and 8 in a given matrix B. Clearly, the number of all two combinations of four nucleotides is equal to 16, and the number of all two combinations of l tuples is equal to $l(l-1)/2$. In this case, the joint information content (JIC) is computed as:

$$JIC = \sum_{j=1}^{l-1} \sum_{k=j+1}^{l} \sum_{b_1\in N} \sum_{b_2\in N} \frac{F([b_1, b_2], [j, k])}{t} \log \frac{F([b_1, b_2], [j, k])}{t \times p(b_1) \times p(b_2)},$$

and for this formula we have $0 \le JIC \le 4l$.

Obviously, we get more information from JIC when the positions are more conserved. Now, the problem is to add up the information of the mutated positions to JIC which have not been considered yet. For this reason, we compute the mutual information (MI) as follows:

$$MI = \sum_{j=1}^{l-1} \sum_{k=j+1}^{l} \sum_{b_1\in N} \sum_{b_2\in N} \frac{F([b_1, b_2], [j, k])}{t} \log \frac{F([b_1, b_2], [j, k])}{t \times F(b_1, j) \times F(b_2, k)}$$

and from this formula we have $0 \le MI \le 2l$. The relation of MI and JIC for each position pairs is as follows. If MI = 0 then JIC = 4 and consequently MI + JIC = 4, if MI = 2 then JIC = 2 and consequently MI + JIC = 4. This condition implies that JIC does show less information and by adding up MI we can get more information. Actually MI carries meaningful information that can not be discarded. On the other hand, IC = 2 means, conservation is low but dependency between positions is high.

With regard to the above discussion, the frequency of the bases b_1 and b_2 in positions j_1 and j_2 can be defined as:

$$P([b_1, b_2],[j_1, j_2]) = \frac{F([b_1, b_2],[j_1, j_2])}{t} + a(b_1, b_2) \ ,$$

where $a(b_1, b_2)$ is a smoothing parameter and can be calculated as:

$$a(b_1, b_2) = a(b_1) \times a(b_2) \ ,$$

Now, for our scoring schema, we make a position weight matrix $W_{16\times((l\times(l-1))/2)}$ whose each entry shows the number of occurrences of a pair of nucleotides in a pair of positions. This matrix is defined as:

$$W_{[b_1,b_2],[j_1,j_2]} = \log \frac{P([b_1, b_2],[j_1, j_2])}{p(b_1) \times p(b_2)} + \log \frac{P([b_1, b_2],[j_1, j_2])}{p(b_1, j_1) \times p(b_2, j_2)} \ ,$$

where $[b_1, b_2] \in (N \times N)$, $1 \le j_1, j_2 \le l$ and $j_1 \ne j_2$.

Finally, for a given subsequence $R = r_1, ..., r_l \ (r_i \in N \ and \ 1 \le i \le l)$ of D, we align position weight matrix W with R and evaluate $Score_3(R)$ as follows:

$$Score_3(R) = \sum_{j_1=1}^{l-1} \sum_{j_2=j_1+1}^{l} W_{[r_{j_1}, r_{j_2}],[j_1, j_2]} .$$

The normalized version of $Score_3(R)$ can be defined as:

$$NScore_3(R) = \frac{Score_3(R) - MinScore_3}{MaxScore_3 - MinScore_3} ,$$

where $MaxScore_3$ and $MinScore_3$ are formulated as follows:

$$MaxScore_3 = \sum_{j_1=1}^{l} \sum_{j_2=j_1+1}^{l-1} \max_{[b_1,b_2] \in (N \times N)} \{W_{[b_1,b_2],[j_1,j_2]}\} ,$$

and

$$MinScore_3 = \sum_{j_1=1}^{l} \sum_{j_2=j_1+1}^{l-1} \min_{[b_1,b_2] \in (N \times N)} \{W_{[b_1,b_2],[j_1,j_2]}\} .$$

4.3 Relative Entropy

And Relative entropy is applied as the current score of the alignments when simulated tempering attempts to adjust the temperatur *T* adaptively.

$$RL = \sum_i p_i \log(\frac{p_i}{q_i})$$

Relative entropy is a non-symmetric measure of the difference between two frequency distributions *P* and *Q*. Relative entropy measures the expected number of extra bits required to code samples from *P* when using a code based on *Q*, rather than using a code based on *P*.

Typically *P* represents the "true" distribution of data, observations, or a precise calculated theoretical distribution. The measure *Q* typically represents a theory, model, description, or approximation of *P*.

In our method, *P* represents the current alignment matrix whereas *Q* represents the background model. Relative entropy is also called the Kullback-Leibler distance, meaning how different the current alignment matrix is from the background matrix. If $p_i = q_i$, RL=0, meaning there is no difference. In our case, we are search the high relative entropy, which means the current alignment matrix is quite different from the background, suggesting a common motif is captured in most sequences.

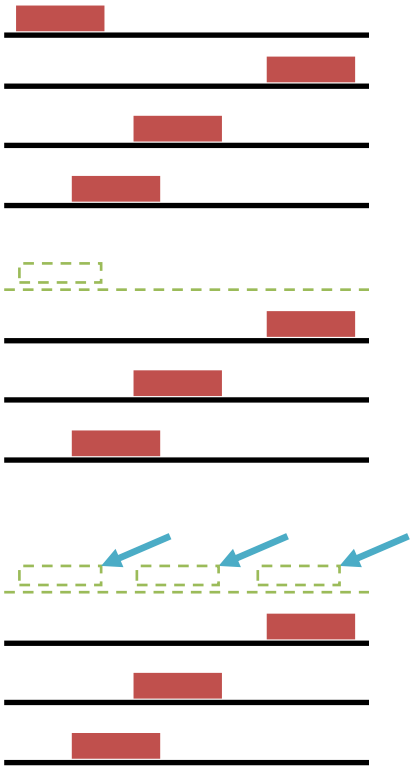
Chapter 5 Method and Result

In this chapter, a combind motif discovery method was described in detail and its result compared with another motif finding method --- Bioprospector, follows.

5.1 Method Sketch

The novelty of Simulated Tempering is that it attempts to adjust the value of *T* adaptively to the current score of alignments. The multivariate 4-nomial distribution matrix $W_{N\times l}$ was then constructed. The trick is to try to match the relative entropy of the current $W_{N\times l}$ to different temperature levels. If the current status is stable, suggesting a common motif is captured in most sequences, and then the relative entropy of this current alignment matrix will be high. Based on this, we tune the temperature low for a quick convergence. If the current status is unstable, suggesting no difference between current matrix and matrix generated from background, then the relative entropy of this current alignment matrix will be low. Based on this, we tune the temperature high for an almost-random search for next step.

By changing *T*, Simulated Tempering adopts continuously changing search methods ranging from a fast deterministic-like search to a random-like search, reducing the possibility of being trapped in local optima. A brief flowchart about the mechanism explained above is shown as below:



Random initialization, $T_0 = 0.98$

Construct background matrix, $W_{4\times l}$ and $W_{16\times(l(l-1)/2)}$ from n-1 sequences

Sampling Step using scroing function:

$$Score_3(R) = \sum_{j_1=1}^{l-1} \sum_{j_2=j_1+1}^{l} W_{[r_{j_1}, r_{j_2}],[j_1, j_2]}$$

Choose motif position by probability:

$$\frac{\exp(Score(i)/T_i)}{\sum_i \exp(Score(j)/T_i)}$$

$Score(i)$ is chosen by the new scoring schema

Adjust temperature T_i :

$$R(T_i \to T_{i+1}) \propto 1/(1+S_+)$$
$$R(T_i \to T_{i-1}) \propto S_-/(1+S_-)$$

$$S_+ = \frac{e^{-U/T_i}/Z_i}{e^{-U/T_{i+1}}/Z_{i+1}} \; ; \; S_- = \frac{e^{-U/T_i}/Z_i}{e^{-U/T_{i-1}}/Z_{i-1}}$$

where

$$Z_i = \sum e^{-U/T_i}$$

Fig 5.1 Method flowchart

5.2 Testing Data

According to the above steps, a motif discovery program was developed. The test data used was a set of DNA sequences comprising CRP binding site. CRP is a protein of E.coli; it takes an important role in metabolism by combining to special DNA sequences and forming DNA-protein complex which regulates some gene transcription. Stormo has collected 18 pieces of DNA sequence; all of them have the ability to combine to CRP. The location of the binding site in each DNA sequences was validated by experiments (Stormo and Hartzell, 1989).

The consensus sequence is TGTGAnnnnnnTCACA; the length is 16. In order to simulate the true situation that some sequences have no motif instance, we have added two computer generated sequences according to a background base distribution. Altogether there are 20 sequences to form the data set, and each sequence is at the length of 105bp. Then we used these data serving as input data to perform the discovery.

```
>cole1
TAATGTTTGTGCTGGTTTTTGTGGCATCGGGCGAGAATAGCGCGTGGTGTGAAAGACTGTTTTTTTGATCGTTTTCACAAAAATGGAAGTCCACAGTCTTGACAG
>ecoarabop
GACAAAAACGCGTAACAAAAGTGTCTATAATCACGGCAGAAAAGTCCACATTGATTATTTGCACGGCGTCACACTTTGCTATGCCATAGCATTTTTATCCATAAG
>ecobgirl
ACAAATCCCAATAACTTAATTATTGGGATTTGTTATATATAACTTTATAAATTCCTAAAATTACACAAAGTTAATAACTGTGAGCATGGTCATATTTTTATCAAT
>ecocrp
CACAAAGCGAAAGCTATGCTAAAACAGTCAGGATGCTACAGTAATACATTGATGTACTGCATGTATGCAAAGGACGTCACATTACCGTGCAGTACAGTTGATAGC
>ecocya
ACGGTGCTACACTTGTATGTAGCGCATCTTTCTTTACGGTCAATCAGCAAGGTGTTAAATTGATCACGTTTTAGACCATTTTTTCGTCGTGAAACTAAAAAAACC
>ecodaop
AGTGAATTATTTGAACCAGATCGCATTACAGTGATGCAAACTTGTAAGTAGATTTCCTTAATTGTGATGTGTATCGAAGTGTGTTGCGGAGTAGATGTTAGAATA
>ecogale
GCGCATAAAAAACGGCTAAATTCTTGTGTAAACGATTCCACTAATTTATTCCATGTCACACTTTTCGCATCTTTGTTATGCTATGGTTATTTCATACCATAAGCC
>ecoilvbpr
GCTCCGGCGGGGTTTTTTGTTATCTGCAATTCAGTACAAAACGTGATCAACCCCTCAATTTTCCCTTTGCTGAAAAATTTTCCATTGTCTCCCCTGTAAAGCTGT
>ecolac
AACGCAATTAATGTGAGTTAGCTCACTCATTAGGCACCCCAGGCTTTACACTTTATGCTTCCGGCTCGTATGTTGTGTGGAATTGTGAGCGGATAACAATTTCAC
>ecomale
ACATTACCGCCAATTCTGTAACAGAGATCACACAAAGCGACGGTGGGGCGTAGGGGCAAGGAGGATGGAAAGAGGTTGCCGTATAAAGAAACTAGAGTCCGTTTA
>ecomalk
GGAGGAGGCGGGAGGATGAGAACACGGCTTCTGTGAACTAAACCGAGGTCATGTAAGGAATTTCGTGATGTTGCTTGCAAAAATCGTGGCGATTTTATGTGCGCA
>ecomalt
GATCAGCGTCGTTTTAGGTGAGTTGTTAATAAAGATTTGGAATTGTGACACAGTGCAAATTCAGACACATAAAAAAACGTCATCGCTTGCATTAGAAAGGTTTCT
>ecoompa
GCTGACAAAAAAAGATTAAACATACCTTATACAAGACTTTTTTTTCATATGCCTGACGGAGTTCACACTTGTAAGTTTTCAACTACGTTGTAGACTTTACATCGCC
>ecotnaa
TTTTTTAAACATTAAAATTCTTACGTAATTTATAATCTTTAAAAAAAGCATTTAATATTGCTCCCCGAACGATTGTGATTCGATTCACATTTAAACAATTTCAGA
>ecouxu1
CCCATGAGAGTGAAATTGTTGTGATGTGGTTAACCCAATTAGAATTCGGGATTGACATGTCTTACCAAAAGGTAGAACTTATACGCCATCTCATCCGATGCAAGC
>pbr322
CTGGCTTAACTATGCGGCATCAGAGCAGATTGTACTGAGAGTGCACCATATGCGGTGTGAAATACCGCACAGATGCGTAAGGAGAAAATACCGCATCAGGCGCTC
>trn9cat
CTGTGACGGAAGATCACTTCGCAGAATAAATAAATCCTGGTGTCCCTGTTGATACCGGGAAGCCCTGGGCCAACTTTTGGCGAAAATGAGACGTTGATCGGCACG
>(tdr)
GATTTTTATACTTTAACTTGTTGATATTTAAAGGTATTTAATTGTAATAACGATACTCTGGAAAGTATTGAAAGTTAATTTGTGAGTGGTCGCACATATCCTGTT
```

Fig 5.2 Test data in FASTA format

5.3 Result

Our combined method, as well as Bioprospector, was run on the same testing data. Results are shown below:

	A	B	C	D	E	F	G
1	Gene Names	Motifs copies	Bioprospector	This Method	Experiment Verified	Discovered Motifs	Raw Scores
2	CE1CG	1	64,71	64	20,64	TTTGATCGTTTTCACA	6.472
3	ECOARABOP	1	58	58	20,58	TTTGCACGGCGTCACA	6.308
4	ECOBGLR1	1	79	79	79	TGTGAGCATGGTCATA	6.25
5	ECOCRP	1	66	66	66	TGCAAAGGACGTCACA	6.353
6	ECOCYA	1	38	53	53	TGTTAAATTGATCACG	6.264
7	ECODEOP2	1	63	10	10,63	TTTGAACCAGATCGCA	6.426
8	ECOGALE	2	27,45	45,54	45	TGTCACACTTTTCGCA	6.144
9	ECOILVBPR	2	42	25,42	42	TCTGCAATTCAGTACA	5.975
10	ECOLAC	2	12,15	12,83	12,83	TGTGAGTTAGCTCACT	6.473
11	ECOMALE	1	17	17	17	TGTAACAGAGATCACA	6.615
12	ECOMALK	2	59	64,32	32,64	CGTGATGTTGCTTGCA	6.096
13	ECOMALT	1	47	44	44	TGTGACACAGTGCAAA	6.447
14	ECOOMPA	1	51	51	51	CCTGACGGAGTTCACA	6.414
15	ECOTNAA	1	74	74	74	TGTGATTCGATTCACA	6.476
16	ECOUXU1	1	71	20	20	TGTGATGTGGTTAACC	6.249
17	PBR322	1	35	56	56	TGTGAAATACCGCACA	6.444
18	TRN9CAT	2	4	3,87	3,87	TGAGACGTTGATCGGC	5.604
19	TDC	1	81	81	81	TGTGAGTGGTCGCACA	6.435

Table 5.1 Results from testing data

The table listed the locations and the found motifs in each sequence, altogether there are 18 sequences identified motifs. The program did not found any motif instances from the two artificial sequences (not listed in the table). Actually, there are 24 motifs in this data set, and the program found out 23 copies where of which 21 copies are true motif. There are also 2 false positives and 3 true negatives.

The Sensitivity $Se = \frac{TP}{TP + FN}$ = 0.87, Specificity $S_p = \frac{TP}{TP + FP}$ = 0.91. The defination of Sensitivity and Specificity is shown in Fig 5.4

		Condition (as determined by "Gold standard")		
		Positive	*Negative*	
Test outcome	*Positive*	True Positive	False Positive (Type I error, P-value)	→ Positive predictive value
	Negative	False Negative (Type II error)	True Negative	→ Negative predictive value
		↓ Sensitivity	↓ Specificity	

Table 5.2 Sensitivity and Specificity

To make comparison, we used other programs to discover motifs from the same data set. The first program used is Bioprospector (Liu et al., 2001), the service is at http://bioprospector.stanford.edu. This program discovered 23 motifs, of which 12 motifs are exactly matches and 12 are missed. The sensitivity and specificity of this program are 0.5 and 0.52.

Chapter 6 Conclusion and Discussion

This thesis brought out a combined method to discover conserved TFBS motif of functional DNA sequences. The combined method is a mixture of a new scoring schema with mutual information and joint information content involved. It gets over the defect that the basic PWM model only contained either single position information or just neighbourhood base information. In addition, a varied Gibbs sampling algorithm with simulated tempering embedded was employed as the discover algorithm. This algorithm suits the situation of DNA sequence comprised no copy or multiple copies of motif (Fig).

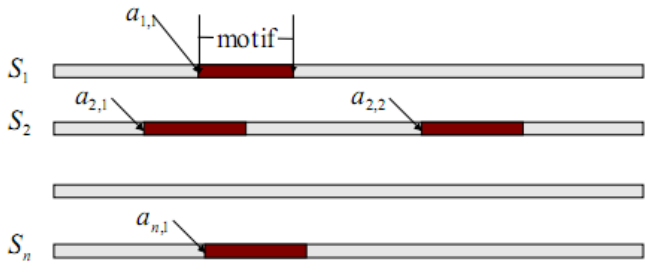


Fig 6.1 Illustration of multi-motif case

Through the analysis of a set of CRP binding gene sequences, the algorithm found out most motif instances of the binding site. The results excel that obtained by Bioprospector algorithm using default parameters. Results of the study case indicate that this method is feasible in motif discovery. In the implementation of simulated tempering into the traditional Gibbs sampling, ST proves to be a powerful solution for local optima problems found in pattern discovery. Extended application of simulated tempering for various bioinformatic problems is promising as a robust solution against local optima problems.

The new scoring schema improves TF binding site discovery and show that the joint information content and mutual information provide a better and more general criterion to investigate the relationships between positions in the TFBS. The scoring function is formulated by simple mathematical calculations and can be induced to perform better

than methods that do not consider dependencies between positions. Therefore the new method with the varied Gibbs sampling algorithm can be further applied in the field such as motif discovery or co-expressed gene analysis.

References

[1] Lawrence CE, Altschul SF, Boguski MS, Liu JS, Neuwald AF, Woontton JC: Detecting subtle sequence signals: a Gibbs sampling strategy for multi alginment, Science (1993).

[2] Bailey TL, Elkan C: Fitting a mixture model by expectation maximization to discover motifs in biopolymers, Proc Int Conf on Intell Syst Mol Biol (1994).

[3] Stormo GD: DNA binding sites: representation and discovery, Bioinformatics (2000).

[4] Jamshidian, Mortaza; Jennrich, Robert I : Acceleration of the EM Algorithm by using Quasi-Newton Methods, Journal of the Royal Statistical Society (1997).

[5] Thompson W, Rouchka EC, and Lawrence CE.: Gibbs Recursive Sampler: finding transcription factor binding sites Nucleic Acids Res (2003).

[6] K. Blekas, D. I. Fotiadis, A. Likas : A Sequential Method for Discovering Probabilistic Motifs in Proteins Bioinformatics (1998).

[7] Finding DNA Regulatory Motifs within Unaligned Non-Coding Sequences Clustered by Whole-Genome mRNA Quantitation, Roth, FR, Hughes, JD, Estep, PE & GM Church, Nature Biotechnology (1998).

[8] Liu X, Brutlag DL, Liu JS. BioProspector: discovering conserved DNA motifs in upstream regulatory regions of co-expressed genes. Pac Symp Biocomput. (2001).

[9] Vladislav Vyshemirsky, Mark Girolam BioBayes: A software package for Bayesian inference in systems biology Bioinformatics (2008).

[10] Geman, S.; Geman, D. "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images". IEEE Transactions on Pattern Analysis and Machine Intelligence (1984).

[11] Jensen LJ, Knudsen S: Automatic discovery of regulatory patterns in promoter regions based on whole cell expressiondata and functional annotation. Bioinformatics (2000).

[12] Van Helden J, Andre B, Collado-Vides J: Extracting regulatory sites from the upstream region of yeast genes by computational analysis of oligonucleotide frequencies. J Mol Biol (1998).

[13] Sinha S, Tompa M: A statistical method for finding transcription factor binding sites. Proc Int Conf Intell Syst Mol Biol (2000).

[14] Vanet A, Marsan L, Labigne A, Sagot MF: Inferring regulatory elements from a whole genome. An analysis of Helicobacter pylorisigma (80) family of promoter signals. J Mol Biol (2000).

[15] Liu XS, Brutlag DL, Liu JS: An algorithm for finding protein-DNA binding sites with applications to chromatin immuno-precipitation microarray experiments. Nat Biotechnol (2002).

[16] Marino-Ramirez L, Spouge JL, Kanga GC, Landsman D: Statistical analysis of over-represented words in human promoter sequences. Nucleic Acids Research (2004).

[17] Bailey TL, Elkan C: Unsupervised learning of multiple motifs in biopolymers using expectation maximization. Machine Learning (1995).

[18] Hughes JD, Estep PW, Tavazoie S, Church GM: Computational identification of cis-regulatory elements associated with groups of functionally related genes in Saccharomyces cerevisiae. J Mol Biol (2000).

[19] Workman CT, Stormo GD: ANN-Spec: a method for discovering transcription factor binding sites with improved specificity. Pac Symp Biocomput (2000).

[20] Frith MC, Fu Y, Yu L, Chen JF, Hansen U, Weng Z: Detection of functional DNA motifs via statistical over-representation. Nucleic Acids Res (2004).

[21] Favorov AV, Gelfand MS, Gerasimova AV, Ravcheev DA, Mironov AA, Makeev VJ: A Gibbs sampler for identification of symmetrically structured, spaced DNA motifs with improved estimation of the signal length. Bioinformatics (2005).

[22] N. Kwak, C.H. Choi, Input feature selection for classification problems, IEEE, Trans. Neural Networks (2002).

[23] N. Kwak, C.H. Choi, Input feature selection by mutual information based on Parzen window, IEEE Trans. Pattern Anal. Mach. Intell (2002).

[24] F. Fleuret, Fast binary feature selection with conditional mutual information, J. Mach. Learning Res (2004).

[25] N. Tishby, F.C. Pereira, W. Bialek, The information bottleneck method, in The 37th Annual Allerton Conference on Communication, Control and Computing, (1999).

[26] S.Winters-Hilt. Hidden Markov Model Variants and their Application. BMC Bioinformatics (2006).

[27] Stormo G: Information content and free energy in DNA-Protein interaction. J Theor Biol (1998).

[28] Benos P, Lapedes A, Stormo G: Probabilistic code for DNA recognition by proteins of EGR family. J Mol Biol (2002)

Appendix

A.1 Gibbs Sampling Source Code in PERL

```
#!/usr/perl/bin
use strict;
use FileHandle;
#-------------------------------------------------------------
# MAIN
#-------------------------------------------------------------
# basic parameters
my $motif_width=6;
my $num_seq=0;
my $seq_length;
my @rawSeq;
my @M;
my @startArray;
my $M_width=$motif_width+1;
#-------------------------------------------------------------
# read in raw genome sequences ,save to @rawSeq
my $input_data_fh = new FileHandle "<SD.txt";
while(<$input_data_fh>){
       chomp($_);
       $rawSeq[$num_seq]=$_;
       $num_seq++;
}
       $num_seq=scalar(@rawSeq);
#-------------------------------------------------------------
my $b=0.000000001;  # pseudocounts
my $B=$b*$num_seq;  # total pseudocounts
#-------------------------------------------------------------
$seq_length=length($rawSeq[0]);
#-------------------------------------------------------------
# Random start site for each sequence
my $possibleStartPostion=$seq_length-$motif_width;
for (my $i=0;$i<$num_seq ;$i++) {
       $startArray[$i]=int(rand($possibleStartPostion+1));
}

#-------------------------------------------------------------
```

```
# Kernel algorithm
#-------------------------------------------------------------

my $sthChanged=1;

while($sthChanged){

	$sthChanged=0;

	my $excluded=0;
	####################
	## Gibbs Sampling ##
	####################
	while($excluded<$num_seq){
		# ----------- predictive update step -----------

		# initialize matrix to all ZERO
		setZeroM();

		# count background and motif position
		# save to M, which is 4 * (motif_width+1)
		# 0th column is used to save background
		calcCountMatrix($excluded);

		# calculate frequency matrix based on count matrix
		calcFreqMatrix();

		# till now, information is learnt from N-1 and
		# we got the model matrix M

		# ----------- predictive update step -----------

		# ---------------- sampling step ---------------

		# the previously excluded sequence
		my $targetSeq=$rawSeq[$excluded];

		# try all possible start sites in $targetSeq,
		# choose one with the highest likelihood to our model
		my $likelihood=0;
		my $properStart;
		for(my $i=0;$i<=$seq_length-$motif_width;$i++){
```

```
                        # calculate likelihood, find MAX Likelihood
                        my $current = calcLikelihood($targetSeq,$i);
                        if($current>$likelihood){
                                $likelihood=$current;
                                $properStart=$i;
                        }
                }

                # if some start site is updated,
                # then switch the flag $sthChanged to TRUE
                if($properStart != $startArray[$excluded]){
                        $startArray[$excluded]=$properStart;
                        $sthChanged=1; # TRUE
                }

                # displayStartArray(); # for testing
                # ---------------- sampling step ---------------

                $excluded++; #go to next sequence
        }
        ####################
        ## Gibbs Sampling ##
        ####################
}

#---------------------------------------------------------------
# END : Kernel algorithm
#---------------------------------------------------------------

# OUTPUT

my $output_data_fh = new FileHandle ">GibbsResult.txt";

for (my $i=0;$i<$num_seq ;$i++) {
        my $currentMotif=substr($rawSeq[$i],$startArray[$i],$motif_width);
#       $output_data_fh->print($startArray[$i]," \t\t ",$currentMotif,"\n");

# Seq Logos Format
        $output_data_fh->print($currentMotif,"\n");
}

#---------------------------------------------------------------
```

```
# All support Functions
#-------------------------------------------------------------
sub calcLikelihood{
	my ($targetSeq,$startSite)=@_;
	my $result=1;
	for(my $i=$startSite;$i<=$startSite+$motif_width-1;$i++){
		my $symbol;
		my $base=substr($targetSeq,$i,1);
		if($base eq 'A'){$symbol=0;}
		if($base eq 'C'){$symbol=1;}
		if($base eq 'G'){$symbol=2;}
		if($base eq 'T'){$symbol=3;}

		$result*= $M[$symbol][$i-$startSite+1]/$M[$symbol][0];
	}

	# print "\n $targetSeq :result = $result \n";
	return $result;
}
#-------------------------------------------------------------
sub displayStartArray{
	print"\n---------------------\n";
	for (my $i=0;$i<$num_seq ;$i++) {
		print $startArray[$i],"\n";
	}
	print"---------------------\n";
}
#-------------------------------------------------------------
sub setZeroM{
	for (my $i=0;$i<4;$i++) {
		for (my $j=0;$j<$M_width;$j++) {
			$M[$i][$j]=0;
		}
	}
}
#-------------------------------------------------------------
sub displayM(){
	print "\n---------------------------------\n";
	for (my $i=0;$i<4;$i++) {
		for (my $j=0;$j<$M_width;$j++){
			print $M[$i][$j],"\t";
		}
```

```
		print "\n";
	}
	print "--------------------------------\n";
}
#-------------------------------------------------------------
sub calcCountMatrix{
	my ($excluded)=@_;
	for(my $iter=0;$iter<$num_seq;$iter++){
		#calculate count matrix M for N-1 sequences
		if($iter==$excluded) {next;}
		my $currentSeq=$rawSeq[$iter];
		# scan current Seq, update matrix M;
		for(my $i=0;$i<$seq_length;$i++){

			my $base=substr($currentSeq,$i,1);
			my $symbol;
			if($base eq 'A'){$symbol=0;}
			if($base eq 'C'){$symbol=1;}
			if($base eq 'G'){$symbol=2;}
			if($base eq 'T'){$symbol=3;}
			# print "$base ";
	if($i>=$startArray[$iter] and i<=$startArray[$iter]+$motif_width-1){
				my $motif_pos=$i-$startArray[$iter];
				$M[$symbol][$motif_pos+1]=$M[$symbol][$motif_pos+1]+1;
			}
			else{
				$M[$symbol][0]=$M[$symbol][0]+1;
			}
		}# print "\nEND\n";
		# scan this sequence END
	} # Have got count Matrix 'M' with excluded sequence excluded :)
}
#-------------------------------------------------------------
sub calcFreqMatrix{
	# calculate freq Matrix from count Matrix
	for(my $i=0;$i<4;$i++){
		my $temp=$M[$i][0];
		$M[$i][0]=  ($temp+$b)/(($num_seq-1)*($seq_length-$motif_width)+$B);
	}

	for(my $i=0;$i<4;$i++){
		for (my $j=1;$j<=$motif_width ;$j++) {
```

```
                    $M[$i][$j]=($M[$i][$j]+$b)/($num_seq-1+$B);
                }
        }
        # END : calculate freq Matrix from count Matrix
}
#-------------------------------------------------------------
```

A.2 Simulated Tempering Code in C++

simulatedtempering.h

```
00066 #ifndef SIMULATEDTEMPERING_H_
00067 #define SIMULATEDTEMPERING_H_
00068
00069 #include "mcmc.h"
00070 #include "simulatedtemperingparams.h"
00071 #include "maxwalksat.h"
00072 #include "convergencetest.h"
00073 #include "gelmanconvergencetest.h"
00074
00078 class SimulatedTempering : public MCMC
00079 {
00080  public:
00081
00085   SimulatedTempering(VariableState* state, long int seed,
00086                      const bool& trackClauseTrueCnts,
00087                      SimulatedTemperingParams* stParams)
00088     : MCMC(state, seed, trackClauseTrueCnts, stParams)
00089   {
00090       // User-set parameters
00091     subInterval_ = stParams->subInterval;
00092     numST_ = stParams->numST;
00093     numSwap_ = stParams->numSwap;
00094       // Number of chains is determined here
00095     numChains_ = numSwap_*numST_;
00096     // ---------------------------------------- //
00097     // Chained method
00098     //  10 chains: i and i+1 swap attempt at
00099     //      selInterval*k + selInterval/10*i
00100     // ---------------------------------------- //
00101       // 9 possible swaps out of 10 chains
00102     selInterval_ = subInterval_*(numSwap_ - 1);
00103
00104       // invTemp for chain chainIds_[i]
00105     invTemps_ = new double*[numST_];
00106       // curr chainId for ith temperature
00107     chainIds_ = new int*[numST_];
00108       // curr tempId for ith chain
00109     tempIds_ = new int*[numST_];
00110       // We don't need to track clause true counts in mws
00111     mws_ = new MaxWalkSat(state_, seed, false, stParams->mwsParams);
00112   }
00113
00117   ~SimulatedTempering()
00118   {
00119     for (int i = 0; i < numST_; i++)
00120     {
00121       delete [] invTemps_[i];
```

```
00122       delete [] chainIds_[i];
00123       delete [] tempIds_[i];
00124     }
00125     delete [] invTemps_;
00126     delete [] chainIds_;
00127     delete [] tempIds_;
00128     delete mws_;
00129   }
00130
00134   void init()
00135   {
00136       // Initialize gndPreds' truthValues & wts
00137     //state_->initTruthValuesAndWts(numChains_, start);
00138     initTruthValuesAndWts(numChains_);
00139
00140       // Initialize with MWS
00141     cout << "Initializing Simulated Tempering with MaxWalksat" << endl;
00142     state_->eliminateSoftClauses();
00143       // Set num. of solutions temporarily to 1
00144     int numSolutions = mws_->getNumSolutions();
00145     mws_->setNumSolutions(1);
00146     for (int c = 0; c < numChains_; c++)
00147     {
00148       cout << "for chain " << c << "..." << endl;
00149         // Initialize with MWS
00150       mws_->init();
00151       mws_->infer();
00152       saveLowStateToChain(c);
00153     }
00154     mws_->setNumSolutions(numSolutions);
00155     state_->resetDeadClauses();
00156
00157     // *** Initialize temperature schedule ***
00158     double maxWt = state_->getMaxClauseWeight();
00159     double maxWtForEvenSchedule = 100.0;
00160     double base = log(maxWt) / log((double)numSwap_);
00161     double* divs = new double[numSwap_];
00162     divs[0] = 1.0;
00163
00164     for (int i = 1; i < numSwap_; i++)
00165     {
00166       divs[i] = divs[i - 1] / base;
00167     }
00168
00169     for (int i = 0; i < numST_; i++)
00170     {
00171       invTemps_[i] = new double[numSwap_];
00172       chainIds_[i] = new int[numSwap_];
00173       tempIds_[i]  = new int[numSwap_];
00174       for (int j = 0; j < numSwap_; j++)
00175       {
00176         chainIds_[i][j] = j;
00177         tempIds_[i][j] = j;
00178           // log vs even
00179         if (maxWt > maxWtForEvenSchedule)
00180         {
00181           invTemps_[i][j] = divs[j];
00182         }
00183         else
00184         {
00185           invTemps_[i][j] = 1.0-((double)j)/((double) numSwap_);
00186         }
00187       }
```

```
00188     }
00189     delete [] divs;
00190
00191       // Initialize gndClauses' number of satisfied literals
00192     //int start = 0;
00193     initNumTrueLits(numChains_);
00194   }
00195
00199   void infer()
00200   {
00201     initNumTrue();
00202     Timer timer;
00203       // Burn-in only if burnMaxSteps positive
00204     bool burningIn = (burnMaxSteps_ > 0) ? true : false;
00205     double secondsElapsed = 0;
00206     double startTimeSec = timer.time();
00207     double currentTimeSec;
00208     int samplesPerOutput = 100;
00209
00210       // If keeping track of true clause groundings, then init to zero
00211     if (trackClauseTrueCnts_)
00212     for (int clauseno = 0; clauseno < clauseTrueCnts_->size();clauseno++)
00213         (*clauseTrueCnts_)[clauseno] = 0;
00214
00215       // Holds the ground preds which have currently been affected
00216     GroundPredicateHashArray affectedGndPreds;
00217     Array<int> affectedGndPredIndices;
00218
00219     int numAtoms = state_->getNumAtoms();
00220     for (int i = 0; i < numAtoms; i++)
00221     {
00222       affectedGndPreds.append(state_->getGndPred(i), numAtoms);
00223       affectedGndPredIndices.append(i);
00224     }
00225     for (int c = 0; c < numChains_; c++)
00226       updateWtsForGndPreds(affectedGndPreds, affectedGndPredIndices, c);
00227     affectedGndPreds.clear();
00228     affectedGndPredIndices.clear();
00229
00230     cout << "Running Simulated Tempering sampling..." << endl;
00231       // Sampling loop
00232     int sample = 0;
00233     int numSamplesPerPred = 0;
00234     bool done = false;
00235     while (!done)
00236     {
00237       ++sample;
00238
00239       if (sample % samplesPerOutput == 0)
00240       {
00241         currentTimeSec = timer.time();
00242         secondsElapsed = currentTimeSec-startTimeSec;
00243    cout << "Sample (per pred per chain) " << sample << ", time elapsed =;
00244         Timer::printTime(cout, secondsElapsed); cout << endl;
00245       }
00246
00247         // Attempt to swap temperature
00248       if ((sample % selInterval_) % subInterval_ == 0)
00249       {
00250         int attemptTempId = (sample % selInterval_) / subInterval_;
00251         if (attemptTempId < numSwap_ - 1)
00252         {
00253           double wl, wh, itl, ith;
```

```
00254           for (int i = 0; i < numST_; i++)
00255           {
00256             int lChainId = chainIds_[i][attemptTempId];
00257             int hChainId = chainIds_[i][attemptTempId + 1];
00258               // compute w_low, w_high: e = -w
00259               // swap acceptance ratio=e^(0.1*(w_h-w_l))
00260             wl = getWeightSum(i*numSwap_ + lChainId);
00261             wh = getWeightSum(i*numSwap_ + hChainId);
00262             itl = invTemps_[i][attemptTempId];
00263             ith = invTemps_[i][attemptTempId + 1];
00264
00265           if (wl <= wh || random() <= RAND_MAX*exp((itl - ith)*(wh - l)))
00266             {
00267               chainIds_[i][attemptTempId] = hChainId;
00268               chainIds_[i][attemptTempId+1] = lChainId;
00269               tempIds_[i][hChainId] = attemptTempId;
00270               tempIds_[i][lChainId] = attemptTempId + 1;
00271             }
00272           }
00273         }
00274       }
00275
00276         // Generate new truth value based on temperature
00277       for (int c = 0; c < numChains_; c++)
00278       {
00279           // For each block: select one to set to true
00280         for (int i = 0; i < state_->getDomain()->getNumPredBlocks(); i++)
00281         {
00282             // If evidence atom exists, then all others stay false
00283           if (state_->getDomain()->getBlockEvidence(i)) continue;
00284
00285           double invTemp =
00286             invTemps_[c/numSwap_][tempIds_[c/numSwap_][c%numSwap_]];
00287   // chosen is index in the block, block[chosen] is index in gndPreds_
00288           int chosen = gibbsSampleFromBlock(c, i, invTemp);
00289
00290           const Predicate* pred =
00291             state_->getDomain()->getPredInBlock(chosen, i);
00292         GroundPredicate* gndPred = new GroundPredicate((Predicate*)pred);
00293           int idx = state_->getIndexOfGroundPredicate(gndPred);
00294
00295           delete gndPred;
00296           delete pred;
00297
00298             // If gnd pred in state:
00299           if (idx >= 0)
00300           {
00301             bool truthValue = truthValues_[idx][c];
00302               // If chosen pred was false, then need to set previous true
00303               // one to false and update wts
00304             if (!truthValue)
00305             {
00306               int blockSize = state_->getDomain()->getBlockSize(i);
00307               for (int j = 0; j < blockSize; j++)
00308               {
00309                 const Predicate* otherPred =
00310                   state_->getDomain()->getPredInBlock(j, i);
00311                 GroundPredicate* otherGndPred =
00312                   new GroundPredicate((Predicate*)otherPred);
00313                int otherIdx = state_->getIndexOfGroundPredicate(gndPred);
00314
00315                 delete otherGndPred;
00316                 delete otherPred;
```

```
00317
00318                 // If gnd pred in state:
00319               if (otherIdx >= 0)
00320               {
00321                 bool otherTruthValue = truthValues_[otherIdx][c];
00322                 if (otherTruthValue)
00323                 {
00324                   truthValues_[otherIdx][c] = false;
00325
00326                   affectedGndPreds.clear();
00327                   affectedGndPredIndices.clear();
00328                   gndPredFlippedUpdates(otherIdx, c, affectedGndPreds,
00329                                         affectedGndPredIndices);
00330                   updateWtsForGndPreds (affectedGndPreds,
00331                                         affectedGndPredIndices, c);
00332                 }
00333               }
00334             }
00335               // Set truth value and update wts for chosen atom
00336             truthValues_[idx][c] = true;
00337             affectedGndPreds.clear();
00338             affectedGndPredIndices.clear();
00339             gndPredFlippedUpdates(idx, c, affectedGndPreds,
00340                                   affectedGndPredIndices);
00341       updateWtsForGndPreds(affectedGndPreds, affectedGndPredIndices, c);
00342           }
00343
00344             // If in actual sampling phase, track the num of times
00345             // the ground predicate is set to true
00346           if (!burningIn && tempIds_[c/numSwap_][c%numSwap_] == 0)
00347             numTrue_[idx]++;
00348         }
00349       }
00350
00351         // Now go through all preds not in blocks
00352       for (int i = 0; i < state_->getNumAtoms(); i++)
00353       {
00354           // Predicates in blocks have been handled above
00355         if (state_->getBlockIndex(i) >= 0) continue;
00356           // Calculate prob
00357         double invTemp =
00358           invTemps_[c/numSwap_][tempIds_[c/numSwap_][c%numSwap_]];
00359         double p = getProbabilityOfPred(i, c, invTemp);
00360
00361           // Flip updates
00362         bool newAssignment = genTruthValueForProb(p);
00363         //if (newAssignment != pred->getTruthValue(c))
00364         if (newAssignment != truthValues_[i][c])
00365         {
00366           //pred->setTruthValue(c, newAssignment);
00367           truthValues_[i][c] = newAssignment;
00368           affectedGndPreds.clear();
00369           affectedGndPredIndices.clear();
00370           gndPredFlippedUpdates(i, c, affectedGndPreds,
00371                                 affectedGndPredIndices);
00372       updateWtsForGndPreds(affectedGndPreds, affectedGndPredIndices, c);
00373         }
00374
00375           // if in actual sim. tempering phase, track the num of times
00376           // the ground predicate is set to true
00377         if (!burningIn && newAssignment &&
00378             tempIds_[c/numSwap_][c%numSwap_] == 0)
00379           //pred->incrementNumTrue();
```

```
00380             numTrue_[i]++;
00381         }
00382       }
00383       if (!burningIn) numSamplesPerPred += numST_;
00384
00385         // If keeping track of true clause groundings
00386       if (!burningIn && trackClauseTrueCnts_)
00387         state_->getNumClauseGndings(clauseTrueCnts_, true);
00388
00389       if (burningIn)
00390       {
00391         if (   (burnMaxSteps_ >= 0 && sample >= burnMaxSteps_)
00392             || (maxSeconds_ > 0 && secondsElapsed >= maxSeconds_))
00393         {
00394           cout << "Done burning. " << sample << " samples per chain " <<
endl;
00395           burningIn = false;
00396           sample = 0;
00397         }
00398       }
00399       else
00400       {
00401         if (   (maxSteps_ >= 0 && sample >= maxSteps_)
00402             || (maxSeconds_ > 0 && secondsElapsed >= maxSeconds_))
00403         {
00404           cout << "Done simulated tempering sampling. " << sample
00405                << " samples per chain" << endl;
00406           done = true;
00407         }
00408       }
00409       cout.flush();
00410     } // while (!done)
00411
00412     cout<< "Time taken for Simulated Tempering sampling = ";
00413     Timer::printTime(cout, timer.time() - startTimeSec); cout << endl;
00414
00415       // update gndPreds probability that it is true
00416     for (int i = 0; i < state_->getNumAtoms(); i++)
00417     {
00418       //GroundPredicate* gndPred = state_->getGndPred(i);
00419       //gndPred->setProbTrue(gndPred->getNumTrue() / numSamplesPerPred);
00420       setProbTrue(i, numTrue_[i] / numSamplesPerPred);
00421     }
00422
00423       // If keeping track of true clause groundings
00424     if (trackClauseTrueCnts_)
00425     {
00426         // Set the true counts to the average over all samples
00427       for (int i = 0; i < clauseTrueCnts_->size(); i++)
00428        (*clauseTrueCnts_)[i] = (*clauseTrueCnts_)[i] / numSamplesPerPred;
00429     }
00430   }
00431
00432  private:
00433
00441   long double getWeightSum(const int& chainIdx)
00442   {
00443     long double w = 0;
00444     for (int i = 0; i < state_->getNumClauses(); i++)
00445     {
00446       long double wt = state_->getClauseCost(i);
00447       if ((wt > 0 && numTrueLits_[i][chainIdx] > 0) ||
00448           (wt < 0 && numTrueLits_[i][chainIdx] == 0))
```

```
00449         w += abs(wt);
00450     }
00451     return w;
00452   }
00453
00454   private:
00455
00456     // User-set parameters:
00457     // Selection interval between swap attempts
00458   int subInterval_;
00459     // Number of simulated tempering runs
00460   int numST_;
00461     // Number of swapping chains
00462   int numSwap_;
00463
00464     // MaxWalksat is used for initialization
00465   MaxWalkSat* mws_;
00466
00467     // 9 possible swaps out of 10 chains
00468   int selInterval_;
00469     // invTemp for chain chainIds_[i]
00470   double** invTemps_;
00471     // curr chainId for ith temperature
00472   int** chainIds_;
00473     // curr tempId for ith chain
00474   int** tempIds_;
00475 };
00476
00477 #endif /*SIMULATEDTEMPERING_H_*/
```

A.3 Mutual Information Source Code in PERL

```
#!/usr/perl/bin
use strict;
use FileHandle;
use Data::Dumper;
#>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
my @stat;

my @bases=("A","C","G","T");
my $output = new FileHandle ">MIresults.txt";
my $stat = new FileHandle ">Analysis.txt";

my $index=1;

F:
my $start_position=-1; # anchored !!!

my $length=18;
my $width=21;
my $order=3;


#
my @start_array;
toArray();
#
my %p_i;
get_p_i();
#
```

```
###    Get Start position with highest information, i.e., lowest entropy!

my $lowest=999;


for (my $ind=0; $ind<$width ;$ind++) {

       my @bases=("A","C","G","T");

       my $sum=0;

       foreach  my $b (@bases) {
              my $p = $p_i{$ind}{$b};
              $sum = $sum - $p * log2($p) ;
       }

       if($sum<$lowest){
              $lowest=$sum;
              $start_position=$ind;
       }

}
###    Get Start position with highest information, i.e., lowest entropy!

print "$start_position *\n";

my @done_arr=($start_position);
my @ordered_position=($start_position);
my @ordered_mi=();
my $order_backup=$order;

while($order>0){
       my $max_mi=-1;
       my $max_mi_index=-1;
       my $current;
       S:for (my $index=0;$index<$width;$index++) {
                     foreach my $omission (@done_arr) {
                            if($index==$omission){next S;}
                     }
                     $current=MI($index,@done_arr);
                     if($current>$max_mi){
                            $max_mi=$current;
                            $max_mi_index=$index;
                     }
              }
       push @done_arr,$max_mi_index;
       push @ordered_position,$max_mi_index;
       push @ordered_mi,$max_mi;
       $order--;
}

for (my $i=0;$i<scalar(@ordered_position);$i++) {
       my $temp=$ordered_position[$i];
       $output->print ("$temp\t");
       $stat[$temp]++;
}

       $output->print("\n");

$index++;
if ($index<=1000) {
       goto F;
```

```
}

foreach my $elem (@stat) {
	$stat->print("$elem\t");
}

#>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
sub toArray
{
	my @line_array  = ();
	my $line_index=0;
	my $data_fh = new FileHandle "<GibbsResult$index.txt";
	while (<$data_fh>) {
		chomp;
		@line_array = split '';

		@{$start_array[$line_index]}=@line_array;
		$line_index++;
		@line_array  = ();
	}
}
#-----------------------------------------------------------
sub get_p_i{
	my %P_xi_count;
	my ($row,$column);

	for ($column=0; $column<$width ;$column++) {
		my $base;
		for ($row=0;$row<$length ;$row++) {
			$base = $start_array[$row][$column];
			$P_xi_count{$column}{$base}++;
		}
	}

	my @bases=("A","C","G","T"); # calc %p_i
	for ($column=0; $column<$width ;$column++) {
		foreach  my $base (@bases) {
			$p_i{$column}{$base}=$P_xi_count{$column}{$base}/18.0;
		}
	}

}
#-----------------------------------------------------------
sub log2{
	my ($in) = @_;
	if($in==0){
		$in=0.0001;
	}

	my $result = log($in)/log(2);

	if($result==0){
		return 0.0001;
	}
	else{
		return $result;
	}
}
#-----------------------------------------------------------
sub MI_atom{
	my ($Pxy,$Px,$Py) = @_;
	if($Px == 0){$Px=0.0001;}
	if($Py == 0){$Py=0.0001;}
```

```
        my $p = $Pxy/($Px*$Py);
        if($p == 0){$p = 0.0001;}
        return $Pxy * log2($p);
}
#-----------------------------------------------------------
sub MI{ # P(Xi ; X24,X15...)
# this is the optimized mutual information calculator
        my ($single,@members)=@_;
        my (%p_rest_count,%p_combination_count);

        for (my $index=0;$index<$length ;$index++) {
                my ($head,$rest);
                $head=$start_array[$index][$single];
                foreach my $i (@members) {
                        $rest=$rest.$start_array[$index][$i];
                }
                $p_rest_count{$rest}++;

                my $combination = "$head$rest";
                $p_combination_count{$combination}++;
        }
        my $mi=0;
        foreach my $combi (keys %p_combination_count) {
                my ($head,$rest);
                $head = substr($combi,0,1);
                $rest = substr($combi,1);
                my $Pxy = $p_combination_count{$combi}/$length;
                my $Py = $p_rest_count{$rest}/$length;
                my $Px = $p_i{$single}{$head};
                $mi += MI_atom($Pxy,$Px,$Py);
        }
        return $mi;}
```

A.4 Gibbs Sampling Source Code in PERL

Mutual Information provides a measure of the interdependence between random variables, (X;Y), or groupings of random variables, (A,B;X,Y,Z). For the base definition, consider two random variables X and Y with joint distribution p(x,y) and individual (marginal) distributions p(x) and p(y), then the MI(X;Y) is:

$$MI(X,Y) = \sum_{x}\sum_{y} p(x,y)\log\frac{p(x,y)}{p(x)p(y)}$$

If X and Y are independent r.v's then MI=0. If we have a DNA sequence $x_1 \ldots . x_i x_{i+1} x_{i+2} \ldots \ldots . x_n$ (where x_k ={a,c,g, or t}) then we can get counts on pairs $x_i x_{i+1}$ for i=1..n, and assuming stationarity on the data and large enough n, we can speak of the joint probability p(X,Y). Calculation of MI(X,Y) then gives an indication of the linkage

between base probabilities in dinucleotide probabilities. This can be extended to linkages when the two bases aren't sequential (have a base gap between them greater than zero), such as pairs based on $x_i x_{i+2}$ (gap=1), etc. This type of statistical framework can then be iterated to higher order MI calculations in a variety of ways to explore a number of statistical linkages and build towards a motif identifier based on such linkages.

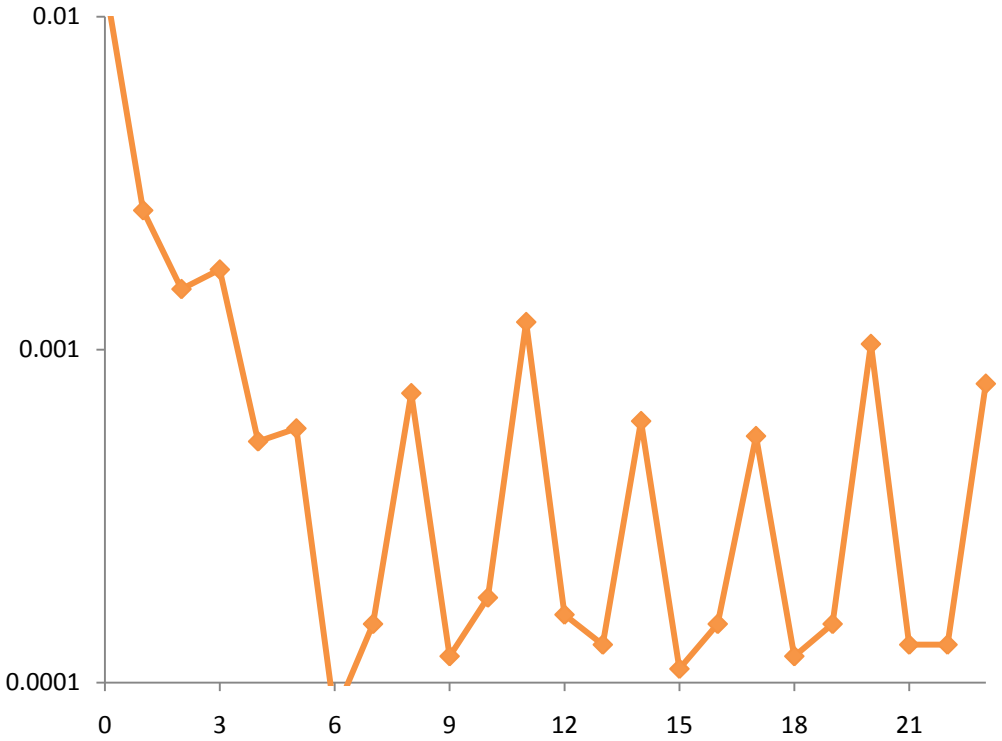


Fig A.4.1 codon structure revealed, hexamer stat's good

Towards Given 'ATG' start coding site in Vibrio Cholarae, the conserved upstream regulaton, Shine-Dalgarno sequence, was captured via mutual information.

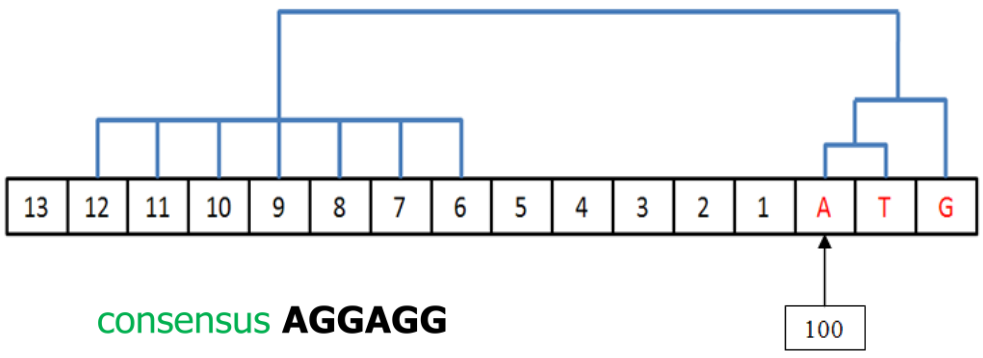


Fig A.4.2 Shine-Dalgarno Sequence caputured by mutual information

Vita

Daming Lu was born in Dalian, Liaoning, China. In 2007, he graduated from Dalian University of Technology, China with a Bachelor's Degree of Engineering in Computer Science and Technology. He will start a one-year internship in August, 2009 but plans to eventually return to academia to work on a doctorate degree in Computer Science.