A parallel simulation of the evolution of transcription factor binding sites

Robert Forder (rforder1@umbc.edu)

Department of Mathematics and Statistics, University of Maryland, Baltimore County

May 27, 2011

Abstract

The analysis of transcription factor binding motifs may aid in understanding the process by which transcription factors recognize their binding sites. We wish to investigate the likelihood that transcription factors use correlation between positions in potential binding sites as a criteria for recognition. We implement a genetic algorithm in parallel using a simple server-client organization to simulate the evolution of these binding sites. We then evaluate the performance of this application and conclude that exhibits excellent speedup and efficiency.

1 Introduction

A transcription factor is a specific type of DNA binding protein. The binding motif of a transcription factor is the set of all DNA sequences to which the transcription factor is known to bind. The process by which a transcription factor recognizes its binding sites is currently unknown. Through analysis of a binding motif, we are able to infer information about this process. We use information theory to quantify certain properties of a binding motif, namely its information content. We further divide this information content into two categories: information that arises as the result of correlations between positions in binding sites and information content of each position individually. Correlation information is the result of dependencies between columns. We develop a simulation in order to investigate the evolution of transcription factor binding sites and the likelihood that a transcription factor may utilize correlation information when recognizing binding sites. We then attempt to efficiently parallelize this algorithm in order to accelerate research.

We utilize the definition of information originally proposed by Shannon [2]. The application of Shannon entropy to the study of DNA binding sites was originally proposed by Schneider et al. [1] Schneider applies Shannon entropy to DNA binding sites under the assumption that transcription factors are not able to recognize correlation information in binding sites.

We implement a genetic algorithm to simulate the evolution of transcription factor binding sites and examine the manner in which binding sites evolve under various conditions by supplying

Table 1.1: Wall-clock execution time in MM:SS of varying numbers of unique parameter sets N on $p = 1, 2, 4, 8, 16$ and 32 client processes.

N	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$
32	22:51	11:27	5:56	2:57	1:31	0:51
64	45:16	23:23	11:53	6:05	3:08	1:33
128	93:53	45:39	23:29	12:06	6:11	3:04
256	183:10	92:38	46:59	23:47	12:30	6:14

various parameters to the simulation. We parallelize the algorithm using a simple server-client structure. Each client process runs an independent simulation. The server process exists to efficiently distribute parameters among the client processes.

Our use of parallelism results in greatly decreased execution time overall for the parameter studies performed. Table 1.1 depicts the total execution time for parameter studies utilizing N unique parameter sets over p client processes. For $N = 256$, the total execution time for the parameter study in serial requires over three hours. On 32 client processes, this time is reduced to just over six minutes.

This paper contains four sections. Section 2 introduces the application of information theory to transcription factor binding sites in more detail and rephrases the problem within this context. Section 3 explains how a genetic algorithm may answer this question, how it is implemented, and how it is parallelized. Finally, Section 4 enumerates the results of a performance study on the parallel algorithm. There is also a short appendix attached to this report which describes in detail the formulas used to arrive at entropy values discussed in Section 2.

2 Problem

The mechanism by which transcription factors recognize specific areas of a genome as binding sites is poorly understood. We wish to determine the likelihood that transcription factors use correlation between positions within potential binding sites as a criteria for recognition.

We adopt the information theory model created by Shannon [2] and applied to DNA binding sites by Schneider [1], to quantify the information content of a transcription factor binding motif. A more detailed discussion of Shannon entropy can be found in Appendix A. We consider two forms of information. The first form, which Schneider refers to as $R_{sequence}$, is positionally independent, column-wise information which we obtain by examining each column independently and summing the information content of each.

We define the information content of a single column in the following way. Suppose we choose a point arbitrarily in the genome. Let H_g denote our uncertainty regarding the base which occupies that point. Now let H_x denote our uncertainty after we learn that our transcription factor recognizes this point as position x in a binding site. Assuming that $H_g \geq H_x$, our uncertainty has decreased. The decrease in uncertainty that we experience, or information gain, is

$$R_x = H_g - H_x \text{ bits.} \tag{2.1}$$

The correlation between the columns in a binding motif, known as mutual information, is also a possible source of information. Mutual information represents the dependence of the content of one column in a motif upon the content of another. The mutual information between two columns x and y is

$$I_{xy} = H_x + H_y - H_{xy} \text{ bits} \tag{2.2}$$

and can be thought of as the correlation between two columns. For our purposes we restrict the number of columns in a binding motif to two. We describe these columns as x and y, as their ordering is irrelevant. Thus, we define the total information content of a binding motif I_{total} as

$$I_{total} = R_{sequence} + I_{xy} \text{ bits} \tag{2.3}$$

where $R_{sequence} = R_x + R_y$ represents the contribution of independent, column-wise information and I_{xy} represents the contribution of correlation between the two columns to the total information content of the binding motif. We begin with the assumption that transcription factors can, and do, use $R_{sequence}$ to recognize binding sites. This assumption is supported by current experimental evidence. We are interested in whether or not it is possible for a transcription factor to evolve which uses I_{xy} to recognize binding sites.

3 Method

We use a genetic algorithm to simulate the evolution of transcription factor binding sites. Genetic algorithms, generally speaking, are an optimization tool. Our genetic algorithm acts on a population of artificial organisms, each of which contains a binding motif. An iteration of the genetic algorithm is called a generation. A generation has three stages. First, we apply a fixed number of point mutations to the binding motif of each organism in the population. Then, we evaluate the fitness of each organism in the simulated population and select parents based on the finesses of those organisms. Finally, we replace the parent generation with their offspring.

We are interested in running a large number of these simulations with varying parameters. The total execution time for all of these simulations will become vary large. We parallelize the algorithm to reduce the total run-time, thus allowing us to perform more complex parameter studies in a shorter period of time.

3.1 The genetic algorithm

The genetic algorithm has three primary components: the mutation rate, the fitness function and the parent selection algorithm.

At the beginning of each generation, m positions are selected from the binding motif of each organism at random and the bases at those positions are randomly altered. This serves to simulate the biological process of mutation and serves as the driving force behind the genetic algorithm.

When this is complete, we calculate the fitness of each organism. The fitness function is used to determine the quality of an organism. We define the fitness of an organism as the effectiveness by which the transcription factor recognizes the sites within that organism's binding motif. This is determined by how closely the total information I_{total} of the organism's binding motif approximates a target value I_{target}. The target value is provided to the simulator as a parameter. Thus, we define the fitness of organism x as

$$f(x) = |I_{total}(x) - I_{target}| \tag{3.1}$$

where x is an organism in the simulated population and $I_{total}(x)$ is the total information content of the binding motif of x.

Parent selection is then conducted using a tournament selection strategy. For each organism in the population, we select k organisms randomly. The organism with the greatest fitness out of those k is chosen to be a parent. The number of offspring which an organism produces is equal to the number of tournament rounds it wins. There are as many tournament rounds as there are organisms in the population, so the total population size remains constant from one generation to the next. The parent generation is then completely replaced by the offspring. Note that k serves to modulate the selection pressure on the population. Higher values of k reduce the likelihood that weaker organisms are able to reproduce by increasing the likelihood that they be paired against a strong organism.

The completion of these steps concludes a single generation. Thus, there are three main parameters which influence the behavior of our genetic algorithm: the mutation rate m, the target information value I_{target}, and the selection pressure k. In addition to these parameters, we are able to alter the size of our population and the number of sequences in the binding motif of each organism, though the influence of these parameters is less pronounced. As the genetic algorithm executes, we are primarily interested how much of the total information per organism is composed of mutual information and how much is composed of independent, column-wise information. That is to say, we are interested in the average values of $R_x + R_y$ and I_{xy} for each organism in the population over the course of the simulation.

Table 3.2: Wall-clock execution time in MM:SS of varying numbers of unique parameter sets N on $p = 1, 2, 4, 8, 16$ and 32 client processes.

N	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$
32	22:51	11:27	5:56	2:57	1:31	0:51
64	45:16	23:23	11:53	6:05	3:08	1:33
128	93:53	45:39	23:29	12:06	6:11	3:04
256	183:10	92:38	46:59	23:47	12:30	6:14

Table 3.3: Speedup of parallel algorithm for $N = 32, 64, 128$ and 256 on $p = 1, 2, 4, 8, 16$ and 32 client processes.

N	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$
32	1.00	2.00	3.84	7.75	15.07	26.88
64	1.00	1.94	3.81	7.44	14.44	29.20
128	1.00	2.06	4.00	7.76	15.18	30.61
256	1.00	1.98	3.90	7.70	14.65	31.67

Table 3.4: Efficiency of parallel algorithm for $N = 32, 64, 128$ and 256 on $p = 1, 2, 4, 8, 16$ and 32 client processes.

N	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$
32	1.00	1.00	0.96	0.97	0.94	0.84
64	1.00	0.97	0.95	0.93	0.90	0.91
128	1.00	1.03	1.00	0.97	0.95	0.96
256	1.00	0.99	0.97	0.96	0.92	0.99

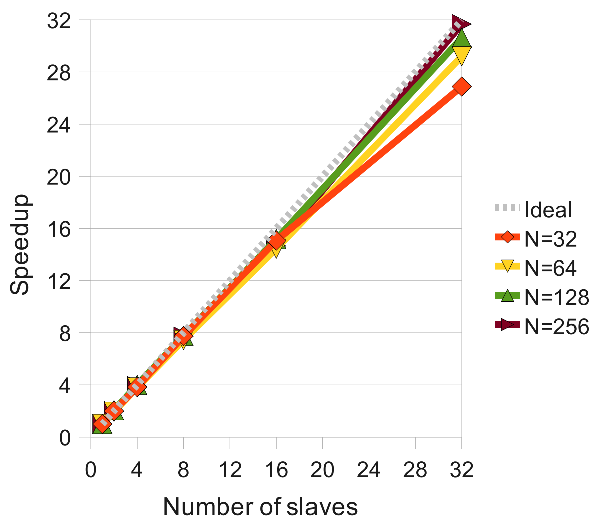
3.2 Parallelizing the genetic algorithm

The average values of $R_x + R_y$ and I_{xy} for each organism in the population over the course of the simulation is dependent upon the parameters m, I_{target}, and k. In order to understand the influence of these parameters on the simulation, a variety of parameter combinations must be evaluated, requiring the execution of many simulations. The total execution time of these simulations becomes large as the number of unique parameter combinations we execute increases. For this reason, we choose to parallelize the genetic algorithm.

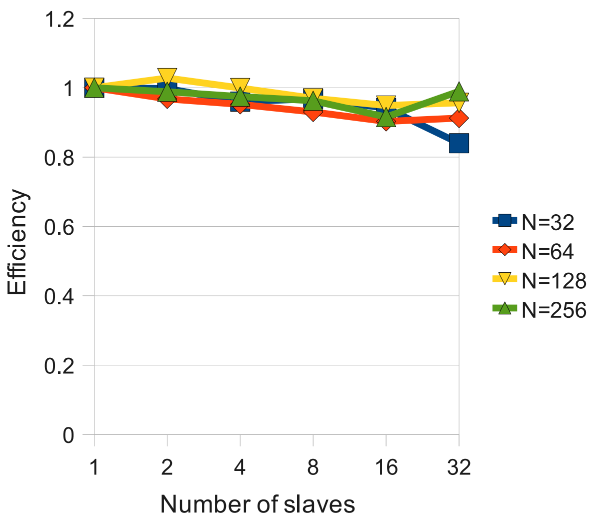
We adopt a server-client model using point-to-point MPI communication to distribute parameter sets among processes. Process 0 acts as the server process. All other processes are client processes. We will use p to denote the number of client processes. In total, there are $p + 1$ processes, of which p process are clients. The server process begins by loading a master list of parameter sets from the disk. It then awaits messages from client processes by issuing a call to the `MPI_Recv` function, specifying a source of `MPI_ANY_SOURCE`.

Each client begins by issuing a ready command to the server by calling `MPI_Send`. When the server process receives a ready command, it determines the source of the command by examining the content of the `MPI_Status` structure. It then proceeds to send a parameter set from the master list to the client which issued the ready command via the `MPI_Send` function. Each parameter set in the master list is executed exactly once. As a client executes the simulation, it writes the average mutual information and column-wise information per organism to the disk along with the parameter set which was executed. When a client finishes executing a simulation, it sends another ready command to the server.

If the server process exhausts the entries in the master list, then the server responds to all ready commands with a kill command. A kill command is a dummy parameter set with a population size of zero. If the client receives a kill command, it terminates. Once the server has sent a kill



(a) Observed speedup S_p.



(b) Observed efficiency E_p.

Figure 4.1: Performance in parallel.

command to all of the client processes, the server terminates.

We expect this strategy to provide the greatest performance benefit when the number of parameter sets is large. In particular, we expect to see the greatest benefit when the number of parameter sets greatly exceeds the number of client processes. Clearly, there is no benefit to having more clients than parameter sets, as the excess clients will be idle.

4 Results

We execute the parallel algorithm using N unique parameter sets. For each parameter set, the population size is 100 organisms, there are 100 sequences in each binding motif, and the genetic algorithm runs for 5,000 generations. For each N, we vary m, k, and I_{total} in order to produce N unique parameter sets.

We rely on two measures to evaluate the performance of the parallel algorithm. Let T_p denote the execution-time of the parallel algorithm over p client processes. We define the speedup on p processes as

$$S_p = \frac{T_1}{T_p}.$$

Ideally $S_p = p$, which is to say that if we execute the algorithm over p client processes, then it should run p times as fast. This is not possible due to overhead, but we wish to approximate it as closely as possible. We also consider efficiency, which we define as

$$E_p = \frac{S_p}{p}$$

which compares the speedup of p client processes to the ideal. Table 3.3 exhibits the speedup of the parallel algorithm for $p = 1, 2, 4, 8, 16, 32$ as derived from the execution time provided by Table 1.1. Table 3.4 exhibits the efficiency of the parallel algorithm for $p = 1, 2, 4, 8, 16, 32$ as derived from Table 3.3. We use N to denote the number of entries in the master list, which is the number of unique parameter sets which are to be executed. Figure 4.1(a) is a plot of Table 3.3 and Figure 4.1(b) is a plot of Table 3.4.

We notice several trends in the performance of the parallel algorithm over varying N and p. Perhaps obviously, the run-time of the algorithm decreases as p increases. This is our first indication that the parallelization of the algorithm was successful. We notice in Figure 4.1(a) that S_p generally approximates the ideal, particularly for $N = 256$. This suggests that our implementation scales effectively. We see this displayed again in Figure 4.1(b), though we notice more clearly that E_p does in fact degrade for larger values of p. For smaller values of p, S_p remains virtually constant over varying N. As p increases, S_p exhibits less stability. Compare the values of S_p in the $p = 4$ and $p = 32$ columns of Table 3.3. When observing 3.3, we see that $N = 256$ is nearly ideal, while speedup appears to degrade for lower values of N.

Acknowledgements

Many thanks are due to Dr. Matthias Gobbert of the Department of Mathematics and Statistics at UMBC and Dr. Ivan Erill of the Department of Biological Sciences at UMBC, who have served as research mentors. To Joe Cornish with whom this project is collaborative as part of Interdisciplinary Training for Undergraduates in Biological and Mathematical Sciences (UBM@UMBC `www.umbc.edu/ubm`). To Rick Wagner, who provided a `C++` class which is used by the simulation to load and parse configuration files [3], and to everyone else who assisted in the completion of this report.

The hardware used in the computational studies is part of the UMBC High Performance Computing Facility (HPCF). The facility is supported by the U.S. National Science Foundation through the MRI program (grant no. CNS–0821258) and the SCREMS program (grant no. DMS–0821311), with additional substantial support from the University of Maryland, Baltimore County (UMBC). See `www.umbc.edu/hpcf` for more information on HPCF and the projects using its resources.

References

[1] T. D. Schneider and G. D. Stormo and L. Gold and A. Ehrenfeucht, *Information content of binding sites on nucleotide sequences* Journal of Molecular Biology, vol. 188, pp. 415-431, 1986.

[2] C. E. Shannon, *A mathematical theory of communication*, Bell System Technical Journal, vol. 27, pp. 379-423 & pp. 623-656, 1948.

[3] R. Wagner, Home page. 26 May 2004. retreived 20 April 2011. `http://www-personal.umich.edu/~wagnerr/ConfigFile.html`

A Entropy

Here we provide the definitions of several forms of entropy which have been provided to us by Shannon. As is common, we use H to denote some form of entropy. Let $B = \{A, T, C, G\}$, where A, T, C, and G are abreviations for adenine, cytosine, guanine and thymine, respectively. Given some genome, let $P(m)$ be equal to the frequency with which a base $m \in B$ appears in that genome. Our uncertainty concerning which base occupies a specific position in the genome is

$$H_g = \sum_{m \in B} P(m) \log_2(P(m)) \text{ bits.} \tag{A.1}$$

We refer to this as the genomic entropy. For our purposes, we assume $P(x) = 0.25$, which is to say that bases are distributed equiprobably throughout the genome. Thus, $H_g = 2$ is fixed for our simulations.

Given the binding motif of a transcription factor composed of sequences within this genome, let $P_x(m)$ be equal to the probability that the position x of a random sequence within that motif is occupied by some base $m \in B$. We define the entropy of column x of the binding motif as

$$H_x = \sum_{m \in B} P_x(m) \log_2(P_x(m)) \text{ bits.} \tag{A.2}$$

Finally, we define the joint entropy between two columns in a binding motif. If $P_{xy}(m, n)$ is the probability that $m \in B$ appears in column x and $n \in B$ appears in column y in the same sequence, then

$$H_{xy} = \sum_{m,n \in B} P_{xy}(m, n) \log_2(P_{xy}(m, n)) \text{ bits} \tag{A.3}$$

is the joint entropy between column x and column y.