8-2012

A New Algorithm for DNA Motif Detection

Ran Ding
University of Texas-Pan American

A NEW ALGORITHM FOR DNA MOTIF DETECTION

A Thesis

by

RAN DING

Submitted to the Graduate School of the
University of Texas-Pan American
In partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2012

Major Subject: Computer Science

A NEW ALGORITHM FOR DNA MOTIF DETECTION

A Thesis
by
RAN DING

COMMITTEE MEMBERS

Dr. Bin Fu
Chair of Committee

Dr. Zhixiang Chen
Committee Member

Dr. Artem Chebotko
Committee Member

August 2012

ABSTRACT

Ding, Ran, A New Algorithm for DNA Motif Detection. Master of Science (MS), August, 2012, 31 pp., 18 references, 5 titles.

Nowadays, people are more and more concentrated on the research of deoxyribonucleic acid(DNA). And a very important problem is how to identify the DNA performance. A DNA sequence contains lots of genes, and every gene has a regulatory region. This regulatory region contains the transcription factor binding sites (TFBS) which is also known as motif. How to identify this motif becomes a very important problem. There are many efficient methods and tools are dedicated to the research of DNA motif detection.

In this paper, we will introduce a new method of DNA motif detection which basically is an advanced approximate voting algorithm. Through a series of experiments, we find that this method performs well on small number for input sequences compared with other famous tools.

DEDICATION

The completion of my master studies would not have been possible without the love and support of my family. My father, Honghui Ding and my mother Aiying Qin, wholeheartedly inspired, motivated and supported me by all means to accomplish this degree. Thank you for your love and patience.

ACKNOWLEDGMENTS

I will always be grateful to Dr. Bin Fu, chair of my dissertation committee, for all his mentoring and advice. Under his supervision, I have greatly enhanced my knowledge and skills in the areas of bioinformatics and algorithms. Finally, my thanks go to the committee members: Dr. Zhixiang Chen and Dr. Artem Chebotko. Their advice and comments on my thesis helped the quality of my intellectual work.

TABLE OF CONTENTS

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER I

INTRODUCTION

For a long time till today, finding motif in DNA sequences is an important research in genetics. Every gene contains the motifs also known as transcription factor binding sites are inside of the regular region. The transcription factor may have different expressions due to location of the binding sites. So if we want to find the feature of the sequence, we have to find the motifs. Meanwhile, since this problem has a high complexity, how to develop a suitable algorithm to detect the motifs becomes an important problem. Nowadays, there are three major heuristic algorithms.[4]

Enumeration as its name, it will go through the whole sequences to detect the motif. Early enumeration can only handle the exactly same pattern as the given motif. But the gene is highly random; the researchers find that it may not be rigorous if the algorithm only handles the same patterns. So after the further development, it has the fault tolerance which means it can detect the patterns with a certain amount of mismatches. The main problem of this type of algorithm is it has to cover the whole DNA sequences so that it may have a bad performance when the number of sequences is huge.[4]

The voting algorithm is easy to be understood. First there is a set of sequences with the same length M, and a given pattern t with length L. Then use this given pattern to compare with the L length patterns in the given set of sequences and calculate the distance between them. The distance here means how many different letters between these two patterns. If the two patterns are the same, then put the patterns from sequences into a new set as the result. One paper named"Voting Algorithm for the Motif Finding Problem" [17] gives a detailed introduction of voting algorithm.

The present of deterministic optimization algorithm is the expectation maximization (EM) algorithm, which is a statistical method and was first provided in 1977 by Arthur Dempster, Nan Laird and Donald Rubin. They use a position weight matrix (PWM) to describe the motifs. First it will calculate the probability based on the given motif pattern then expectation maximization will calculate the position weight and get a new motif model. The important feature of this type of algorithm is through the calculation; it may get the best motif for the sequences. Meanwhile the disadvantage is obvious; it needs a lot of computation. [11]

The most famous tool based on EM algorithm is MEME suit (Multiple EM for Motif Elicitation). Basically, it can detect the motifs through EM algorithm. It accepts the normal

sequences input and the fasta format files input. On the official web site, you can submit a job to find the result for the sequences. It connects with its own DNA databases–MEME-ChIP, so that every time it works on a job it will search in the database to get a more reliable result. It also can compare the result to the motif database to see if the motif is existing. It has a couple of tools to support finding the motifs, like MAST (Motif Alignment and Search tool), GLAM2 (Gapped Local Alignment of Motifs), GOMO (Gene Ontology for Motifs) and so on. It is truly a convenient tool to detect the motif. [1]

There is another alternative deterministic optimization algorithm which is very popular now. Gibbs sampling is the one. [2, 10, 13] This algorithm is named after J.W. Gibbs and first described in 1984 by brothers Stuart and Donald Geman. From the given sequences, first initial the start point of the motif at random and then calculate the appearance of each letter for each position. Then turn the matrix into probability matrix. The next step is calculating the division between motif part and the background part. Then it can get the new position of the motif. This type of algorithm is heuristic, and the most important advantage is that it does not need the initial given motif. The former two need a given motif to compare with the sequences and Gibbs sampling does not. But the limit of this is it depends on the sequences. If the sequences do not contain the motif, it is very hard to sample a correct motif by this algorithm.

Gibbs sampling also has a good implementation tool called Gibbs Motif Sampler. This one also accepts the normal sequences input and the fasta format files input. It has 4 different types of samplers–Site Sampler, Motif Sampler, Recursive Sampler and Centroid Sampler. Site sampler will give the result motif which exists in most of the sequences and the Motif sampler will give the result motif which may exist in one or more sequences. Recursive sampler can use the recursive way to calculate the ratio for the letters in the motif results and replace it. Centroid

sampler collects all the possible motif results and gets the minimum distance between two alignments. It is also a efficient tool for detecting motifs.

About these three mainly algorithms, there is a survey [3] well described. In that survey, it mentions that early algorithms about finding motif deals with the sequences from one gene and looking for the highest ratio motifs. They can perform good on some simulate data but the application is not good on the real world data. And different algorithms perform different when they deal with different input data.

There are lots of other tools for detecting and analyzing the motifs, like Compo[15], MochiView[6], PhyME[16], HeliCis[9], WebMotif[14] and so on. Each of them has their advantages and disadvantages. Like Compo performs very well on a collection of benchmark data sets. MochiView is good at organizing and analyzing large genomic data sets. PhyME uses some original discovered information from the data to detect the motif and it is efficient. HeliCis does a good job on analyzing the motif results. It can get the conclusion that whether this result is the best for this DNA data. WebMotif uses a web server to work and it uses several different programs to detect the motifs.

Overall there are lots of implementations now, and each has their strong point. In this paper, we only use our implementation compared with MEME and Gibbs Sampling, because these two are the most popular methods today.

Of course, the algorithm of detecting motif is not the only problem on detecting motifs. Analyzing data resource is one of them too. One set of DNA sequences may have lots of different motifs [8, 18]. So how to tell which the best is for this DNA is also important. Also one paper named “ELM: the status of the 2010 eukaryotic linear motif resource” [5] mentions that chose a suitable database can make the detecting more efficient.

One interesting problem is that if with the input sequences grow, the accuracy for the motif finding should be better or not. One good point mentioned in paper “Limitations and Potentials of Current motif discovery algorithms” [7] is that through the experiment on the numbers of input sequences for different detecting tools, the accuracy is not increased. Most of them are decreasing and only a few of them is on the same level. This point is very meaningful. The more input sequences are selected, the more noise is joined. So it is better to input a reasonable number of sequences to detect the motifs. And there may be some other factors to infect the performance of detecting, like in agriculture area, fusarium graminearum(Fg)[12] may be well detected by a computational pipeline to systematically discover but not others.

CHAPTER II

OVERVIEW OF MAIN ALGORITHMS

In this section, I will present some important algorithms and protocols which are used. Also there will be a brief flow diagram to show how the program runs.

Voting algorithm plays an important role in the whole program. It is used in the last part of the progress which is for recovering the possible result motifs. The advantage of this algorithm is it can give a comparatively accurate final result.

The algorithm is:

1. Input a pattern set T with length L.

2. For each pattern of T, compare all the input patterns and for each position to vote one character that appears most.

3. The output is a new pattern t which has the minimum distance to all the patterns.

Calling it Approximate Voting Algorithm because it has the important core idea of voting algorithm, but meanwhile, I use the different comparisons and output to accomplish my object.

The output can bring us the best possible recovery motif, although it may seems like have some disadvantage in fault-tolerant, actually, it can maintain the different letter in the sequences and give the result of the one appears the most. And in the program, there are lots of sequences-to-sequences comparisons so that they can make sure to work at a good mutation rate. That is why I think it would not hold back the performance.

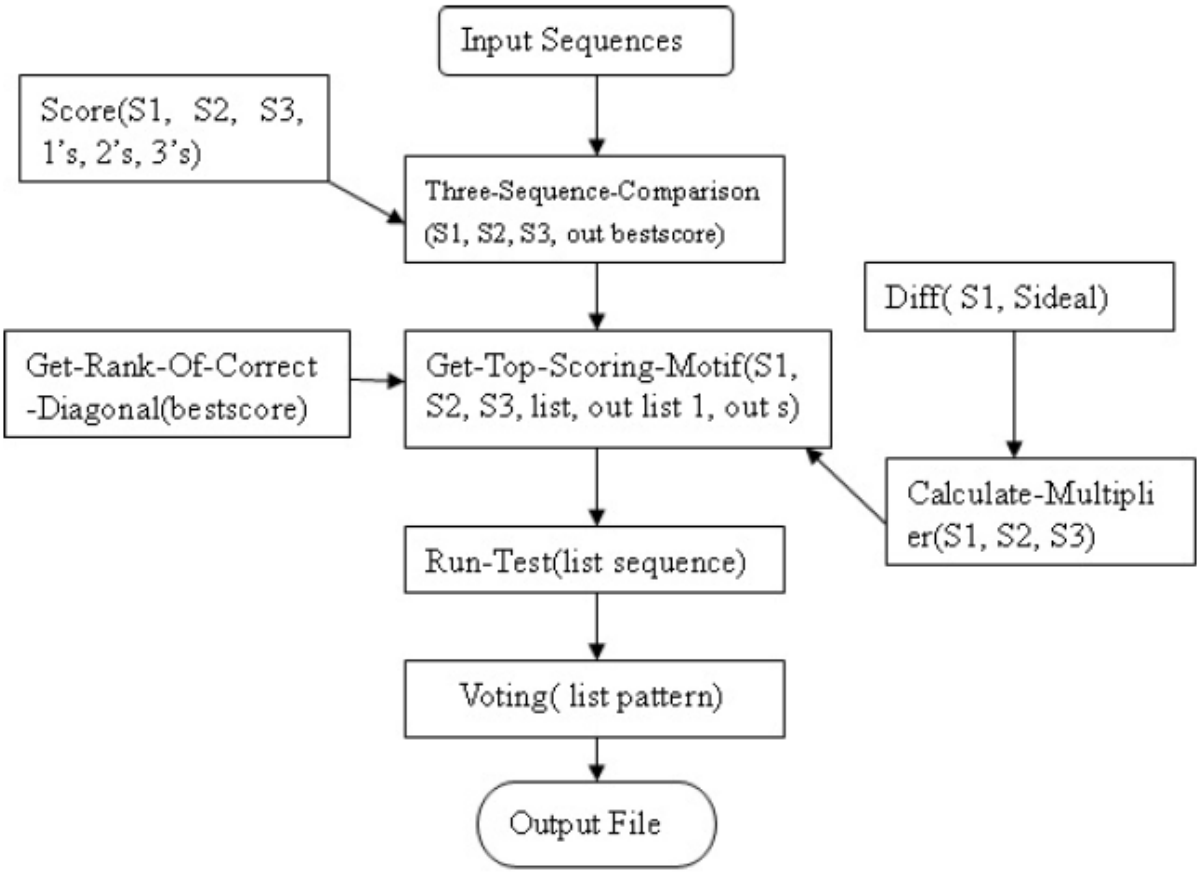

Fig 2.1. Flow Diagram

CHAPTER III

ALGORITHM

In this section, I will introduce some key parameters that are used in the algorithm.

First is the five sequences comparison. I can set the sequence length m and the number of sequences is n. Assume a certain parameter a as a mutation rate in this program, and a = 0.27. S is a set of input sequences.

It is a sequences-to-sequences comparison, so every time after one comparison, the object sequence will shift to left a certain position then do another comparison. This certain distance I set it to motif length L. And the result should be candidate motif.

After this, we have to calculate the score of each candidate motif. We compare them with ideal motif patterns. And we have a parameter diff to record the different rate about the comparison. The multiplier is equal to the number of different letters we use the parameter c divided by the length of patterns which is motif length L. diff = c / L. And then, the multiplier is multi = $\sqrt{\prod_{n=1}^{5} diff(Si, Sj)} * 0.5 + 0.5$. And this multiplier is for the further use.

Then the target is to find the top scoring candidate motif patterns. Instead of 2-sequence comparisons we make 3-sequence comparisons. We select three sequences out of five sequences and make the first one as the comparable sequence and shift the other two to do the comparisons. In this function, candidate motif scores are based on a vector v where v[i=1...n] is equal to the number character positions where i sequences agree on a particular consensus character and where n is the number of sequences being compared. We defined a function f(v) that assigns a scalar score to each candidate motif based on vector v. We establish a history table to record the number of the same letters in one position of the patterns as a parameter of function f (2's, 3's).

If the at the same position, there are two characters are the same, then count 1 more for parameter 2's. In the same situation, if there are 3 characters are the same, then count 1 more for parameter 3's. We use the count of 2's and 3's to multiple by multi and to get the approximate

number of comparisons and then divide by the count of comparisons to get the score of the candidate motif patterns. And after we get all the scores of candidate motif patterns, we use a filter function to rank all the scores and select the best candidate for each comparison. The mutation rate is controlling the candidate motif patterns. Because the mutation rate equals to the number of mismatch of the comparison divide by the length of motif, if the result cross the limit, then it is not in our consideration.

When we get all the top scores of candidates, we use Approximate Voting Algorithm to recover the best motif pattern. We select the first one pattern as a comparable pattern and we use a parameter d to record the distance between comparable pattern and the object patterns. Since they are at the same length L, the output will be a pattern with length L and have the minimum distance to all the other patterns in the comparison. And that maybe theoretical motif we recover.

3.1 Main Functions

3.1.1 Score Function

In this function, it will calculate the parameters of 2's and 3's. And this function will be called in the three-sequence-comparison function.

Score(S1;S2;S3; 1's, 2's, 3's)

Input: Three Sequences and the count of comparisons of each position.

Output: The value of 1's, 2's and 3's.

Steps:

1. Add the same position of each sequence to a new list.

2. If the occurrence is the same, then make counter plus 1.

3. For each position, if counter is 2, then 2's plus 1, if counter is 3, then 3's plus 1.

4. Output the value of 1’s, 2’s and 3’s.

End of Score.

3.1.2 Three-sequence-comparison Function

In this function, the mainly idea is to run the three-sequence comparison and get some results, and calculate the score for each result. The output result is the parameters for each pattern and which will be recorded into a list. The variable bestscore contains the score list. And S1; S2; S3 are the three input sequences. For the Score function, the 1’s, 2’s and 3’s are the variables that contains the result from three sequences comparison. The detail will be introduced in following Score function part.

Three-Sequence-Comparison(S1;S2;S3; out bestscore)

Input: Three sequences

Output: A list of patterns with scores.

Steps:

1. Setup history table to contain the parameter for the calculation of the scores.

2. Use double for loop to do the three sequences comparison. Shift one position right of two object sequences each comparison.

3. Record the parameters to the history table from comparison.

4. Calculate the result for each pattern based on Score(S1;S2;S3; 1’s, 2’s, 3’s). Get the parameters for further calculation.

5. Add the result to the list bestscore.

6. Output the list.

End of Three-Sequence-Comparison function.

3.1.3 Distance Function

In this function, it will calculate the distance between two sequences.

diff(S1;S2)

Input: One comparable sequence and one ideal pattern. This function will be called in calculate multiplier function

Output: The value of distance.

Steps:

1. From the first position of sequences, if the characters are different, then counter c plus 1.

2. Go through step 1 from the next position to the end of the sequences.

3. Return c/L for each position

End of diff.

3.1.4 Calculate Multiplier Function

In this function, the mainly task is to calculate a multiplier. This multiplier will be used in the get top scoring motif function.

Calculate-Multiplier(S1;S2;S3)

Input: Three sequences

Output: The value of multiplier.

Steps:

1. Set length L consensus as an initial pattern

2. Call diff(S1;S2) to calculate the distance between three sequences and the initial consensus.

3. Use multi = $\sqrt{\prod_{n=1}^{5} diff(Si, Sj)} * 0.5 + 0.5$ to get the multiplier.

4. Output the multiplier.

End of Calculate-Multiplier.

3.1.5 Get Rank of Correct Diagonal

In this function, it will calculate the rank for each candidate motif pattern. The rank will be used in the get top scoring motif function.

Get-Rank-Of-Correct-Diagonal(bestscore)

Input: The score from three-sequence-comparison function

Output: The value of rank.

Steps:

1. If the score is better than the initial score, the rank plus 1.

2. Return rank.

End of Get-Rank-Of-Correct-Diagonal.

3.1.6 Get Top Scoring Motif Function

In this function, it will get the result from three-sequence-comparison, and get the top scoring candidate motif patterns.

Get-Top-Scoring-Motif(S1;S2;S3; list, out list 1, out s)

Input: Three sequences.

Output: A list of high rank candidate motifs and the rank.

Steps:

1. Call the function Three-Sequence-Comparison(S1;S2;S3;outbestscore)

2. Call the function Get-Rank-Of-Correct-Diagonal(bestscore) to get the rank for all the scores.

3. Call the function Calculate-Multiplier(S1;S2;S3) to get a multiplier for calculating score of candidate motif.

4. Output the high rank candidate motifs and the rank.

End of Get-Top-Scoring-Motif.

3.1.7 Run Test Function

In this function, it will run the comparison among five input sequences and get the final result of the comparison.

Run-Test(list sequence)

Input: A sequence list with five different sequences

Output: The best candidate motifs and the scores

Steps:

1. Setup the basic parameter as we mentioned before.

2. Calculate the result for ideal conditions. Using multiplier we calculated before multiple the f(2's, 3's) to get the result.

3. Get the top score candidate motif patterns we selected before by calling Get-Top-Scoring-Motifs(S[1], S[2], S[3], S, out score, out s).

4. Repeat step 3 to cover all 5 sequences in the list.

5. Output the best candidate motifs and the scores.

End of Run-Test.

3.1.8 Recover Function

In this function, it will collects all the patterns from Run-Test function and then use the voting algorithm to get one most possible motif pattern.

Voting(list pattern)

Input: A list of patterns T with length L.

Output: A best pattern t which has the minimum distance to all the patterns.

Steps:

1. Input a pattern set T with length L.

2. For each pattern of T, compare all the input patterns and for each position to vote one character that appears most.

3. The output is a new pattern t which has the minimum distance to all the patterns.

End of Voting.

3.2 Example

In this section, we will give an example to make the main idea of this method as clear as possible.

1. Set S is the set of sequences we want search, S = S1, S2, S3…., i, j, k are 3 integers (i≠ j≠ k). L is the length of motif and m is the length of sequence. In this example, all the sequences have the same length m. *bestscore* is the variable of scores for the comparisons. *c* is the counter of difference count.

2. First we randomly select 5 sequences as input sequences.

3. Then we select three sequences Si, Sj, Sk of these five sequences.

Si = ACCTTGTGTGCATTAGCGATAGCGGGGAATTT....

Sj = ATCTCGATCGGGTGTACATTAACTTTCGCCCC....

Sk = GTGTGCAATAAAACTTGCCTATACTCATACAA....

4. Start the comparison. First from Si[0] to the motif length Si[n-1], compare with Sj[0] to Sj[n-1] and Sk[0] to Sk[n-1].From these sequences we can see, on the first position S[0] is AAG. So the value of parameter 2's increases 1 because there are 2 same characters. And for S[3] which is the fourth position, it is TTT, so the value of parameter 3's increases 1. For the S[4] is TCG so the value of parameter 1's increases 1 because there are three different characters.

5. After first comparison, record the result into a list and then shift Sj right for 1 position and do the second comparison and after that shift Sk right for 1 position and do the third comparison. Compare the whole sequences like this till the end.

6. According to the score list, call function Get-Rank-Of-Correct-Diagonal (bestscore) to sort the scores in descending order. And the next step is calculating the multiplier.

multi=$\sqrt{\prod_{n=1}^{5} diff(Si, Sj)} * 0.5 + 0.5$ In this example, for the first 10 positions of Si and Sj ,the distance is 4. Since we have 5 input sequences, we get the multiplier for all of them.

7. Use the scores multiply the multiplier we just calculated, and get the top 10 final results to the output and with the patterns.

8. The final step is to recover the motif by using voting algorithm. On each position, which character appears most will be voted to the final result. In this example, if the ten patterns are:

GTGTGCATTA

GTGTACATTA

GTGTGCATTA

GTGTACATTA

GTGTGCATTA

GTGTACATTA

GTGTGCATTA

GTGTGCAATA

GTGTGCAATA

GTGTGCAATA

Then the final result should be GTGTGCATTA.

3.3 Parallel implementation – SOAP

Since the motif detection is dealing with large amount of data, the speed of the project is also very important factor. So a good way to speed up a program is to try the distribute system. And the core part of it is SOAP (Simple Object Access Protocol).

SOAP uses two widely used protocols: HTTP and XML. HTTP is used to achieve the transfer of SOAP-RPC style, and XML is its encoding scheme. A few lines of code and an XML parser and HTTP Server (MS IIS or Apache) immediately became the SOAP ORBS. SOAP communication protocol uses HTTP to send XML-formatted information. HTTP and RPC protocol is very similar; it is simple, widely deployed, and the role of firewalls more easily than other protocols. HTTP requests are typically handled by the Web server software (such as IIS and Apache), but more and more application server products is supporting HTTP, XML as a better network data representation (NDR). SOAP to the use of XML code into the request and response parameter encoding mode, and transmitted using HTTP. Specifically, a SOAP method can be viewed simply as HTTP requests and responses that follow the SOAP encoding rules, a SOAP Terminal can be seen as an HTTP-based URL is used to identify method calls.

CHAPTER IV

EXPERIMENT

In this section, we will introduce a series of experiments and the analysis of the results.

4.1 Algorithm Implementation

This experimental implementation is written in C#. In this implementation, I use 3-sequence comparisons instead of 2-sequence comparisons. Each time we pick 5 sequences to detect possible consensus. When we do the 3-sequence comparisons, we use a vector function to calculate the score of each consensus. Then I will use voting algorithm on the highest score consensuses to recover the possible motif. At last, put this implementation through a web server, and take advantage of distributed system to finish the job.

There are some experimental data: First is the artificial data experiment. We use the auto simulated data from the implementation program. All the sequences are automatically generated. We set the length of each sequences at 600 and the length of motif at 15. Each test we generate 5 sequences. Our test results show that this implementation is stable to detect the possible consensus. The most important thing is that it is similar to the ideal motif. Although some of the mutation is a little high, but we can see that the possible characters contain the same character as the ideal motif.

Tab 4.1. Artificial Data Result

	set 1	set 2	set 3	set 4	set 5
Mismatch positions per sequences	1.95	3.05	3.95	3.4	3.35
Time cost(s)	175	175	175	175	175

Then the test moves to the biological data. This part is some experiments on the real biological data. For the implementation test, we use the data from the Promoter Database of Saccharomyces Cerevisiae of Cold Spring Harbor. We select several different DNA sequences. They have different sequence length and motifs. For each experiment, the number of sequences

and the length of motif are not the same. We use the MEME suite, Gibbs, InfoGibbs and Consensus tools to test the same sequences to compare with our simulation. And the result shows that according to the different sequences we select, our implementation sometimes is a little better than the other tools on the mutation side. For example, when we run the BAS1 and GCR1 experiments, we find that the result by MEME suite has the higher mutation value than our implementation. That means when our simulation detects the BAS1 and GCR1 sequences, we have a more stable results. On the other hand, when we deal with RAP1, GCN4 and HSE sequences, our implementation has the higher or equal mutation value comparing with the MEME suite. According to all the experiments, we can see that this implementation performs better than any other tools in GCN4 motif detection. But in BAS1, it is the worst and others is in the average.

Tab 4.2. Total Number of Mismatch Positions

	Bas1	GCN4	GCR1	Rap1Ebf1	HSE-HTSF
Implementation	38	6	18	37	6
Gibbs	8	51	5	202	7
MEME	8	15	10	32	3
InfoGibbs	9	21	5	46	9
Consensus	8	9	5	42	7

Tab 4.3. Average Mismatch Numbers per Sequence

	Bas1	GCN4	GCR1	Rap1Ebf1	HSE-HTSF
Implementation	6.3	0.6	3	2.46	1.2
Gibbs	1.33	5.6	0.83	13.46	1.4
MEME	1.33	1.67	1.67	2.13	0.6
InfoGibbs	1.5	2.33	0.83	3.06	1.8
Consensus	1.33	1	0.83	2.8	1.4

4.2 Parallel Experiment

In theory, in order to complete 30 jobs, a single computer would need to execute 7 batches of 4 jobs and 1 batch of 2 jobs (because 1 * 7 *4 + 1 * 1 * 2 is equal to 30); three computers would each need to execute 2 batches of 4 jobs and 1 batch of 2 jobs (because 3 * 2 *4 + 3 * 1 * 2 is equal to 30); and 5 computers would each need to execute 1 batch of 4 jobs and 1 batch of 2 jobs (because 5 * 1 *4 + 5 * 1 * 2 is equal to 30). Thus, a single machine needs to execute 7 batches at 180 seconds each and 1 batch at 100 seconds, which would take 1360 seconds or 22.6 minutes; three machines would need to execute 2 batches at 180 seconds each and 1 batch at 100 seconds, which would take 460 seconds or 7.6 minutes; and finally, 5 machines would each need to execute 1 batch that takes 180 seconds and 1 batch that takes 100 seconds, which would take 280 seconds or 4.6 minutes in all.

In short, the expected times would be 22.6, 7.6, and 4.6 minutes for jobs that execute in 1, 3, and 5 machines, respectively. The results of 7.6 and 4.6 minutes are within reasonable agreement with our experimental results. The result of 22.6 minutes, however, is not in so much agreement with our experiment according to the results. Although the time expected from a single machine is considerably greater than for the other setups, our experimental time for a single machine is much greater than expected (around twice as much).

This discrepancy might be due to the time required to exchange data between the server and the client. Since a single machine would need to contact the server much more often than multiple machines would, the effect of all these communication times is much more pronounced for a single machine. Furthermore, we did not optimize our system to be fast with respect to communication time because we expected CPU time to be much greater than networking time. Finally, the effect of synchronous communication might have also had a penalty on our implementation; particularly in the case when single machine does all of the work.

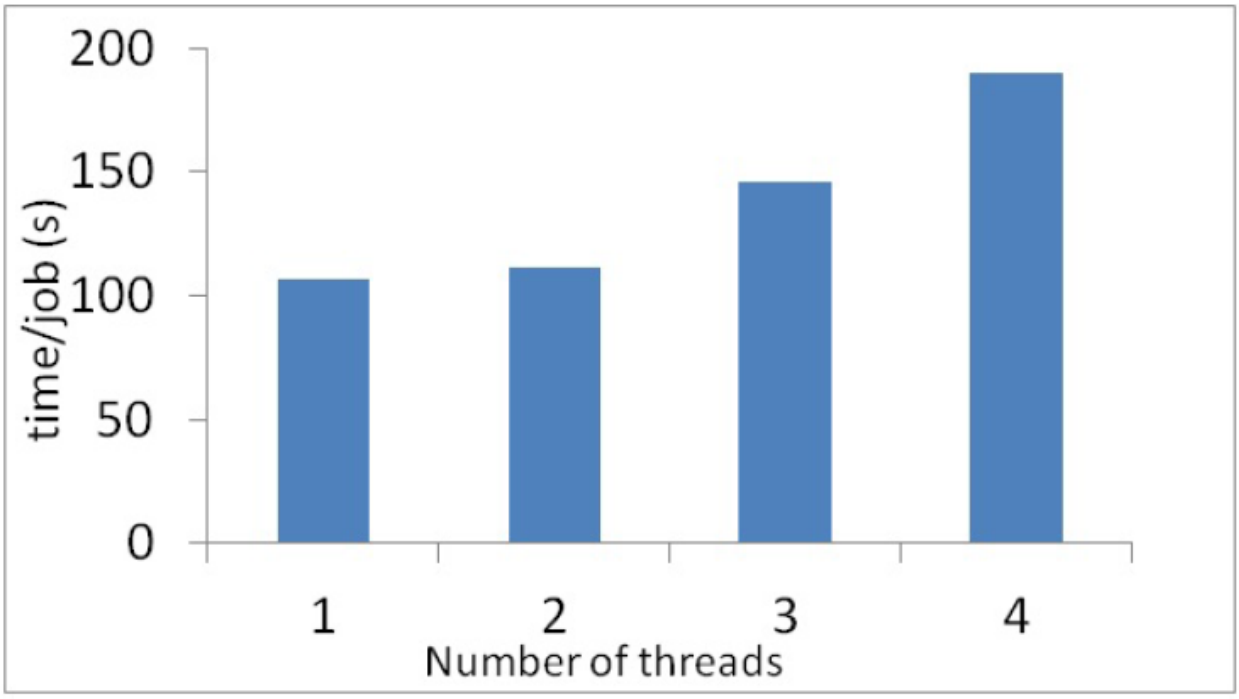

Fig 4.1. The number of threads working at the same time for how many seconds to finish one job

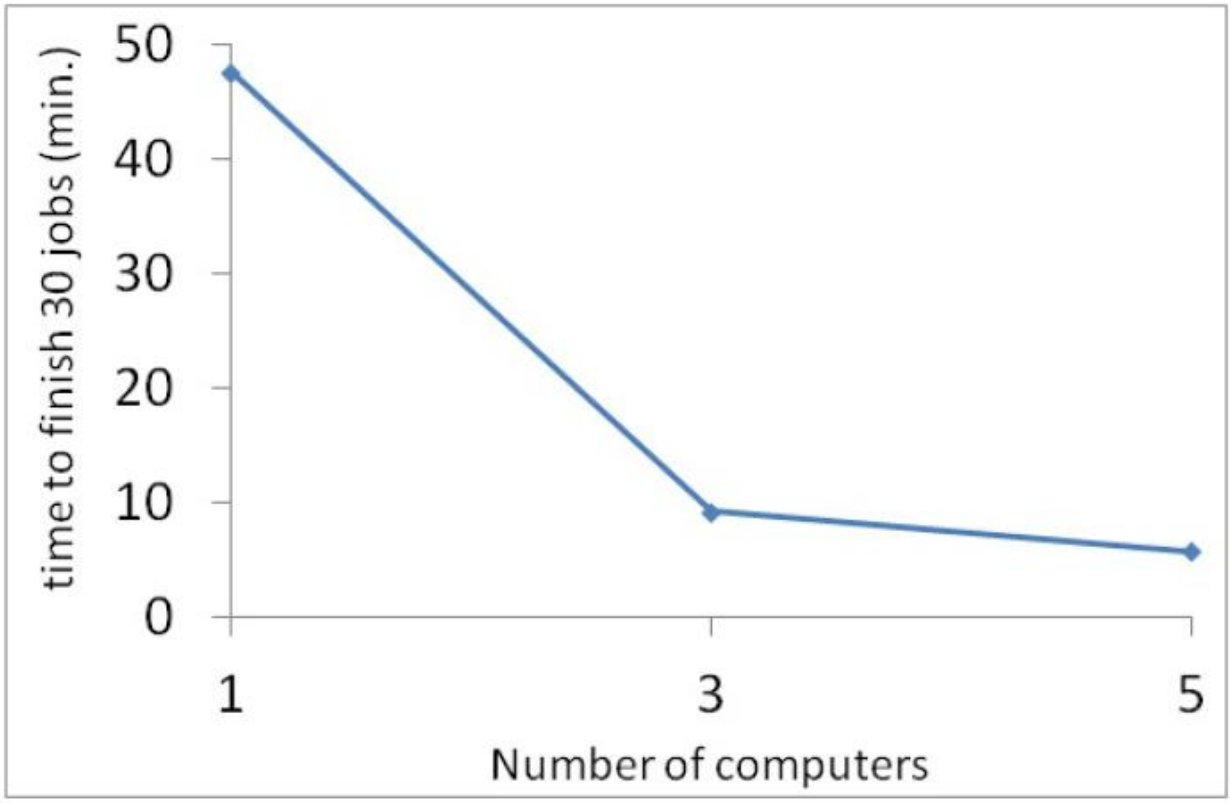

Fig 4.2. The number of computers working at the same time for time to finish 30 jobs

CHAPTER V

CONCLUSION AND FUTURE WORK

In this section, we will give a brief conclusion and an outlook for the future development of DNA motif detection algorithms.

In this project, according to the experiment results, using the Monte Carlo approach can handle the small number of input sequences data very well and by the help of voting algorithm, the result shows pretty good.

In this experiment, we successfully implement the three sequence comparison to detect motif patterns. In future, we hope we can implement four-sequence or even more comparison because according to the other research, the larger number sequence comparison can handle a larger mutation rate. This may make sure the accuracy of the result. But the meanwhile, it will increase the time complexity. That's why it should use like distributed system or other technology to speed up the detection.

We hope with the powerful database technology nowadays, detecting motif would have a bright future by using good from databases. And the network can be another helper for the problem. What worth looking forward is it may take advantage of databases and network, to make the problem be solved as accurate and fast as possible.

REFERENCES

[1] Timothy L. Bailey, Nadya Williams, Chris Misleh, and Wilfred W. Li. MEME: *Discovering and analyzing DNA and protein sequence motifs*. Nucleic acids research, 34(Web Server issue):W369–W373, 2006.

[2] Xin Chen and Tao Jiang. *An improved gibbs sampling method for motif discovery via sequence weighting*. Computational Systems Bioinformatics Conference, pages 239–247, 2006.

[3] ModanK Das and Ho-Kwok Dai. *A survey of dna motif finding algorithms*. BMC Bioinformatics, 8:S7–S21, 2007.

[4] Patrik D'haeseleer. *How does DNA sequence motif discovery work?* Nature Biotechnology, 24(8):959–961, 2006.

[5] Cathryn M. Gould, Francesca Diella, Allegra Via, P ̈al Puntervoll, Christine Gem ̈und, Sophie Chabanis-Davidson, Sushama Michael, Ahmed Sayadi, Jan Christian C. Bryne, Claudia Chica, Markus Seiler, Norman E. Davey, Niall Haslam, Robert J. Weatheritt, Aidan Budd, Tim Hughes, Jakub Pas, Leszek Rychlewski, Gilles Trav ́e, Rein Aasland, Manuela Helmer Citterich, Rune Linding, and Toby J. Gibson. *ELM: the status of the 2010 eukaryotic linear motif resource*. Nucleic acids research, 38(Database issue):D167–D180, 2010.

[6] O.R. Homann and A.D. Johnson. *Mochiview: versatile software for genome browsing and dna motif analysis*. BMC Biol, 8, 2010.

[7] J. Hu, B. Li, and D. Kihara. *Limitations and potentials of current motif discovery algorithms*. Nucleic Acids Res, 33(15):4899–913+, 2005.

[8] Lokesh Kumar, Andrew Breakspear, Corby Kistler, Li J. Ma, and Xiaohui Xie. *Systematic discovery of regulatory motifs in fusarium graminearum by comparing four fusarium genomes.* BMC Genomics, 11(1):208+, 2010.

[9] Erik Larsson, Per Lindahl, and Petter Mostad. *HeliCis: a DNA motif discovery tool for colocalized motif pairs with periodic spacing*. BMC bioinformatics, 8:418+, 2007.

[10] C. E. Lawrence, S. F. Altschul, M. S. Boguski, J. S. Liu, A. F. Neuwald, and J. C. Wootton. *Detecting subtle sequence signals: a gibbs sampling strategy for multiple alignment*. Science (New York, N.Y.), 262(5131):208–214, 1993.

[11] Charles E. Lawrence and Andrew A. Reilly. *An expectation maximization (em) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences*. Proteins: Structure, Function, and Bioinformatics, 7:41–51, 1990.

[12] Corby Kistler2 Li-Jun Ma Xiaohui Xie Lokesh Kumar, Andrew Breakspear. *Systematic discovery of regulatory motifs in fusarium graminearum by comparing four fusarium genomes*. BMC Genomics, 11:208–220, 2010.

[13] Andrew F. Neuwald, Jun S. Liu, and Charles E. *Gibbs motif sampling: detection of bacterial outer membrane protein repeats*. Protein Science, 4:1618–1632, 1995.

[14] Katherine Romer, Guy R. Kayombya, and Ernest Fraenkel. *WebMOTIFS: automated discovery, filtering and scoring of DNA sequence motifs using multiple programs and Bayesian approaches*. Nucleic Acids Res, 2007.

[15] Geir Kjetil K. Sandve, Osman Abul, and Finn Drabløs. *Compo: composite motif discovery using discrete models*. BMC bioinformatics, 9(1), 2008.

[16] Mathieu Blanchette Saurabh Sinha and Martin Tompa. *Phyme: A probabilistic algorithm for finding motifs in sets of orthologous sequences*. BMC Bioinformatics, 5:170, 2004.

[17] Lusheng Wang, Xiaowen Liu and Bin Ma. *Voting algorithm for the motif finding problem*. Compute Syst Bioinformatics Conf, pages 37–47, 2008.

[18] Xiaohui Xie, Jun Lu, E. J. Kulbokas, Todd R. Golub, Vamsi Mootha, Kerstin Lindblad-Toh, Eric S. Lander, and Manolis Kellis. *Systematic discovery of regulatory motifs in human promoters and 3 UTRs by comparison of several mammals*. Nature, 434(7031):338–345, 2005.

BIOGRAPHICAL SKETCH

Ran Ding received the Bachelor of Science degree in Computer Science from the Shenyang Ligong University of China in 2009 before moving to Texas, USA to pursue Master degree of Computer Science. He started his graduate studies in January, 2010. After two years studies, he decided to looking for a application job in computer science. His address is 1809 West Schunior Street, Apartment 111, Edinburg, Texas.

His professional interests include design and analysis of algorithms, database and bioinformatics.