

Motif Detection in Yeast

ICS 175A Winter 2004

Vishakh (vishakh@uci.edu)

Joe Bertolami (joe@bertolami.com)

Nick Urrea (nurrea@uci.edu)

Jeff Weiss (jmweiss@uci.edu)

Introduction

The purpose of our project is to find regulatory sequences in the upstream DNA of yeast. Regulatory sequences are segments of DNA where proteins can bind to enhance transcription of a gene. Our data is drawn from the upstream region of the yeast genome. The genome consists of gene families, which in turned consist of individual genes. The genes themselves are long strings of the characters A, T C and G. Our program attempts to find substrings that are unusually frequent in gene families given their distribution in the whole upstream genome. Furthermore, we account for variations in the structure of these regulatory sequences by constructing probability matrices for each of our candidate regulatory sequences. Our method is based on the work of Jacques van Helden, which is described in his 1998 paper.

We have collaborated to create a program implemented in C++ for this project. It currently produces results consistent with those of van Helden's own program and successfully generates probability matrices consistent with the current dataset. It is also equipped with a graphical front end that allows users to intuitively generate results for various data sets.

Motivation

Organisms such as yeast, although unicellular, share many genes with humans. As a result, they share many genetic diseases too. Gaining an understanding of gene regulation in yeast would make the corresponding task in humans much easier. Finding regulatory sequences in yeast, then, might lead to advances in medical science. This includes genetic therapies for diseases like cystic fibrosis.

History

The previous century saw tremendous advances in genetics, especially after the discovery of the double helical structure of DNA by Watson and Crick. The scientific community has since been trying to gain a better understanding of how genes function. Ventures like the Human Genome Project have ushered in a new era of discovery and scientific advancement. The specific approach that we use in this project was developed by Jacques van Helden in 1998. He has since put his data and program on the Internet. He has also published several related papers subsequently.

Our Approach

We created an application using C++ and Win32 based on the methods enumerated in van Helden's paper. We examined other programming languages such as Java and Perl but decided to use C++ because of its speed and optimization capabilities. C++ allowed us to efficiently manipulate very large data sets in short amounts of time.

Our program first goes through the entire upstream genome and records the frequencies of all substrings of sizes 6-8. We restrict the size of the recorded substrings since, due to biological reasons, regulatory sequences in yeast can be found within this range. This information is stored in a hash table (henceforth called Table A). It then runs through the genomic data for the gene family in consideration and processes it similarly into another table (henceforth called Table B).

We use a high-speed file reader that takes in genomic data files in FASTA format and extracts the pertinent information from them. The reader utilizes a custom string library to provide a high level of abstraction from the low-level memory details of our data. This reader provides a single unified interface to the rest of the application for optimized loading and manipulation of gene files.

The tables are constructed using a custom hash table. A typical table entry is shown below. The hash table is implemented in the standard method, and each position in the hash table contains the information for a unique genomic substring. Our selection of the hash function had to consider the fact that the values to be hashed will most likely all be the same length and only have 4 possible characters at each position. We developed a hash function that would not produce many hash collisions given these limitations of possible inputs.

Substring	Frequency	Score 1	Score 2	Score 3
AGAGAT	8	3	0.24	0.23

Figure 1: Sample Output of GeneGui

The program then goes through each entry in Table B and assigns three scores. The first score is based simply on the formula:

$$\text{score1} = \text{Expected occurrence} / \text{Actual occurrence}$$

Where:

- Expected Occurrence = Frequency in Table A / size of upstream genome * size of gene family
- Actual Occurrence is the frequency of the substring in Table B

Figure 2: Expected/Actual Formula

This gives a rough estimate of how surprisingly frequent a substring is. As can be seen in the results section, this doesn't turn out to be too effective. The second score is calculated using the Binomial distribution:

$$\text{score2} = C(n, k) * p^k * (1-p)^{(n-k)}$$

Where:

- N is the size of the gene family
- K is the frequency of the current substring in Table B
- P is the probability of the current substring occurring at any position in Table A

Figure 3: Binomial Distribution

Score2 effectively gives us the odds of a substring having occurred K times in the gene family, given its frequency in the entire upstream genome. The lower the odds, the more surprising it is that the current substring has that particular frequency. However, low probability of occurrence of a substring doesn't by itself imply that it is unusually frequent. For this reason, we calculate the probability of a substring occurring not just 'k' times, but at 'k' times or more.

Similar to score2, the third score is calculated using the Poisson distribution:

$$\text{score3} = (e^{-\lambda} * \lambda^k) / k!$$

Where:

- E is the constant 2.71828182846
- K is the frequency of the current substring in Table B
- λ is the probability of the current substring occurring at any position in Table A

Figure 4: Poisson Distribution

The program then sorts the table using one of the scores, generally the Binomial score. The top ten substrings thus obtained are then used to generate the probability matrices.

We encountered a major problem in calculating the above scores. In the case of the Binomial score, numbers on the order of 800! needed to be calculated. This posed a storage problem because of the large numbers involved. We circumvented this by using logarithms of both sides of our equation. The logarithm of a factorial is much easier to calculate using the formula:

$$\log n! = \log(n) + \log(n-1) + \log(n-2) + \dots + \log(2) + \log(1)$$

Figure 5: Calculating the logarithm of a factorial

The binomial score equation could then be rewritten and calculated with ease:

$$\log(\text{score2}) = \log[C(n, k)] + k * \log(p) + (n-k) \log(1-p)$$

Figure 6: Reformulating the Binomial score equation using logarithms

The program then uses the top ten substrings as seeds to create probability matrices. A probability matrix gives the probability of each letter occurring at each position for a given substring. An example of a probability matrix using ATGC as its seed would be:

	Position 1	Position 2	Position 3	Position 4
A	0.7	0.1	0.1	0.1
T	0.1	0.7	0.1	0.1
G	0.1	0.1	0.7	0.1
C	0.1	0.1	0.1	0.7

Figure 4: A probability matrix with seed ATGC

Notice that the letters are assigned probabilities of 0.1 even though they don't actually occur at some positions. This is done in order to eliminate a statistical bias since it would be imprecise to completely rule out any letter appearing in a given position.

After creating probability matrices with the top substrings as seeds, we refined them to more accurately reflect the probability distributions in the gene family. For each matrix, we found the best substring match in each member gene of the gene family. Substrings were scored by simply adding up probabilities. For example, for the above matrix and the substring AAGG, the score would be $0.7 + 0.1 + 0.7 + 0.1$, i.e. 1.6. The highest scoring substring in each gene of the current gene family was then used to generate an updated matrix. The probability matrix would be computed by simply taking the "average" of the best matches. For example, if the best matches were AATG, AGTG and GGGG, the new matrix (without correction) would be:

	Position 1	Position 2	Position 3	Position 4
A	0.6	0.3	0.0	0.0
T	0.0	0.0	0.6	0.0
G	0.3	0.6	0.3	1.0
C	0.0	0.0	0.0	0.0

Figure 4: Updated probability matrix

The program iterates this process for each matrix until it stabilizes, i.e. the values stop changing significantly. The final probability matrix gives a probability distribution for a possible regulatory sequence.

Evaluation

For evaluating the results generated by our program, we first considered comparing them to van Helden's as described in his paper. However, we soon realized that he used an outdated data set for his paper and had to look elsewhere. Fortunately, we could obtain more accurate results based on more current data from his site (<http://rsat.ulb.ac.be/rsat/>). This became the primary criterion by which we measured the accuracy of our program.

We also wanted to find out which of the three scoring methods utilized is the most accurate. We sorted the entries in our tables by each of the three scores and examined how they fared. Of course, even if our results tallied with those of van Helden, it would not mean they are biologically correct. It is conceivable that there is a fundamental flaw in van Helden's data or methodology, and hence, also in ours. However, after some discussion we concluded that van Helden's was the most substantive data source to which we had access. We also concluded that

this source provides data that we could pragmatically assume is consistent with that of current biology.

Results

The results generated by our program are shown here. For each gene family mentioned in the van Helden paper, we show the top ten substrings ranked by van Helden's site. For each substring, we show the rank it obtained using the three scoring methods. For instance, "CACGTG" is ranked the highest by van Helden's site, also using the binomial score and third and fourth for the other two scores in the MET family. We also present the corresponding ranks from van Helden's paper for the MET family to demonstrate that he was using an older dataset. His own site now generates results that contradict his paper.

MET family:

String	V.H. Site	Binomial	Poisson	Expected/Actual	V.H. Paper
CACGTG	1	1	3	4	1
ACGTGA	2	2	1	2	3
TCACGT	3	3	2	1	2
ATATAT	4	4	N/A	N/A	5
TATATA	5	5	N/A	N/A	10
AACTGT	6	6	4	28	4
ACAGTT	7	7	N/A	29	N/A
ACACAC	8	8	7	N/A	N/A
GTGTGT	9	9	6	N/A	N/A

NIT family:

String	V.H. site	Binomial	Poisson	Expected/Actual
CTTATC	1	1	1	8
GATAAG	2	2	2	7
ATAAGA	3	7	7	N/A
TCTTAT	4	8	8	N/A
CGATAA	5	9	9	N/A
TTATCG	6	10	10	54
ACATCT	7	12	12	N/A

AGATGT	8	13	13	N/A
AACATC	9	14	14	82

PHO family:

String	V.H. site	Binomial	Poisson	Expected/Actual
TGCCAA	1	1	1	14
TTGGCA	2	2	2	13
AAACGT	3	3	3	24
ACGTTT	4	4	4	23
GCACGT	X	5	5	8
ACGTGC	X	6	6	7
CCCACG	X	7	7	1
CGTGGG	X	8	8	2
ACGTAT	X	9	9	N/A

PDR Family:

String	V.H. site	Binomial	Poisson	Expected/Actual
CCACGG	1	1	1	1
CCGTGG	2	2	2	2
CGTGGA	3	3	3	6
TCCACG	4	4	4	5
CACGGA	5	5	5	9
TCCGTG	6	6	6	10
CCGCAG	7	7	7	8
CTGCCG	8	8	8	7
CCGCCG	9	9	9	20

GAL Family:

String	Binomial	Poisson	Expected/Actual
TCGGAG	1	2	15
CTCCGA	2	1	88
GATCAC	3	4	11

GTGATC	4	3	45
ATGTCT	5	6	21
AGACAT	6	5	48
TCCGAA	7	8	48
TTCGGA	8	7	78
TACCCC	9	9	77

GCN Family:

String	V.H. site	Binomial	Poisson	Expected/Actual
GAGTCA	1	1	3	1
TGACTC	2	2	4	2
AGTCAT	3	3	5	7
ATGACT	4	4	6	8
GACTCA	5	5	7	3
TGAGTC	6	6	8	4
AGTCAC	7	7	9	7
GTGACT	8	8	10	8
TAGTCA	9	11	12	N/A

INO Family:

String	V.H. site	Binomial	Poisson	Expected/Actual
CACATG	1	1	1	1
CATGTG	2	2	2	2
TGTGAA	3	6	5	37
TTCACA	4	7	6	36
CCACAC	5	31	N/A	N/A
GTGTGG	6	32	N/A	N/A
GCGGCA	7	27	25	10
TGCCGC	8	28	26	12
TAGTGA	9	46	N/A	N/A

HAP Family:

String	V.H. site	Binomial	Poisson	Expected/Actual
AGCAAT	1	5	5	N/A
ATTGCT	2	6	6	N/A
ACAAGG	3	7	7	43
CCTTGT	4	8	8	44

YAP Family:

String	V.H. site	Binomial	Poisson	Expected/Actual
GGGGTA	1	1	3	N/A
TACCCC	2	2	4	N/A
CCCCAC	3	3	5	N/A
GTGGGG	4	4	6	N/A
ACCCCG	5	9	11	N/A
CGGGGT	6	10	12	N/A
ACCCCA	7	7	9	N/A
TGGGGT	8	8	10	N/A
AGGCAC	9	5	6	N/A
GTGCCT	10	6	7	N/A
AAAAAT	11	33	33	N/A
ATTTTTT	12	34	34	N/A

Discussion

On the whole, the results generated by our program match those of van Helden's site very closely. In some cases, such as that of the INO family, there is considerable variation. This can be explained by the fact that van Helden uses several filtering techniques to generate more accurate results and we don't have full access to these. He briefly outlines them in his paper and on his site. While some of the filters are easy to comprehend, the effect of others is difficult to decipher. In our program, we used some of these filters and we created others ourselves. These include:

- Setting a minimum limit for substring occurrences
- Restricting the size of analyzed substrings

- Not adding the reverse complement of a substring if it is symmetrical to the original substring
- Halving the frequency, expected occurrences and occurrences of substrings to speed up the binomial distribution.

We will now address the results for each gene family individually:

1. **MET:** In the MET family, we arrive at the same results as van Helden. In this case, the filters – purge sequences, etc – did not change the outcome. The Binomial and Poisson distributions came up with very accurate, usable results.
2. **NIT:** Similar to the MET family, our program found the same substring frequencies as his, except in the case of some lower ranked substrings. Specifically, GCGCGG and CCGCGC appeared in our list of top substrings, but not in van Helden's. Substring with 'G' and 'C' are overly abundant in this family and Van Helden handles this special case. Other subsequent cases diverged by a few ranks as a result. Since we cannot confirm the biological truth of this overrepresentation, we decided not to use the same filter. Of course, when the frequencies computed by our program match van Helden, we get the same results.
3. **PHO:** van Helden's program only found four significant substrings. Since we do not use a significance coefficient like he does, but use only raw probabilities instead, we rank more candidates.
4. **PDR:** Despite differences in filters, our results matched van Helden's.
5. **GAL:** Van Helden did not come up with any plausible regulatory genes, thus we do not have any of his results to compare ours with. However, our results are listed anyway.
6. **GCN:** Here, we are accurate on all results except for the very last subsequence (rank 10). Again, this can be attributed to the different filters we use. The Poisson distribution ranked AAAAAA as 1. The letter A seems to be unusually common in this family and van Helden seems to have accounted for this.
7. **INO:** We see significant variation in results here. Once again, these can be attributed to difference in filters. Despite this, we still returned the same substring in the top two spots.
8. **HAP:** Our results were fairly accurate. Our ranks five through ten matched his ranks one through five. Filtering methods led to these differences. Our program chose the substrings GAGAGA, AGAGAG, TCTCTC and CTCTCT as the top ones. These are overly abundant in the family gene and van Helden seems to have filtered these out.
9. **YAP:** Due to filtering differences, we only accurately matched the top four nucleotides.

We have concluded that, in spite of variations caused by filtering differences, we have managed to match van Helden's technique and results to a satisfactory degree. Our first scoring method, using expected and actual frequencies, turns out only to be a rough heuristic since it disagrees with the other scoring methods to

a significant degree. The Binomial score has the closest correspondence with van Helden's results. However, the Poisson score, although divergent, seems to have a better correspondence to actual biology. It seems to eliminate substrings that are overly represented by the inherent biological nature of the yeast genome. Perhaps van Helden should also use the Poisson distribution on his site to come up with more interesting results.

There is much scope for future work in this area. The presence and effectiveness of filters underscores the fact that our program and method are to a large extent only as good as the filters we use. Without the use of these filters, the results our program generated seemed haphazard and chaotic. Thus, the development of new, effective filters is very important for the development of this method. This process of motif detection seemed very intuitive and elegant. In our opinion, it could be applied to the genomes of other similar organisms to gain even more information about gene regulation. Finally, the only way by which the veracity of these results can be proved is in the lab. Biological verification would go a long way in the refining of the method. It would be interesting to see how much actual correspondence the generated probability matrices have to reality.

We did learn a lot in the process of creating this program. We had a first-hand look at genetic data and became familiar with it. What we had only heard and talked about in the abstract before, we were able to experience fully. We had a peek into some of the frantic biological investigation that is being performed at such breakneck speed today. We also learned how to work more cohesively as parts of a group, learning important lessons about co-ordination and collaboration. Finally, we learned how to effectively interface Computer Science with other scientific disciplines. In all, it was a very valuable experience.

References

van Helden, J., André, B. & Collado-Vides, J. (1998). Extracting regulatory sites from the upstream region of yeast genes by computational analysis of oligonucleotide frequencies. *J Mol Biol* 281(5), 827-42.

van Helden, J., André, B. & Collado-Vides, J. (2000). A web site for the computational analysis of yeast regulatory sequences. *Yeast* 16(2), 177-187.