Inferring Interactions, Expression Programs and Regulatory Networks from High Throughput Biological Data

by

Ziv Bar-Joseph

BSc. Computer Science and Mathematics, Hebrew University (1997)
MSc. Computer Science, Hebrew University (1999)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2003

Author .....
Department of Electrical Engineering and Computer Science
June 20, 2003

Certified by .....
David K. Gifford
Professor
Thesis Supervisor

Certified by .....
Tommi S. Jaakkola
Associate Professor
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Inferring Interactions, Expression Programs and Regulatory Networks from High Throughput Biological Data

by

Ziv Bar-Joseph

Submitted to the Department of Electrical Engineering and Computer Science
on June 20, 2003, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

In this thesis I present algorithms for analyzing high throughput biological datasets. These algorithms work on a number of different analysis levels to infer interactions between genes, determine gene expression programs and model complex biological networks.

Recent advances in high-throughput experimental methods in molecular biology hold great promise. DNA microarray technologies enable researchers to measure the expression levels of thousands of genes simultaneously. Time series expression data offers particularly rich opportunities for understanding the dynamics of biological processes. In addition to measuring expression data, microarrays have been recently exploited to measure genome-wide protein-DNA binding events.

While these types of data are revolutionizing biology, they also present many computational challenges. Principled computational methods are required in order to make full use of each of these datasets, and to combine them to infer interactions and discover networks for modeling different systems in the cell.

The algorithms presented in this thesis address three different analysis levels of high throughput biological data: Recovering individual gene values, pattern recognition and networks. For time series expression data, I present algorithms that permit the principled estimation of unobserved time-points, alignment and the identification of differentially expressed genes. For pattern recognition, I present algorithms for clustering continuous data, and for ordering the leaves of a clustering tree to infer expression programs. For the networks level I present an algorithm that efficiently combines complementary large-scale expression and protein-DNA binding data to discover co-regulated modules of genes. This algorithm is extended so that it can infer sub-networks for specific systems in the cell. Finally, I present an algorithm which combines some of the above methods to automatically infer a dynamic sub-network for the cell cycle system.

Thesis Supervisor: David K. Gifford
Title: Professor

Thesis Supervisor: Tommi S. Jaakkola
Title: Associate Professor

ליפעת:
זכרתי לך חסד נעורייך
אהבת כלולותייך
לכתך אחריי במדבר
בארץ לא זרועה
ירמיהו ב'

To Ifat:
I remember for thee the affection of thy youth,
the love of thine espousals;
how thou wentest after Me in the wilderness,
in a land that was not sown

Jeremiah, Chapter 2

Acknowledgments

First and foremost I want to thank my advisors Profs. David Gifford and Tommi Jaakkola.

In addition to his vast knowledge in many areas of computer science, Dave has a true passion for biology. I have benefited greatly from his knowledge of molecular biology, and from his perspective about computational biology. Dave was always there when I needed help with identifying research problems, and ways to approach these problems. Particularly, he helped me focus on the big picture, and always encouraged me to work on the most important problems in computational biology. In addition, Dave has initiated a very fruitful collaboration with the Ricks' Young laboratory at the Whitehead institute, which allowed me to work with some of the leading biologists on some of the most fundamental problems in genomics.

It probably took me over a year to start deciphering Tommi's comments, but once I was able to do that my research really took off. It was always great to come to Tommi when my research seemed to be reaching a dead end, and emerge from his room thirty minutes later with a solution to all of my problems (though I had a spend another week figuring out what exactly the solution was). Tommi was also instrumental in initiating my interest in time series expression analysis and in machine learning. While I still have a lot to learn about machine learning, I was truly fortunate to be advised by someone with such understanding of this area.

In addition to my advisors, I was fortunate to have Georg Gerber as a fellow graduate student, colleague and a friend. A lot of the work described in this thesis was done in collaboration with Georg. I believe that Georg represents the ultimate computational biologists, combining vast knowledge of computer science with active training in biology and medicine. Georg and I have similar background (in computer graphics) which is probably why we see eye to eye on many of the research problems we have worked on. I am sure Georg will become one of the leaders in this field, and I hope that when this happens, he will still be interested in working with me. I look forward to many more years of joint work.

I am also grateful to my third committee member, Prof. Rick Young. Throughout my work, Rick encouraged me and gave me the (hopefully correct) impression that he was excited about my research. I believe that collaboration is key to success in this area, and I can just hope I will be able to find someone who is as passionate and dedicated as Rick is

to work with in the future.

I have also enjoyed fruitful collaboration and discussions with many of the students and postdocs at the Young lab. In particular, I want to thank Tony Lee, Itamar Simon, Duncan Odom and Nicola Rinaldi. I am sure that at least Tony will be relived when I am finished, he still dreads my phone calls asking him to for 'small' (a few experiments) favors.

I enjoyed discussions with the other past and present members of our computational genomics group, including Alex Hartemink, Tim Danford and Rienna Reiman. I also want to thank our administrative assistant, Jeanne Darling, for making my life much easier during these four years, and for educating me about horses. I was also involved with Tommi's machine learning group. I especially want to thank Nati Srebro, for both working with me on a number of issues, and for helping me during my first year at MIT. I also had very interesting discussions with Chen-Hsiang Yeang. We share many research interests (and a room at ISMB) and I am sure I will continue to stay in touch with him in the future.

While I spent most of my time working on computational biology, I spent my first year (and parts of my other years) working with Prof. Nancy Lynch and Dr. Idit Keidar on distributed computing. It is probably because of Nancy's support and encouragement that I was able to complete my first year at MIT. While my research interests shifted to another area after that year, I still learned a lot from Nancy and Idit, and I can just hope I will be able to treat and help my graduate students the same way Nancy treated me.

Finally, I want to thank my family. My mother and father played an important role in my decision to come to MIT. Growing up with parents that are dedicated to, and passionate about their work, leaves you almost no choice but to try and discover what's so interesting in research. Though it took me many years, I can say that following their footsteps was the right decision. They have also supported me both emotionally and financially throughout my studies, and I am grateful for that. My brother and sister, Chen and Yaara, where always there when I needed them, and I hope to see much more of them in future years.

Last, but not least, I must thank my wife. Ifat had to make the almost impossible decision of leaving her home country and her family with a two month old baby, to help me pursue my dreams. Still, when during my first year I had doubts if I should stay or go back, Ifat was there to support and encourage me, which eventually led to this thesis. I am forever grateful for her help and support through this long process.

Contents

1 Introduction **17**

1.1 High throughput biological data sources 17

1.2 Microarrays 18

1.2.1 Synthesized arrays 19

1.2.2 Pre-synthesized DNA arrays 20

1.2.3 Location arrays 21

1.3 Gene expression data 22

1.4 Time series expression data 22

1.4.1 Examples of time series expression experiments 24

1.5 Computational challenges in expression analysis or Thesis roadmap 27

1.5.1 Experimental design 28

1.5.2 Individual gene level 29

1.5.3 Pattern recognition level 29

1.5.4 Networks level 31

1.6 Contribution 32

I Individual Gene Level **34**

2 Continuous representation of time series expression data **37**

2.1 Related work 38

2.2 Splines 39

2.3 B-splines 41

2.4 Estimating unobserved expression values and time points 42

2.4.1 A probabilistic model of time series expression data 43

2.4.2 Estimating spline coefficients from experimental data 45
2.5 Results 46
2.6 Summary 49

3 Alignment **51**
3.1 Related Work 51
3.2 Continuously aligning temporal expression datasets 52
3.3 Results 54
3.4 Summary 57

4 Identifying differentially expressed genes in time series expression data **59**
4.1 Overview of our algorithm 60
4.2 Related work 61
4.3 Hypothesis testing for differentially expressed genes 62
4.3.1 Noise model for individual samples 63
4.3.2 Combining a sample noise model with global difference 64
4.3.3 Solving the maximization problem 65
4.3.4 The complete algorithm 67
4.3.5 Value specific variance 68
4.4 Results 70
4.4.1 Synthetic data 70
4.4.2 Yeast cell cycle and knockout data 71
4.5 Summary 79

II Clustering and Ordering **81**

5 Clustering continuous curves **83**
5.1 Model based clustering algorithm for temporal data 84
5.1.1 The modified EM algorithm 85
5.2 Results 87
5.3 Summary 89

6 Optimal leaf ordering for hierarchical clustering trees **91**

6.1 Hierarchical clustering 92

6.2 Related work 92

6.3 Problem statement 93

6.4 An $O(n^3)$ algorithm for binary trees 94

6.4.1 Early termination of the search. 96

6.4.2 Top-level improvement. 96

6.5 Results 96

6.5.1 Generated data 98

6.5.2 Biological results 99

6.6 Summary 100

7 K-ary clustering and ordering **103**

7.1 Related work 104

7.2 Constructing K-ary Trees 105

7.2.1 Problem statement 105

7.2.2 A heuristic algorithm for constructing k-ary trees 107

7.2.3 Reducing the number of children 108

7.3 Optimal leaf ordering for k-ary trees 110

7.3.1 Improving the running time using a divide and conquer algorithm . 112

7.4 Experimental results 114

7.4.1 Generated data 114

7.4.2 Visualizing biological datasets 115

7.4.3 Clustering biological datasets 116

7.4.4 Comparing the sum of similarities for different trees 119

7.5 Extensions: optimal ordering for lossless image compression 119

7.6 Summary 120

III Modeling Biological Systems **121**

8 Computational discovery of gene modules and regulatory networks **123**

8.1 Related work 125

8.2 The GRAM algorithm 126

8.2.1 Identifying gene modules 127
8.2.2 Sub-network discovery algorithm 131
8.3 Results 132
8.3.1 Gene modules in rich media conditions and modes of regulatory control 132
8.3.2 Validation of the GRAM algorithm results using independent data sources 141
8.3.3 Discovering networks for systems in the cell 144
8.4 Discussion 153
8.4.1 Limitations of the GRAM algorithm 156
8.4.2 Summary 157

9 Conclusions and future work **159**
9.1 Conclusions 159
9.2 Relation to other problems in computer science 160
9.2.1 Graphical models 161
9.2.2 Visualization and graphics 162
9.3 Future work 162
9.3.1 Determining sampling rates for time series expression experiments . 163
9.3.2 Combining static and time series expression data 164
9.3.3 From model organism to humans 164

A Appendix **167**
A.1 The symmetric version of our algorithm 167
A.2 Non cycling genes 168

List of Figures

1-1 Central dogma of molecular biology. 23

1-2 The hierarchy of analysis levels for gene expression. 27

2-1 B-spline basis functions. 42

2-2 Comparison among different missing value interpolation techniques. 48

2-3 Comparison among four different methods for interpolating time series expression data. 49

3-1 Alignment of genes from the cdc28DS to cdc15DS. 55

3-2 Alignment of three different genes for the three WT cell cycle experiments. 56

4-1 Complete algorithm for identifying differentially expressed genes in time series data. 68

4-2 Value specific variance 69

4-3 The four different sets we generated to test our algorithm. 72

4-4 Comparison with methods that work directly on the input samples. 72

4-5 Differentially expressed cell cycle genes. 75

4-6 Results of clustering the 56 cell cycle genes identified by our algorithm as differentially expressed. 76

4-7 Genes identified by our algorithm that were missed by previous methods. . 77

5-1 Estimating the model parameters without class information. 84

5-2 A comparison between Gaussian mixture and TimeFit. 88

6-1 Flipping two subtrees rooted at an internal node. 94

6-2 Computing $M(v, i, j)$ for a binary tree rooted at V. 94

6-3 Average increase in the sum of similarities of neighboring leaves for a randomly generated dataset. 97
6-4 Comparison between the ordered and non ordered hierarchical clustering on a generated dataset. 98
6-5 Comparison between the ordered and non ordered hierarchical clustering. . 99
6-6 Comparison between hierarchical clustering and optimal ordering using the cell cycle data. 101

7-1 Constructing k-trees from expression data 107
7-2 Fixed vs. non fixed k. 109
7-3 Computing $M(v, i, j)$ for the subtrees order $1 \ldots k$. 111
7-4 Comparison between the three different clustering methods on a manually generated dataset. 115
7-5 Comparison of the three different clustering algorithms using the *myc* overexpression dataset. 117

8-1 GRAM algorithm for identifying gene modules. 128
8-2 Rich media gene modules network. 135
8-3 The GRAM algorithm integrates binding and expression data and improves on either data source alone (a). 136
8-4 The GRAM algorithm integrates binding and expression data and improves on either data source alone (b). 137
8-5 Examples of an activator and a repressor. 140
8-6 Combinatorial regulation modes. 142
8-7 Motif abundance 145
8-8 Amino acid starvation sub-network. 147
8-9 Cell cycle sub-network. 149
8-10 Rapamycin gene modules network 151
8-11 Comparison with previous modules work. 154
8-12 Comparison with previous modules work. 155

List of Tables

1.1 Summary of five different time series expression experiments. 25

3.1 The parameters used by our alignment algorithm. 53

3.2 Results of experiments in which random subsets of fixed size were sampled 100 times and alignment of alphaDS and cdc15DS were performed. 57

4.1 Top 30 differentially expressed cell cycle genes identified, ordered by significance p-value. 74

4.2 The 22 non cycling genes identified by our algorithm as differentially expressed. 78

5.1 Comparison between Gaussian mixtures and TimeFit using the cdc15DS. . 89

7.1 Comparison between the binary tree and 4-ary clustering algorithms using protein complexes. 118

8.1 Activators identified by our algorithm. 137

8.2 Repressors identified by our algorithm. 138

8.3 A module controlled by Swi4 and Stb1. 143

A.1 The 20 expression experiments in which the set of non cycling genes identified by our algorithm were significantly correlated. 170

Chapter 1

Introduction

The journal Nature published a special issue recently, celebrating fifty years to the discovery of the structure of DNA by Watson and Crick [102], and the completion of the human genome project. In that issue, a group from the U.S. National Human Genome Research Institute published an article titled: 'A vision for the future of genomics research' [33] in which they outlined fifteen grand challenges for molecular biology for the next several decades (in a fashion similar to the challenges set forth for mathematics by David Hilbert at the beginning of the twentieth century [57]). One of these challenges reads as fellows:

Grand Challenge I-2 *Elucidate the organization of genetic networks and protein pathways and establish how they contribute to cellular and organismal phenotypes.*

In this thesis we try to address parts of the above challenge. Specifically, we present computational methods for answering questions about how genes interact, how are expression programs carried out on the molecular level and what are the regulatory networks that different systems in the cell use to respond to various external and internal changes.

1.1 High throughput biological data sources

The sequencing of the human genome is just the first step toward understanding cellular activity. While many organisms have been fully sequenced by now(human [71, 100], fly [4], yeast [51]) little is known about the function of most of the genes that were identified in these organisms. In order to obtain a more comprehensive picture of the activity in the cell,

we would like to answer questions about how genes interact, how are expression programs carried out on the molecular level and what are the networks that different systems in the cell use to respond to various external and internal changes. While the same DNA sequence is present in all cell types of the same organism, different cells express different subsets of genes to fulfill their tasks. Even in the same cell, different systems use different subsets of the expressed genes, as we show in this thesis. Thus, even if we obtain complete understanding of these sequences, the above questions cannot be answered by using sequence data alone.

Recent advances in DNA microarray technology [91] allow researchers to measure the expression levels of thousands of genes in the cell simultaneously. Expression levels correspond to the amount of mRNA for each gene (see below), and these levels vary under different experimental conditions, and between different cell types. Thus, this data provides an opportunity to address the above question on a global (whole cell) basis. Other experimental techniques [85] provide information about the binding location of transcription factors. Transcription factors are proteins that bind DNA and regulate genes by either activating or repressing their expression. By using this data, we can infer interactions between genes. By combining binding and expression data we can discover complex interaction networks that various systems in the cell employ to fulfill their goals.

While these types of data are revolutionizing biology, they also present many computational challenges. These challenges result from the new technology used, the underlying biology and the fact that each of these experiments presents results for thousands of genes at once. Further, since we are dealing with different types of data, collected at various biological labs, a great challenge is data integration and information fusion. We discuss these challenges in detail below. Thus, careful analysis of this data is required in order to determine the specific function of each gene, to infer interactions between genes and to combine biological dataset in order to discover regulatory and other networks in the cell. This is exactly the goal of the algorithms described in this thesis.

1.2 Microarrays

In this section we present a brief overview to DNA and binding microarrays. These arrays produce the two sources of data we will use later in this thesis to answer the biological problems listed below.

Complementary DNA strands can be hybridized together through incubation. DNA microarrays use a "probe" containing the complementary genetic material in order to determine the expression levels of genes. There are two major types of DNA microarrays and they differ in the process in which these probes are generated. In addition, in this thesis we also analyze data from a different kind of microarray: Protein-DNA binding array (known also as location or CHIP arrays [73]). Unlike other arrays that measure the amount of transcripts for each gene, binding arrays measures the binding of proteins to intergenic regions in the DNA. Below we discuss in more details the two types of DNA expression arrays, and binding arrays.

1.2.1 Synthesized arrays

The largest producer of synthesized or oligonucleotide arrays is a company called Affymetrix [1]. In this thesis we will restrict our discussion to the Affymetrix GeneChip arrays (though arrays produced by other companies are similar). GeneChip arrays consist of hundreds of thousands of *features*. Each feature is an oligonucleotide, typically about 25 bases long. These features act as probes and can bind to specific target DNA molecules. For each gene, 11 to 16 probes are selected among all possible 25-mers to represent this gene. When choosing these probes, care is taken so that cross-hybridizing with similar, but unrelated sequences, is minimized.

In order to further reduce the effect of cross hybridization, Affymetrix uses the Perfect Match/Mismatch probe strategy. For each complementary probe, a partner probe is generated that is identical except for a single base mismatch in its center. These probe pairs allow the quantification and subtraction of signals caused by non-specific cross-hybridization. The difference in hybridization signals between the partners, as well as their intensity ratios, serve as indicators of specific target abundance.

Affymetrix uses a combination of photolithography and combinatorial chemistry to synthesize sequences on a quarts wafer. Probe synthesis occurs in parallel, and the different bases (A, C, T, and G) are added to multiple growing chains simultaneously. Photolithographic masks are used to define which base will be added next to each of the features. Synthesis is performed in steps, and in each of these steps, a new mask is placed over the wafer. This process is repeated until the probes reach their full length of 25 nucleotides.

In order to prepare the sample target, mRNA is collected from cells of interest (for

example, cancer cells). Using reverse transcription, cDNA is generated from the mRNA. This allows the amplification of the amount of starting mRNA. Next, cDNA is transformed back to mRNA, and the mRNA is labeled, so that the amount of transcripts hybridized to each array feature can be measured using an optical scanner. The Affymetrix scanning device reports the expression level by subtracting the spot intensity of the mismatch feature from their partner match feature, and averaging across all features for each gene.

Note that oligonucleotide arrays can only be manufactured for organisms which have been sequenced. Since each gene is represented by a chemically produced 25-mers, we need to know the sequence of each of the genes in advance in order to manufacture the array. While this was an issue in the early days of microarray usage (especially for human arrays), this is becoming less of an issue as more and more organisms are being sequenced. For organism which have not been fully sequenced, cDNA arrays provide a reasonable alternative to oligonucleotide arrays, as we discuss below.

1.2.2 Pre-synthesized DNA arrays

Instead of the 25-mers used by the synthesized arrays, most pre-synthesized arrays [42] use strands of cDNA extracted from a target of interest. Thus, we refer to them as cDNA arrays in this thesis. These strands are usually much longer (up to hundreds of base pairs), and are typically obtained from cDNA libraries. Such libraries are constructed by making cDNA copies of mRNA from a specific organism or cell type. Following polymerase chain reaction (PCR) amplification, these stands are printed onto a glass slide or a nylon membrane (and thus they are also known as *printed* arrays).

Since the probe deposit process used to generate cDNA arrays is much less accurate than the process used in oligonucleotide arrays, different spots on the arrays can vary in size, and in the amount of mRNA per spot. This complicates direct comparison between different cDNA arrays, even if the same library is used to manufacture both arrays. In order to overcome this problem, two separate samples labeled with different fluorescent dyes are simultaneously hybridized to the array. Examples include normal and cancer cells (for the static case) and synchronized and unsynchronized cells (for the time series case). Instead of using the actual reading of the scanning device, the *ratio* between the two samples is used to determine the expression levels. One of the samples is used in all arrays, and the second changes between different arrays. For example, by hybridizing normal samples to all

arrays and samples from different cancer types to different arrays, we can use cDNA arrays to identify genes that are specifically involved in a certain type of cancer, even if the actual transcript level of these genes is unknown.

Apart from using two different cell types and measuring their ratio, sample preparation is similar to that of oligonucleotide arrays. As mentioned above, cDNA arrays can be used for any organism, even if it has not been fully sequenced. Another advantage of cDNA arrays is that they are usually much cheaper than oligonucleotide arrays. Indeed, most of the data analyzed in this thesis (as well as most of the published expression data) comes from cDNA arrays.

1.2.3 Location arrays

While the above two array types are targeted to find the expression of genes in the cell, location (or binding) arrays [73] aim to identify the location at which proteins known as transcriptional factors (TF) binds in the cell. Location arrays are similar to the pre-synthesized arrays (though synthesized arrays can also be used, the results presented in this thesis only use pre-synthesized arrays). However, we regard them as a separate type since, unlike cDNA arrays which are printed with the coding region of the DNA, location arrays contain the *intergenic* (or upstream) regions.

For each gene (or for a subset of genes, depending on the number of genes in the specific organism), a 500 base pair upstream region is extracted and printed on the array. This region contains the binding sites of various transcriptional factors (TFs) that bind in front of the gene, and regulate its expression. In order to determine the genes that are bound by a specific TF, strains are grown in which this TF is labeled using a myc epitope tag. Using printed arrays, the ratio of immunoprecipitated to control DNA (which is not tagged) is determined. This process is repeated a number of times and a confidence value (p-value) is calculated for each spot from each array using an error model. By thresholding the p-values (for example at .001 in [73]), a binary relation is determined for each of the genes with each of the TFs that are profiled.

While binding experiments can in principle be carried out in a time series, to date no such experiment have been performed. Since this is a very new technology (the first paper using this technique is from 2001 [85]) we expect that this will change in the future. Still, since each such experiment can only profile one of the TFs, it is an important challenge to

combine this static binding information with the dynamic (or time series) information that is available from expression arrays. In section 8 we discuss an algorithm that can combine the two array types to discover dynamic regulatory networks.

1.3 Gene expression data

DNA microarray experiments can be classified in a number of ways. A common way is to divide them based on the type of array that is used in the experiment (cDNA and Oligonucleotide arrays [12]). Another common way is to divide these experiments according to the organism that is profiled (yeast [94, 43], mice [95], human [52, 78]). In this thesis we introduce a new way that is targeted towards the analysis of such experiments. Gene expression experiments can be divided into static and time series experiments. In *static* expression experiments, a snapshot of the expression of genes in different samples is measured (e.g., cancer versus normal [52], knockout [63]). In *time series* expression experiments, a temporal process is measured (e.g., infection [62, 78], response to environmental conditions [49], or a specific system in the cell [94, 107, 82]). Since gene expression is a temporal process (see Chapter 1.4), it is essential to use time series expression data to address the inference problem listed above. While most previous work used methods developed for static data to analyze time series data [94, 47, 98, 107], we dedicate a large part of this thesis to principled methods for analyzing time series gene expression data. As we show in this thesis, it is necessary to develop algorithms that are specifically targeted towards time series expression data in order to fully utilize this data. Using such methods we are able to solve many biologically important problems, including the identification of differentially expressed genes, assignment of genes to different functional categories and the discovery of dynamic genetic regulatory networks.

1.4 Time series expression data

One of the goals of this thesis is to provide principled algorithms for analyzing time series expression data, and for combining this data with other biological datasets to discover dynamic regulatory networks.

Gene expression is a temporal process. According to the central dogma of molecular biology[5], in order to transform the information stored in the genome (or DNA sequence)

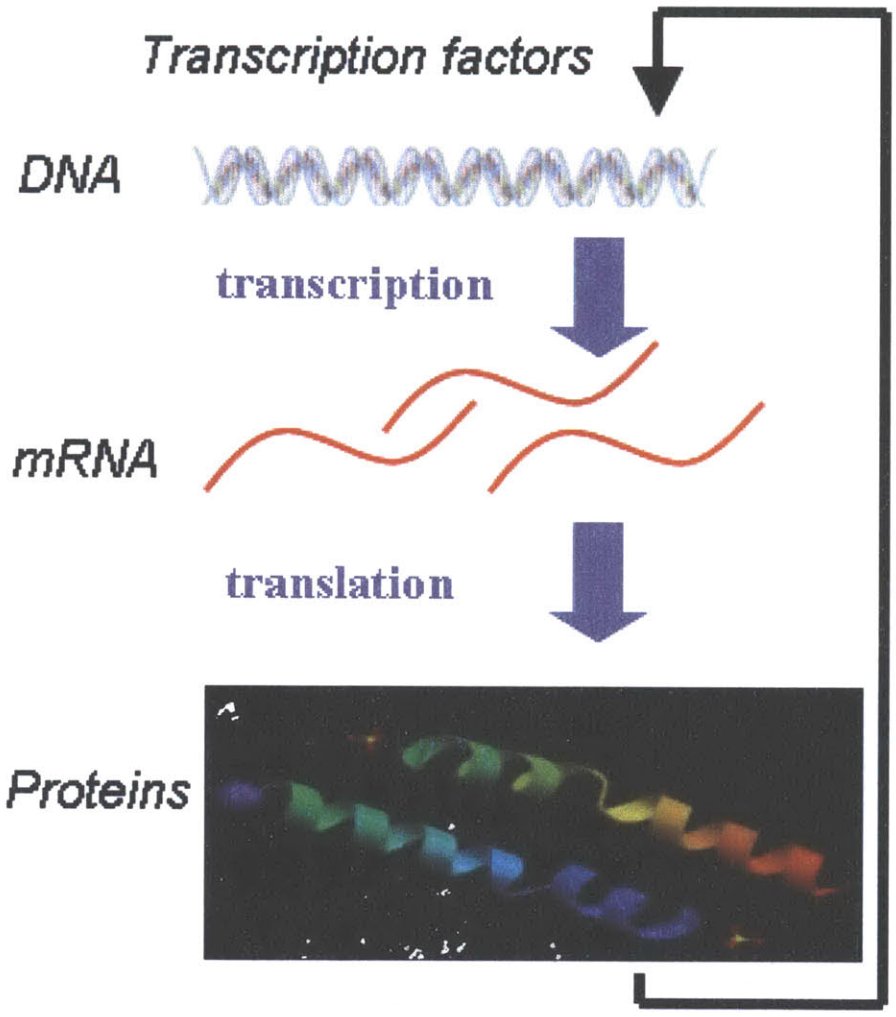


Figure 1-1: Central dogma of molecular biology. Genes are first transcribed to mRNA and later translating into proteins. This process is regulated by a feedback loop, and some of the proteins serve as transcription factors, regulating the expression of other genes.

into proteins (which are the working parts of the cell), genes are first transcribed into mRNA and then translated to proteins (see Figure 1-1). This is a continuous process, as different proteins are required for different functions and under different conditions. Even under stable conditions, due to the degradation of proteins, mRNA is continuously transcribed and new proteins are generated. Since gene expression is vital for cells, this process is highly regulated. One of the most important ways in which the cell regulates gene expression is by using a feedback loop. Some of the proteins are known as transcription factors (TFs). These proteins regulate the expression of other genes (and possibly, their own expression) by either initiating or repressing transcription.

When cells are faced with a new condition (such as starvation [77], infection [78] and stress [49]), they react by activating a new expression program. In many cases, the expression program starts by activating a few transcription factors, which in turn activate many other genes that act in response to the new condition [77]. Taking a snapshot of the expression profile following a new condition can reveal some of the genes that are specifically expressed under the new condition. However, in order to determine the complete set of genes that are expressed under these conditions, and to determine the interaction between these genes, it is necessary to measure a time course of expression experiments. This allows us to determine not only the stable state following a new condition, but also the pathway and networks that were activated in order to arrive at this new state. Indeed, biologist are well aware of these issues, and a large number of systems, diseases and developmental processes are currently studied using time series gene expression data. As microarray technology becomes more common and cheaper, a large number of microarray experiments will be time series experiments.

1.4.1 Examples of time series expression experiments

In this section we describe a few examples from four different types of gene expression experiments. The main purpose of this section is to demonstrate the wide range of biological question that expression data, and time series expression data, can be used to answer. Many of these questions involve computational aspects, as we discuss in Chapter 1.5.

Reference	Method of arrest	Duration	Cell cycle length	Sampling rate
WT alpha [94]	alpha mating factor	0-119m	64m	every 7m
WT cdc15 [94]	temperature sensitive cdc15 mutant	10-290m	112	ev. 20m for 1 hr, ev. 10m for 3 hr, ev. 20 min for final hr
WT cdc28 [31]	temperature sensitive cdc28 mutant	0-160m	85m	every 10m
fkh1/fkh2 knockout [107]	alpha mating factor	0-210m	105m	ev. 15m until 165m, then after 45m
yox1/yhp1 knockout [82]	alpha mating factor	0-120m	60m	every 10m

Table 1.1: Summary of five different time series expression experiments. WT- wild type, m-minutes. All of these experiments were performed to study the cell cycle system in yeast. Note that the sampling rates are not always uniform, and vary between the different experiments. In addition, the cell cycle duration (the time it takes the cells to divide) differs depending on the experiment condition.

Biological systems

The biological system that has been most extensively studied using time series gene expression data is the cell cycle system in yeast. Yeast is a model organism, and many of the yeast genes have homologes in the human genome. The cell cycle is of particular interest, as it plays an important role in cancer, and has thus been extensively studied over the last four decades [92]. In table 1.1 we present 5 different time series expression experiments that where carried out to study various aspects of this system. As can be seen, three of these experiments measure wild type (WT) behavior under different conditions, while the two other look at knockout strains. Table 1.1 can also serve to motivate some of the work in this thesis. While all five experiments study the same system, all show different length of the cell cycle duration (ranging from 60 to 120 minutes). Further, these experiments were sampled at different rates (ranging from 7 to 20 minutes) and different durations (from 2 hours to 5 hours). Since each of these datasets is noisy and contains many missing values, we need to combine multiple datasets to fully utilize these individual experiments. Thus, a major challenge is to develop algorithms that will allow us to combine measurements carried out under different conditions. A number of other systems have also been studied with time series expression experiments, including, among others the circadian clock in mouse and

humans [95, 80].

Genetic interactions and knockouts

While an expression time course of a WT systems is useful to determine the set of genes that function in this system, and potentially the order in which they operate, in order to study the function of individual genes we need to carry out knockout experiments. In a knockout experiment a gene is (or a number of genes are) deleted from the genome, and these deleted strains are profiled using expression experiments. Such experiments allow us to determine the down stream effects of the knockout (or knocked out) gene(s), which in turn can be used to identify target genes and to construct genetic interaction networks. Many knockout expression experiments have been carried out in the static case [63]. More recently many knockout time courses are becoming available. These include cell cycle double knockouts [107, 82] and knockouts under stress conditions [49].

Development

Understanding development is key to understanding many genetic diseases. It is natural to use time series expression experiments to study development at the molecular level, and to identify genes that play key role in different stages of development. For example, an 80 time points expression experiment studying the development of the fruit fly *Drosophila* identified many genes that control specific stages in the fly developmental process [8]. Similar experiments were carried out in other organisms, including the worm *c. elegans* [69]. More recently, expression experiments have been carried out to study human development. In [66] human embryonic stem cells have been profiled in order to identify genes that are involved in the specific differentiation of these cells to various tissue types.

Infectious and other disease

Identifying genes that act in response to certain infectious disease is a key issue in developing drugs to fight these disease. Nau *et al* [78] studied a time course of human cells that were infected by four different pathogens. Other examples include Huntington disease [104] and cancer [52].

As the examples above suggest, expression experiments can be used to answer many biologically important problems. However, as indicated above and as we discuss in the rest

of this section, addressing these issues requires us to solve many computational problems as well. Thus, algorithms that directly target this type of data are essential to all of the above studies.

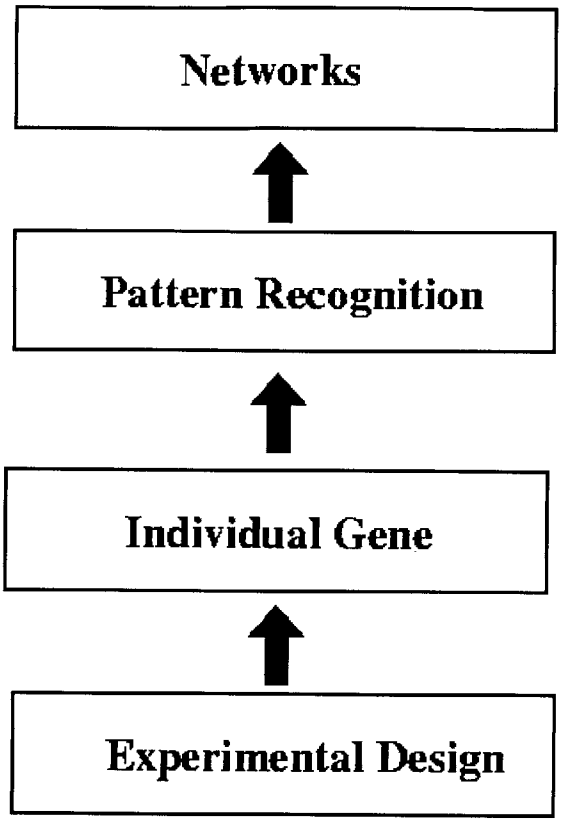


Figure 1-2: The hierarchy of analysis levels for gene expression. Each level addresses a number of computational and biological problems, and also serves as a pre-processing step for higher levels in the hierarchy.

1.5 Computational challenges in expression analysis or Thesis roadmap

The analysis of gene expression experiments in general, and time series expression experiments in particular can be divided into four analysis levels (see Figure 1-2). Three of these levels (individual gene, pattern recognition and networks) are performed after the actual expression experiment, while the fourth (experimental design) is performed prior to the experiment. Since this thesis focuses on publicly available datasets (that is, on experiments

that have already been performed), we restrict our discussion to the three upper levels. However, for the completeness of presentation we discuss the experimental design level in this section. In addition, in Chapter 9 we mention extensions to some of our algorithms that can be used for experimental design. Below we discuss in detail the four analysis levels. For each of them we present the computational and biological problems that arise at that analysis level. We also point to the specific section in this thesis in which these issues are addressed.

1.5.1 Experimental design

Experimental design is key to the success of any expression experiment. In the past, various aspects of experimental design have been studied. For example, Zien *et al* [108] studied the number of microarrays required for expression experiments. Ben-Dor *et al* [25] studied the combinatorial problem of selecting representative probes for genes sequences in order to minimize cross hybridization with other genes.

Determining which experiments to carry out, and how to perform them is another experimental design issue. If we are studying a specific system in yeast, we might need to decide which of the possible 6000 knockout experiment would lead to the biggest improvement in our understanding of the system. Another important problem for designing time series expression experiments is the determination of sampling rates. If the experiment is under-sampled, the results might not correctly represent the activity of the genes in the duration of the experiments, and key events can be missed. On the other hand, over-sampling is expansive and time consuming. Since many experiments are limited by budget constraints, over-sampling will result in shorter experiment duration, which might lead to missing important genes that participate in the process at a later stage. As can be seen in Table 1.1, to date, sampling rates depended on biologists intuition, and varied (depending on the lab) even under similar experimental conditions (for example, the three alpha cell cycle experiments [94, 107, 82] were sampled every 7, 15 and 10 minutes respectfully). An algorithm that will aid in determining the right sampling rates will be an important tool, and can have immediate practical applications. In Chapter 9 we discuss possible extensions to some of our algorithms that can lead to an algorithm for determining the right sampling rate.

1.5.2 Individual gene level

This level focuses on the individual gene. At this level we address issues ranging from the raw expression values for each gene to determining interactions between pairs of genes.

Computational challenges at this level include, among others, normalization and overcoming missing values. For time series expression experiments, there is also the problem of overcoming the fact that we have only a few samples of a continuous process. Other problems at this level are discussed in detail in Chapter 2. Biological problems at this level include determining how biological processes unfold in time under different experimental conditions, which could be addressed by aligning the profiles of genes in different time series experiments (for example the five cell cycle experiments from Table 1.1) and identifying genes that are differentially expressed in time series experiments.

Solving the computational problems at this analysis level is important both for answering biological questions at this level, and as a pre-processing step for higher analysis levels. For example, without proper normalization, one cannot combine different array experiments. Likewise, continuous representation of time series data is important for discovering dynamic models for such data.

In this thesis we present algorithms that address both the computational issues (continuous representation and alignment) and the biological issues (identifying differentially expressed genes). In Chapter 2 we discuss an algorithm that uses statistical spline estimation to represent time-series gene expression profiles as continuous curves. As we show, this algorithm efficiently and accurately solves the missing values / missing experiments problem. In Chapter 3 we extend this algorithm using the same spline representation to continuously time-warp (or align) series. In Chapter 4 we discuss an algorithm that builds upon our continuous representation and alignment algorithms, and uses them for identifying differentially expressed genes in time series data.

1.5.3 Pattern recognition level

Due to the large number of genes that are profiled in each experiment, clustering is needed to provide a global overview of the experiment results. In addition, clustering was used to determine function for unknown genes [43], to look at expression programs for different systems in the cell [94] and for identifying sets of genes that are specifically involved in a certain

type of cancer or other diseases [6]. Another major challenge in gene expression analysis is effective data organization and visualization. Indeed, pattern recognition algorithms for gene expression data have been extensively studied, and several clustering algorithms have been suggested for such data [26, 83, 96].

While clustering is important for all expression experiments (static and time series), there are a number of issues that should be specifically addressed when clustering time series gene expression. First, most clustering algorithms (including k-means and self organizing maps [96]) treat their input as a vector of independent samples, and do not take into account the duration each time point represents. Since many time series are sampled non uniformly, such independence assumption might skew the results. In addition, when analyzing time series expression datasets we are interested not only in the clusters themselves but also in the relationships between the different clusters. This is especially important when using clustering algorithms for visualization purposes (for example, using hierarchical clustering). For time series data, such algorithms should provide an overview of the dynamics of the system as well as the different groups (or clusters) involved.

In this thesis we present algorithms for clustering non uniformly sampled time series expression data, and for determining expression programs by identifying relationships between different clusters. In Chapter 5 we present a clustering algorithm that infers classes by operating directly on a continuous representations of expression profiles. As this representation depends on the actual sampling rates, such clustering algorithm (implicitly) weights each input point based on the time it represents. In Chapter 6 we discuss an algorithm for optimally ordering the leaves of a binary hierarchical clustering tree. Hierarchical clustering is the most popular clustering algorithm in computational biology, and many of the papers that present new expression datasets use it to present their results. By ordering the leaves of the hierarchical clustering tree, our algorithms allows users to identify not only the clusters but also the relationships between these clusters. In Chapter 7 we discuss an extended version of our optimal ordering algorithm with a new hierarchical clustering algorithm which overcomes a number of problems that are inherent in the binary hierarchical clustering method. More specifically, we present a new hierarchical clustering algorithm that constructs a k-ary tree, where each node can have up to k children, and then optimally orders the leaves of that tree. By combining k clusters at each step our algorithm becomes more robust against noise and missing values. By optimally ordering the leaves of

the resulting tree we maintain the pairwise relationships that appear in the original method, without sacrificing the robustness.

1.5.4 Networks level

The final analysis level is the networks level in which we focus on the interactions between genes, and attempt to build descriptive and predictive models for different systems in the cell. The components of these networks are the genes (or their protein products) that are involved in a specific system in the cell, and the transcription factors that regulate this system. Such models should provide a description of the process that is being modeled (for example, the cell cycle or the immune response system), and the interaction that takes place during the activation of the system (how are the different genes involved activated ? which genes are turned on first ? which next ? etc.). Predictive models should also be able to address question about different perturbations of the system (what happens when we knockout a specific gene ? what will happen when we add this drug to the environment ?).

Such models are useful for many applications. For example, in drug discovery researchers are interested in identifying proteins that are at the root of a certain disease [52]. Using these models we can determine which genes are the causes, and target them to prevent the spread of the disease. Another important application is to identify side effects of a certain treatment. Targeting a protein can cause a number of side effects which might be toxic to the cell. Using genetic interaction models we can determine likely side effects in advance, and target only those proteins for which these side effects are minimal.

Unlike the lower analysis levels, expression data is not enough to accurately construct these networks. Due to the large number of genes, many different hypothesis can be generated to explain a specific expression pattern. In order to constrain the number of possible hypothesis, we need to incorporate additional biological data, and to 'fuse' such data with gene expression data. For example, genetic regulatory networks are key to understanding expression programs in the cell [73]. In order to accurately construct such networks, we need to combine disparate biological data sources, including binding and expression data. Thus, a key computational challenge at this level is to design algorithms that are capable of combining large scale biological data sources.

In Chapter 8 we address the above challenges, and introduce a new algorithm for ef-

ficiently combining complementary large-scale expression and transcription factor protein-DNA binding data to discover co-regulated modules of genes and associated regulatory networks. A module is a foundational building block that describes the interaction of transcription factors and the genes they regulate. We use the modules identified by our algorithm to uncover regulatory sub-networks for specific biological processes. We also show how to operate on the discovered networks of modules to label transcription factors as activators or repressors and to identify patterns of combinatorial regulation.

1.6 Contribution

This thesis presents principled algorithms for analyzing time series gene expression and protein-DNA binding datasets, in order to infer interactions, expression programs and models of cellular activity. As mentioned above, these problems can be largely grouped into four analysis levels. To the best of our knowledge, this is the first attempt to provide complete solution for all of these analysis levels. Thus, detailed discussion of previous work is differed to the appropriate chapters (which focuses on the specific algorithms we present in this thesis).

The major contribution of this thesis is:

- We present principled algorithms for the different analysis levels of high throughput biological data. These algorithms allow us to solve important biological problems by inferring interactions between genes, determining expression programs and modeling different systems in the cell.

Specific contributions include:

- The first general algorithm for identifying differentially expressed genes in time series data.
- A clustering algorithm that works on the continuous representation of the expression profile, overcoming problems related to non uniform sampling rates,
- An algorithm for optimally ordering the leaves of a k-ary tree, which allows us to improve (both theoretically and in practice) the most popular clustering algorithm in computational biology, hierarchical clustering.

- An algorithm which efficiently combines disparate data sources for the discovery of genetic regulatory networks, and models for systems in the cell.

We believe that the methods and algorithms described in this thesis will form the basis of a toolkit for experimental biologists working with time series expression data and with other high throughput biological datasets. As more and more time series expression experiments are carried out to study various aspects of cellular activity, these tools will become essential in order to fully utilize the potential of these experiments, and to advance biology and medicine.

Part I

Individual Gene Level

In this part we present algorithms for time-series gene expression analysis at the individual gene level. Our algorithms permit the principled estimation of unobserved time-points, dataset alignment and the identification of differentially expressed genes.

In Chapter 2 we address the problem of assigning a continuous representation to each genes' expression profile. Each expression profile is modeled as a cubic spline (piecewise polynomial) that is estimated from the observed data, and every time point influences the overall smooth expression curve. We constrain the spline coefficients of genes in the same class to have similar expression patterns, while also allowing for gene specific parameters. We show that unobserved time-points can be reconstructed using our method with 10-15% less error when compared to previous best methods.

In Chapter 3 we use the continuous representation described above to align two or more expression datasets. Our continuous alignment algorithm also avoids difficulties encountered by discrete approaches. In particular, our method allows for control of the number of degrees of freedom of the warp through the specification of parameterized functions, which helps to avoid overfitting. We demonstrate that our algorithm produces stable low-error alignments on real expression data.

Finally, in Chapter 4 we use the above algorithms to solve one of the most important problems in gene expression analysis: the identification of genes with altered expression between samples. Using time series expression data, researchers have looked at many different biological systems under a variety of conditions to identify such differentially expressed genes. We present the first general algorithm for identifying genes that are differentially expressed in two time-series expression experiments. Our algorithm overcomes a number of challenges unique to time-series data, including sampling rate differences between the reference and test experiments, variations in the *timing* of biological processes, and the lack of full repeats, by combining a noise model for individual samples with a global error difference. As we show, we have used our algorithm to compare wild type and knockout cell cycle expression experiments in yeast. Our algorithm identified 56 cycling genes as differentially expressed. These results where validated using an independent protein-DNA binding dataset, and were shown to improve upon prior methods. Surprisingly, our algorithm also identified 22 non cycling genes as differentially expressed. These genes were shown to be significantly correlated under a set of independent expression experiments, suggesting additional roles for the transcriptional factor Fkh1 and Fkh2 in controlling cellular activity in

yeast.

Chapter 2

Continuous representation of time series expression data[1]

Principled methods for interpolating gene expression time-series are needed to make such data useful for detailed analysis. Datasets measuring temporal behavior of thousands of genes offer rich opportunities for computational biologists. For example, Dynamic Bayesian Networks may be used to build models and try to understand how genetic responses unfold. However, such modeling frameworks need a sufficient quantity of data in the appropriate format. Another example is the identification of genes that are specifically involved in response to a certain disease (such as a specific infection). In order to identify such genes we need to be able to compare different expression time courses. Current gene expression time-series data often do not meet these requirements, since they may be missing data points, sampled non-uniformly, and measure biological processes that exhibit temporal variation. Thus, computational methods must be used to pre-process data derived from disparate experiments and investigators so that it may be further analyzed.

In many applications, researchers may face the problem of reconstructing unobserved gene expression values. Values may not have been observed for two reasons. First, errors may occur in the experimental process that lead to corruption or absence of some expression measurements. Second, we may want to estimate expression values at time points different from those originally sampled. In either case, the nature of microarray data makes straightforward interpolation difficult. Data are often very noisy and there are few replicates. Thus,

[1]This chapter is based on references [18] and [20]

simple techniques such as interpolation of individual genes can lead to poor estimates. Additionally, in many cases there are a large number of missing time-points in a series for any given gene, making gene specific interpolation infeasible. A particular problem arises when series are not sampled uniformly such as in [94, 32, 43].

In this section we use statistical spline estimation to represent time-series gene expression profiles as continuous curves. Our method takes into account the actual duration each time point represents, unlike most previous approaches that treat expression time-series like static data consisting of vectors of discrete samples [43, 60, 47]. Our algorithm generates a set of continuous curves that can be used directly for estimating unobserved data. However, although our method uses spline curves (piecewise polynomials) to represent gene expression profiles, it is not reasonable to fit each gene with an individual spline due to the issues with microarray datasets discussed above. Instead, we constrain the spline coefficients of genes in the same class to covary similarly, while also allowing for gene specific parameters. A class is a set of genes with similar expression profiles that may be constructed using prior biological knowledge or clustering methods. In this section we assume that the class information is given (for example, from prior biological knowledge). In Section 5 we present a clustering algorithm that extends the algorithm discussed in this section to infer classes automatically by operating directly on the continuous representations of expression profiles.

2.1 Related work

Recently, several papers have focused on modeling and analyzing the temporal aspects of gene expression data. In Holter *et al* [59] a time translational matrix is used to model the temporal relationships between different modes of the Singular Value Decomposition (SVD). Unlike our work, this method focuses on the SVD modes and not on specific genes. In addition, only relationships between time points that are sampled at the lowest common frequencies can be studied. Thus, not all available expression data can be used. In Zhao *et al* [106] a statistical model is fit to all genes in order to find those that are cell cycle regulated. This method uses a custom tailored model, relying on the periodicity of the specific dataset analyzed, and is thus less general than our approach.

Several papers have used simple interpolation techniques to estimate missing values for gene expression data. Aach *et al* [3] use linear interpolation to estimate gene expression

levels for unobserved time-points. D'haeseleer [39] use spline interpolation on individual genes to interpolate missing time-points. As we show in Section 2.5, both techniques cannot approximate the expression curve of a gene well, especially if there are many missing values. In Troyanskaya *et al* [98] several techniques for missing value estimations were explored. However, none of the suggested techniques take into account the actual times the points correspond to, and thus time series data is treated in the same way as static data. As a consequence, their techniques cannot be used to interpolate time series expression data, and to estimate values for time-points between those measured in the original experiments. Even for missing value estimation, as we show in Section 2.5, the method presented in this section outperforms the best method described in [98].

There is a considerable statistical literature that deals with the problem of analyzing non-uniformly sampled data. These models, known as mixed-effect models [28] use spline estimation methods to construct a common class profile for their input data. Recently, James and Hastie [67] presented a reduced rank mixed effects model that was used for classifying medical time-series data. In this paper we extend these methods to gene expression data. Unlike the above papers, we focus on the gene specific aspects rather than the common class profile. In addition, in Section 5 we present a method that is able to deal with cases in which class membership is not given. Another difference between this work and [67] is that we do not use a reduced rank approach, since gene expression datasets contain information about thousands of genes.

Since all the algorithms described in this chapter use splines, before we discuss the actual problems and algorithms for this analysis level, we present a short review of splines in the following subsection.

2.2 Splines

Splines are piecewise polynomials with boundary continuity and smoothness constraints. They are widely used in fields such as computer-aided design, image processing and statistics [24, 103, 45, 67]. The use of piecewise low-degree polynomials results in smooth curves and avoids the problems of overfitting, numerical instability and oscillations that arise if single high-degree polynomials were used. In this paper we use cubic splines since they have a number of desirable properties. For instance, cubic polynomials are the lowest de-

gree polynomials that allow for a point of inflection. Thus, we will restrict the subsequent discussion to the cubic case.

A cubic spline can be represented with the following equation:

$$y(t) = \sum_{i=1}^{n} C_i S_i(t) \qquad t_{min} \leq t < t_{max} \tag{2.1}$$

Here, t is the parameter (e.g., time), $S_i(t)$ are polynomials, and C_i are the coefficients. The typical way to represent a piecewise cubic curve is simply:

$$S_{4j+l}(t) = \begin{cases} t^{l-1}, & x_j \leq t \leq x_{j+1} \\ 0 \quad \text{otherwise} \end{cases} \tag{2.2}$$

Here, $l = 1 \ldots 4$, $j = 0 \ldots n/4 - 1$ and the x_j's denote the *break-points* of the piecewise polynomials. Thus, we have $n/4$ cubic polynomials that can be denoted $p_j(t) = \sum_{l=1}^{4} C_{4j+l} t^{l-1}$. In order to determine the coefficients of these polynomials, n equations are required. If one specifies a value D_j plus continuity constraints up to the second derivative for the piecewise polynomial at each break-point x_j for $j = 1 \ldots n/4 - 1$, four equations are obtained for each of the $n/4 - 1$ internal break-points:

$$\begin{aligned} p_j(x_j) &= D_j \\ p_j(x_j) &= p_{j-1}(x_j) \\ p'_j(x_j) &= p'_{j-1}(x_j) \\ p''_j(x_j) &= p''_{j-1}(x_j) \end{aligned}$$

Additionally, specifying values for the end-points $p_0(x_0) = D_0$ and $p_{n/4-1}(x_{n/4}) = D_{n/4}$ yields a total of $n - 2$ equations. Thus, in order to solve for the spline coefficients, an additional two equations are needed. Typically, these equations are obtained by specifying the first or second derivatives at the two end-points of the spline. Note that since one explicitly specifies the values $p_j(x_j)$ at the break-points, the formulation in equation 2.2 is particularly useful for defining *interpolating* splines.

2.3 B-splines

While the method discussed so far for defining cubic splines is easy to understand, it is not the most flexible or mathematically convenient formulation for many applications. Alternately, one can write a cubic polynomial in terms of a set of four normalized *basis* functions. A very popular basis is the B-spline basis, which has a number of desirable properties. The texts by Rogers and Adams [87] and Bartels *et al* [24] give a full treatment of this topic. Once again, the discussion here will be limited to features relevant to this thesis. Most significantly for the application of fitting curves to gene expression time-series data, it is quite convenient with the B-spline basis to obtain approximating or *smoothing* splines rather than interpolating splines. Smoothing splines use fewer basis coefficients than there are observed data points, which is helpful in avoiding overfitting. In this regard, the basis coefficients C_i can be interpreted geometrically as *control points*, or the vertices of a polygon that control the shape of the spline but are not interpolated by the curve. It can be shown that the curve lies entirely within the convex hull of this controlling polygon. Further, each vertex exerts only a local influence on the curve, and by varying the vector of control points and another vector of knot points (discussed below), one can easily change continuity and other properties of the curve.

The normalized B-spline basis can be calculated using the Cox-deBoor recursion formula [87]:

$$b_{i,1}(t) = \begin{cases} 1, & x_i \leq t < x_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

$$b_{i,k}(t) = \frac{(t - x_i)b_{i,k-1}(t)}{x_{i+k-1} - x_i} + \frac{(x_{i+k} - t)b_{i+1,k-1}(t)}{x_{i+k} - x_{i+1}} \tag{2.4}$$

Here, k is the order of the basis polynomials (i.e. for a cubic polynomial $k = 4$).

The values x_i are called *knots*, where $i = 1 \ldots n + k$. A *uniform* knot vector is one in which the entries are evenly spaced, i.e., $\mathbf{x} = (0, 1, 2, 3, 4, 5, 6, 7)^T$. If a uniform knot vector is used, the resulting B-spline is called periodic since the basis functions will be translates of each other, i.e., $b_{i,k}(t) = b_{i-1,k}(t - 1) = b_{i+1,k}(t + 1)$. See figure 2-1 for an example. For a periodic cubic B-spline ($k = 4$), the equation specifying the curve can be written as:

$$y(t) = \sum_{i=1}^{n} C_i b_{i,4}(t) \quad \text{for} \quad x_4 \leq t \leq x_{n+1} \tag{2.5}$$

The B-spline basis allows one to write a particularly simple matrix equation when fitting splines to a set of data points. Suppose observations are made at m time points $t_1 \ldots t_m$ giving a vector $\mathbf{D} = (D_1, \ldots, D_m)^T$ of data points. We can then write the matrix equation $\mathbf{D} = \mathbf{SC}$, where $\mathbf{C}$ is the vector of n control points and $\mathbf{S}$ is a m by n matrix where $[S]_{ij} = b_{j,4}(t_i)$. If $n = m$ (the number of control points equals the number of data points), then $\mathbf{S}$ is square and the equation may be solved by matrix inversion, yielding an interpolating spline. However, as discussed this may lead to overfitting and it is often desirable to use fewer control points than data points to obtain a smoothing or approximating spline. In this case, the matrix equation must be solved in the least-squares sense, which yields $\mathbf{C} = (\mathbf{S}^T\mathbf{S})^{-1}\mathbf{S}^T\mathbf{D}$.

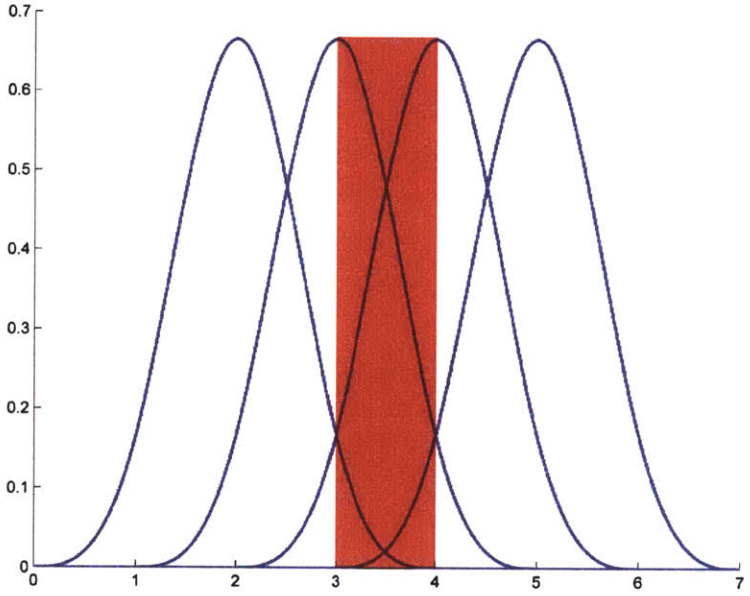


Figure 2-1: The B-spline basis functions are periodic (translates of one another) when a uniform knot vector is used. Shown here is the normalized cubic B-spline basis ($k = 4$) with knot vector $\mathbf{x} = (0, 1, 2, 3, 4, 5, 6, 7)^T$. Using the four basis functions shown, the B-spline will only be defined in the shaded region $3 \leq t \leq 4$, where all four overlap.

2.4 Estimating unobserved expression values and time points

In order to obtain a continuous time formulation, we use cubic B-splines to represent gene expression curves. As mentioned above, by knowing the value of the splines at a set of control points in the time-series, one can generate the entire set of polynomials from the

B-spline basis functions. In our formulation, the spline control points are uniformly spaced to cover the entire duration of the dataset. Once the spline polynomials are generated we can re-sample the curve to estimate expression values at any time-point. Since the control points are uniformly spaced, each data point has an effect that is related to the actual time it represents, and different time points can have different effects on the overall expression curve based on this duration.

When estimating these splines from expression data, we do not try to fit each gene individually. Due to noise and missing values, such an approach could lead to over-fitting of the data and may in general lead to estimates that are very different from the real expression values of that gene (see Section 2.5). Instead, we constrain the spline coefficients of co-expressed genes by assuming that they are drawn from the same normal distribution and thus share the same class covariance matrix. Thus we use other genes in the same class to estimate the missing values of a specific gene.

Though genes in the same class share the same covariance matrix for their spline coefficients, they can have different expression curves since we allow for gene specific variations. Such variation can represent higher or lower expression levels, offsets in peak expression time, etc. If a gene is highly represented (i.e., it does not have a lot of missing values), then the estimated curve is mostly influenced by the the observed data (via the gene specific parameters). However, for those genes with many missing values, or those with a set of consecutive missing values, the class information can be used to determine values at the missing points more effectively than other methods.

2.4.1 A probabilistic model of time series expression data

In this section we follow a method that is similar to the one used by James and Hastie [67] for classification. However, unlike their work, in this chapter we focus on gene specific aspects rather than the common class profile. This allows us to handle variations in expression levels that are caused by gene specific behavior.

A class is a set of genes that are grouped together using prior biological knowledge or a clustering algorithm. In this section we assume that class information is given. We discuss how to deal with cases in which such class information is not given in Section 5.

We represent each gene expression profile by a spline curve. For a gene i in class j, $Y_i(t)$ is the observed value for i at time t. Let q be the number of spline control points used, and

$s(t)$ the vector of spline basis functions evaluated at time t, with $s(t)$ of dimensions q by 1. Denote by μ_j the average value of the spline coefficients for genes in class j, and by γ_i the **gene specific** variation coefficients. We assume that γ_i is normally distributed vector with mean zero and the class spline control points covariance matrix Γ_j, which is a q by q matrix. Denote by ϵ_i a random noise term that is normally distributed with mean 0 and variance σ^2. According to this model, $Y_i(t)$ can be written as:

$$Y_i(t) = s(t)(\mu_j + \gamma_i) + \epsilon_i$$

This model includes both gene specific and class specific parameters. This allows us to use information from other genes in the class based on the extent to which gene specific information is missing. We restrict the missing values of a gene by requiring them to vary with the observed values according to the class covariance matrix Γ_j. Using the class average μ_j and the gene specific variation γ_i, we can re-sample gene i at any time t' during the experiment. This is done by evaluating the spline basis at time t', and setting $\widehat{Y_i(t')} = s(t')(\mu_j + \gamma_i)$.

In order to learn the parameters of this model (μ, γ, Γ and σ) we use the observed values, and maximize the likelihood of the input data. Denote by Y_i the vector of observed expression values for gene i, and by S_i the spline basis function evaluated at the times in which values for gene i were observed. If we observed a total of m expression values for i in our dataset, then S_i is of dimensions m by q. The kth row in S_i contains the spline basis functions evaluated at t_k, where t_k is the time at which the kth value was observed. According to our model, we have:

$$Y_i = S_i(\mu_j + \gamma_i) + \epsilon_i$$

where ϵ_i is now a vector of the noise terms. Note that since we are estimating the spline coefficients at the control points, each observed value has an effect related to the actual time it represents. Thus, different experiments can have different effects on the resulting curve if the expression values were sampled non-uniformly.

2.4.2 Estimating spline coefficients from experimental data

The model described above is a probabilistic model, and thus we can solve the parameter estimation problem by finding the parameters that maximize the likelihood of the input data . For our solution, we assume that the expression values for each gene were obtained independently of other genes. This assumption is not entirely true since different experimental conditions can affect multiple genes in the same experiment. However, this simplifying assumption allows for efficient computations and allows us to capture the essence of the results.

There are two normally distributed parameters in our model, the noise term ϵ and the gene specific parameters γ. Thus, the combined covariance matrix for a gene in class j can be written as:

$$\Sigma_j = \sigma^2 I + S\Gamma_j S^T$$

where S is the spline basis function evaluated at all the time points in which experiments were carried out. Given this formulation, determining the maximum likelihood estimates for our model parameters is a non-convex optimization problem (see [67]). Thus, we turn to the EM algorithm. Our probabilistic model can be factored as follows:

$$p(Y, \gamma|\sigma^2, \Gamma, \mu) = p(Y|\gamma)p(\gamma|\sigma^2, \Gamma, \mu)$$

This results in two terms, each of which depends only on one of the the variance parameters in our model. The above equation translates into the following joint probability:

$$\prod_j \prod_{i \in c_j} \frac{1}{(2\pi)^{n_i/2}\sigma^{n_i/2}} exp[-\frac{1}{2\sigma^2}(Y_i - S_i(\mu_j + \gamma_i))^T(Y_i - S_i(\mu_j + \gamma_i))] \times$$
$$\frac{1}{(2\pi)^{q/2}|\Gamma_j|^{1/2}} exp[-\frac{1}{2}\gamma_i^T \Gamma_j^{-1} \gamma_i] \quad (2.6)$$

where c is the number of classes and c_j is the set of genes in class j. Note that we need to maximize this joint probability simultaneously for all classes since the variance of the noise, σ^2, is assumed to be the same for all genes.

This representation leads to the following procedure. We treat the γ_i's as missing data and solve the maximum likelihood problem using the EM algorithm. In the E step we find the best estimation for γ using the values we have for σ^2, μ and Γ. In the M step we

maximize equation 2.6 with respect to σ^2, μ and Γ while holding the γ_i's fixed. See [50] and Section 5 for complete details.

The complexity of each iteration of the EM algorithm is $O(q(n+c*q))$ since we estimate q parameters for each gene and q^2+q parameters for each class.

2.5 Results

In this section we demonstrate the application of our method to expression time-series datasets, showing results for estimating missing values and unobserved data points. Our results make use of the cell-cycle time-series data from Spellman *et al* [94]. In that paper, the authors identify 800 genes in *Saccharomyces cerevisiae* as cell cycle regulated. The authors assigned these genes to five groups that they refer to as $G1$, S, $S/G2$, $G2/M$, and $M/G1$.

Various treatments can be used to synchronize cells. Expression experiments of the cell cycle system in yeast used a number of different synchronization methods. To test our spline interpolation algorithm we concentrated on the cdc15 dataset from Spellman *et al* [94]. This dataset contains 24 non uniformly sampled time points (see Table 1.1). We chose this dataset because it is the largest cell cycle dataset and the most challenging one as it contains non-uniformly sampled data. However, similar results were obtained for the other WT cell cycle datasets. The results presented in this section were obtained using splines with 7 control points; however, similar results were obtained for different numbers of control points.

We compared our algorithm to three other interpolation techniques that have been used in previous papers: linear interpolation [3], spline interpolation using individual genes [39], and k-nearest neighbors (KNN) with $k = 20$, which achieved the best results on static data out of all the algorithms described in [98]. In order to predict a missing value for a gene g in experiment j, the KNN method described in [98] selects n genes (in our case, $n = 20$) with similar expression values to g in all of the other expression experiments. Next, the average of expression of these genes in experiment j is computed, and this is the value assigned to gene g in experiment j.

In order to test our algorithm on a large scale we chose 100 genes at random from the set of cell-cycle regulated genes. For each of these genes we ran each estimation algorithm

four different times, hiding 1,2,3 and 4 consecutive time points, while not altering the other genes. Next, we computed the error in our estimations when compared to an estimate of the variance of the log ratios of the expression values in the following way.

Denote by $Y_i(t)$ the (hidden) expression values for gene i at time t, and by $\widehat{Y_i(t)}$ an estimated values. Denote by m the number of missing (hidden) data points and by n the number of genes that were used for the test. Denote by v the variance of the log ratios of expression values. Then the error of an estimation for m missing data points is defined as:

$$err_m = \sqrt{\frac{1}{mn}\sum_{i=1}^{n}\sum_{l=1}^{m}\frac{[Y_i(t_l) - \widehat{Y_i(t_l)}]^2}{v}}$$

If err_m is above 1 then the error is (on average) bigger than the replication variance, and vice versa. The variance v was computed using the raw expression data of the unsynchronized cells from two different time points.

Figure 2-2 shows a comparison of the error of our estimation algorithm with the three methods mentioned above. For our method, we performed two separate runs. In the first we used the class information provided in [94] (the assignment of the genes to the five different phases) and in the second we used a novel clustering algorithm described in Section 5 to obtain the class information. As can be seen in the figure, our algorithm achieves more than 10% less error when compared with the second best method (k-nearest neighbors, KNN). We have repeated this test by hiding different sets of experiments in order to compute the variance of the prediction errors. For all methods, results for hiding 2,3 and 4 consecutive points remained the same for all sets of experiments used. Results for hiding one time point varied slightly between different sets (up to 0.1 difference), though in all cases the algorithm presented in this section did better than any of the other methods mentioned above.

Interestingly, our algorithm does somewhat better when it is allowed to estimate class membership than it does when the class information is pre-specified. This can be attributed to the fact that the five classes from [94] are somewhat arbitrary divisions of a continuous cycle. Thus, for missing value estimation our clustering algorithm is able to assign more relevant class labels. Since our algorithm interpolates the input expression points, it can estimate expression values at any time point during the course of the experiments. In Figure 2-3 we present results that were obtained by hiding 1,2,3 and 4 consecutive experiments. Again, our algorithm achieves more than 15% less error than the other two techniques. Note

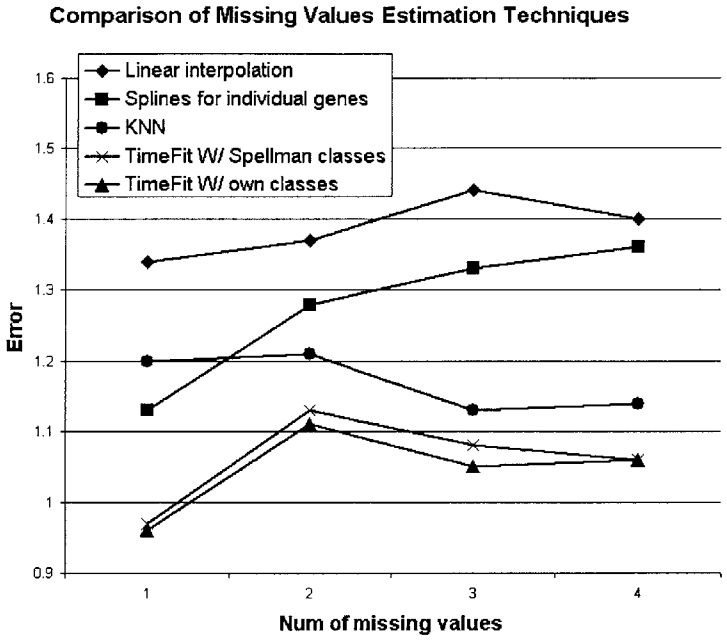


Figure 2-2: Comparison among different missing value interpolation techniques. Five different methods where compared, as discussed in the text. As can be seen, in all cases our algorithm does better than the others methods, achieving up to 15% improvement over the best previously applied methods.

that KNN cannot be used to estimate missing experiments, and thus is not included in this comparison.

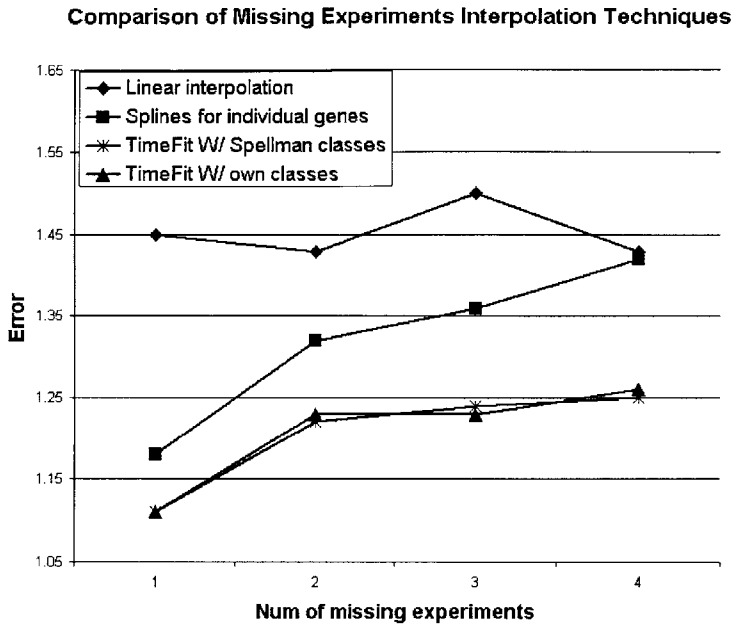


Figure 2-3: Comparison among four different methods for interpolating time series expression data. For this figure we held out complete time points, and used interpolation to predict the values of all genes for the missing time points. KNN is not an interpolation algorithm, and thus cannot be used for this task. Again, in all cases our algorithms achieved the best results.

2.6 Summary

Due to noise, missing values and differences in sampling rates, interpolation is an important first step for combining and comparing time series expression datasets. In this section we presented an interpolation method that uses mixed effects models to assign spline representation to time series expression data. This method combines class and individual gene information to learn a spline model for each gene. This allows us to use class information in places where many of the values are missing, and the gene specific information where the values are present. An additional advantage of a spline based method is that it allows us to weight each sampled point based on the actual duration it represents. This is especially important for expression datasets, since many of these datasets are non uniformly

sampled. We have presented results using real biological datasets that demonstrate that our algorithm fits the data well, and that it improves upon previous methods used for the same task.

It is important to note that our method requires data that has been sampled at a sufficiently high rate. While it works well on several such datasets, as we have shown, for other datasets that have been sampled at rates too low to capture changes in the underlying biological processes, our method will not be effective. A future direction would be to use our method to determine the quality of the sampling rate.

In the next chapter we extend the results presented in this chapter, and use the splines generated by our algorithm to continuously align time series expression datasets.

Chapter 3

Alignment[1]

In addition to different sampling rates, variability in the *timing* of biological processes further complicates gene expression time-series analysis. The rate at which similar underlying processes such as the cell-cycle unfold can be expected to differ across organisms, genetic variants, and environmental conditions. For instance, Spellman *et al* [94] analyze time-series data for the yeast cell-cycle in which different methods were used to synchronize the cells. It is clear that the cycle lengths across the different experiments vary considerably, and that the series begin and end at different phases of the cell-cycle (see Table 1.1). Thus, one needs a method to align such series to make them comparable.

In this section we present an alignment algorithm that uses the same spline representation discussed in the previous section to continuously time-warp series. First, a parameterized function is chosen that maps the time-scale of one series into another. Because we use parameterized functions, we are explicitly specifying the number of allowed degrees of freedom, which is helpful in avoiding overfitting. Our algorithm seeks to maximize the similarity between the two sets of expression profiles by adjusting the parameters of the warping function.

3.1 Related Work

To the best of our knowledge, the only previous attempt to solve the alignment problem for time series expression data is discussed in Aach *et al* [3]. In that paper, the authors present a method for aligning gene expression time-series that is based on Dynamic Time Warping,

[1]This chapter is based on references [18] and [20]

a discrete method that uses dynamic programming and is conceptually similar to sequence alignment algorithms. Unlike with our method, the allowed degrees of freedom of the warp operation in Aach *et al* depends on the number of data points in the time-series. Their algorithm also allow mappings of multiple time-points to a single point, thus stopping time in one of the datasets. In contrast, our algorithm avoids temporal discontinuities by using a continuous warping representation.

There is also a substantial body of work in the speech recognition and computer vision community that deals with data alignment. For instance, non-stationary Hidden Markov models with warping parameters have been used for alignment of speech data [38], and mutual information based methods have been used for registering medical images [101]. However, these methods generally assume high resolution data, which is not the case with available gene expression datasets.

3.2 Continuously aligning temporal expression datasets

The goal of the alignment algorithm is to warp the time-scale of one realization of a biological process into that of another. A set of genes is chosen in which the members are assumed to have the same temporal pattern of expression (e.g., from prior biological knowledge or clustering methods). A parameterized warping function is then selected and our algorithm seeks to produce an optimal alignment by adjusting the function parameters. Note that although it is possible to align individual genes this is problematic unless one has sufficiently high quality data as from replicates or a large number of time points.

Assume that we have two sets of time-series gene expression profiles, one of which we will refer to as the reference set. Denote a spline curve for gene i in the reference time series as $g_i^1(s)$, where $s_{min} \leq s \leq s_{max}$. Here, s_{min} and s_{max} are the starting and ending points for the reference time series respectively. Similarly, we will denote splines in the set to be warped as $g_i^2(t)$ for $t_{min} \leq t \leq t_{max}$. Define a mapping $T(s) = t$, which transforms points in the reference scale into the time-scale of the set to be warped (see Table 3.1 for a summary of the parameters used by our alignment algorithm). In this chapter, we use a linear transformation $T(s) = (s - b)/a$, with a the stretch/squash parameter and b the translation. However, more complex transformations could be used in our framework. We

define the alignment error e_i^2 for each gene as:

$$e_i^2 = \frac{\int_\alpha^\beta [g_i^2(T(s)) - g_i^1(s)]^2 ds}{\beta - \alpha} \tag{3.1}$$

where $\alpha = \max\{s_{\min}, T^{-1}(t_{\min})\}$ is the starting point of the alignment, and $\beta = \min\{s_{\max}, T^{-1}(t_{\max})\}$

g^1, g^2 - splines for the two datasets
s -reference time scale
t - warped set time scale
T - a function from s to t
α, β - start and end points of the alignment
a, b - stretch (a) and shift (b) parameters

Table 3.1: The parameters used by our alignment algorithm.

is its end point. The error of the alignment for each gene is proportional to the averaged squared distance between the curve for gene i in the reference set and in the set to be warped. In order to take into account the degree of overlap between the curves, and to avoid trivial solutions such as mapping all the values in the curve to a single point, we divide this error by the time-length of the overlap $\beta - \alpha$. Thus, our goal is to find parameters a and b that minimize e_i^2. As discussed, we suggest minimizing the error for a set of genes. We define the error for a set of genes S of size n as:

$$E_S = \sum_{i=1}^{n} w_i e_i^2 \tag{3.2}$$

The w_i's are weighting coefficients that sum to one; they could be uniform ($1/n$) or used for unequal weighting. For instance, if one wishes to align wildtype time-series expression data with knockout data, many of the genes' expression patterns are expected to be unchanged in the two experiments. However, a subset of the genes may be highly affected. In this case, we want to down-weight the contribution of such genes, since they are not expected to align well. One way of formulating this is to require that the product $w_i e_i^2$ be the same for all genes. That is, the higher the error, the lower the weight we assign to that gene so that the weight is inversely proportional to the error. This requirement translates into a maximization problem for the inverse squared error, as we show below.

From $w_i e_i^2 = K$, we get that $E_S = nK$ and so we can deduce that:

$$w_i = \frac{K}{e_i^2} \Rightarrow \sum_i w_i = \sum_i \frac{K}{e_i^2} \Rightarrow K = \frac{1}{\sum_i 1/e_i^2} = \frac{E_S}{n}$$

since $\sum_i w_i = 1$. As before, the objective is to minimize E_S, or in this case equivalently to maximize $\sum_i 1/e_i^2$.

Minimization of E_S must be done numerically, since a closed form solution is not possible. In the linear case presented, we are only searching for two parameters, so we minimize E_S directly using standard non-linear optimization techniques. We use the Nelder-Mead simplex search method (available in the Matlab package), which does not use gradients and can handle discontinuities. For the linear warping case, the essential constraints are that $\alpha < \beta$ and $a > 0$. Since the use of a numerical optimization method does not guarantee convergence to a global minimum, multiple random re-starts may be necessary. This leads to an algorithm with running time $O(rmnq^2)$, in which r is the number of random re-starts, m is the number of iterations for convergence, n is the number of genes in S, and q is the number of spline control points used.

If a large number of genes are to be aligned, we suggest the following algorithm to reduce the computation time. Begin by choosing a random subset of fixed size (e.g., 50 genes) and random initial settings for the warping parameters from a uniform distribution. The minimization procedure is then carried out and this process is repeated with a new random choice of warping parameters for a set number of iterations. Upon termination, the alignment parameters that correspond to the minimum error are chosen. These parameters are then used as the starting conditions for the E_S minimization using the full set of genes. See Section 3.3 for experimental results on how this reduces the running time on gene expression datasets.

3.3 Results

To test our alignment algorithm, we used time series expression datasets from experiments that studied the yeast cell cycle. We aligned the three yeast WT cell-cycle gene expression time-series from Table 1.1. We shall refer to these datasets as cdc15DS, cdc28DS, and alphaDS in this section. These datasets clearly occur on different time-scales and begin in different phases Since its the longest, and contains the most time points, cdc15DS was

used as a reference set and the alphaDS and cdc28DS were aligned against it using a linear warping $T(s) = (s - b)/a$ and the full set of cell-cycle regulated genes as identified in [94]. For the cdc28DS, we obtained $a = 1.42$ and $b = 2.25$ with $E_S = 0.1850$. These results indicate that the cdc28DS cell-cycle runs at approximately 1.4 times the speed of the cdc15DS cycle and starts approximately 5.5 minutes before (i.e., we calculate $T(10)$ since cdc15DS starts at 10 minutes). For the alphaDS, we obtained $a = 1.95$ and $b = -5.89$ with $E_S = 0.1812$. Figure 3-1 shows the aligned/unaligned expression values for the $G1$ and $S/G2$ clusters for the cdc28DS to cdc15DS alignment. Alignment for each dataset took approximately 5.5 minutes on a 1 GHz Pentium III machine using our algorithm that performs initial alignments on smaller subsets of genes; the alignments took approximately 45 minutes without this improvement. In Figure 3-2 we present alignment results for three

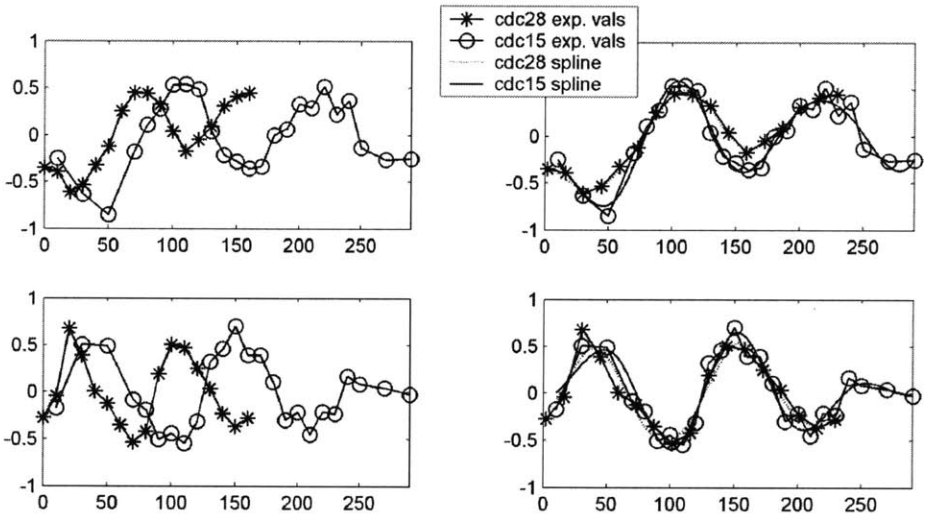


Figure 3-1: Alignment of genes from the cdc28DS to cdc15DS. Linear warping was used with the full set of cell-cycle regulated genes. The left-hand side shows class-averages of unaligned expression values for two clusters. The top row shows aligned results for the $G1$ cluster (186 genes) and the bottom row the $S/G2$ cluster (283 genes). These results indicate that the cdc28DS cell-cycle runs at approximately 1.4 times the speed of the cdc15DS cycle and starts approximately 5.5 minutes before.

cell cycle regulated genes in the three experiments discussed above. As can be seen, using only the input values for each gene, the three profiles (under the different conditions) look very different. However, when using the aligned continuous representation we arrive at a very good agreement between the three experiments for all three genes. To validate the

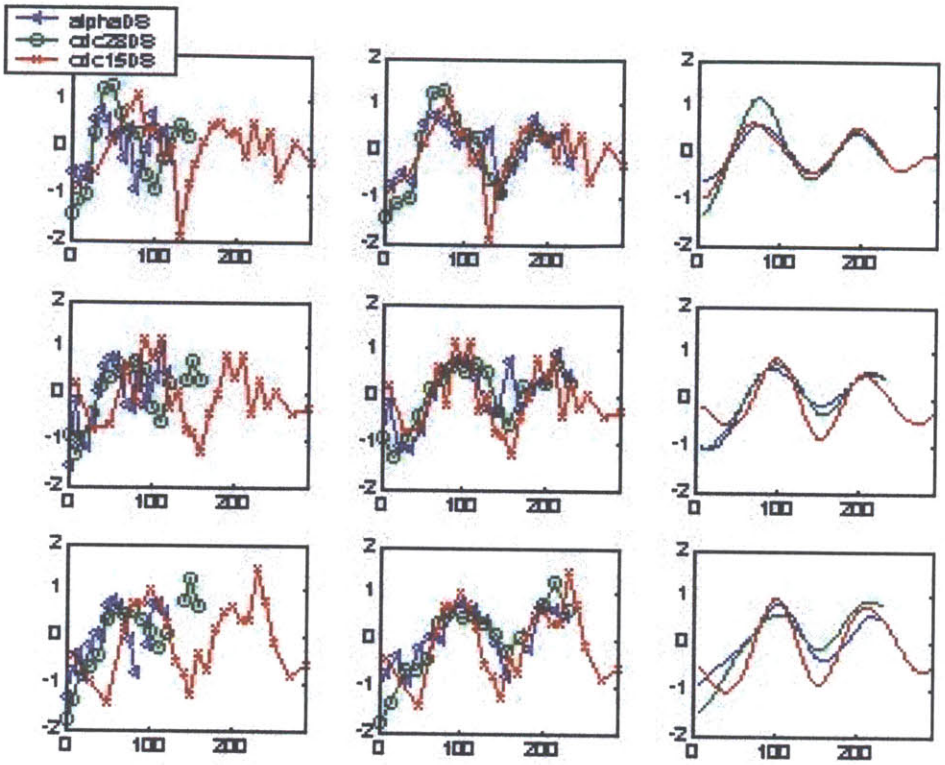


Figure 3-2: Alignment of three different genes for the three WT cell cycle experiments. Left column: input values. Middle column: aligned input values using the temporal mapping function for the three datasets. Right column: aligned splines computed for these genes. As can be seen, by computing the alignment parameters the expression profiles start to match. When we introduce the aligned continuous representation, we are also able to overcome missing values and noise, resulting in a good agreement between the three experiments for all genes.

quality of these alignments we performed two analyses: 1) alignments of genes in alphaDS against genes in cdc15DS with gene identity of those in cdc15DS randomly permuted and 2) alignments of alphaDS to cdc15DS using different numbers of genes. Note that for brevity only the alphaDS was used; we chose this dataset because it is the smallest and presumably demonstrates worst-case results. For the first analysis, we performed 200 trials giving E_S scores between 0.2562-0.3554, with 50% of the scores lying between 0.2780-0.3129. These results suggest that the actual alphaDS to cdc15DS E_S score of 0.1812 would not arise by a chance alignment of the genes. For the second analysis, we sampled subsets of between 5-400 genes 100 times from the full set of cell-cycle regulated genes (Table 3.2). This

# genes	a	std a	b	std b
5	1.80	0.42	-7.70	16.41
10	1.89	0.21	-5.06	21.5
25	1.93	0.10	-4.99	7.07
50	1.93	0.13	-5.13	9.03
100	1.96	0.03	-6.86	2.42
200	1.95	0.02	-6.38	1.76
400	1.96	0.02	-6.12	1.30

Table 3.2: Results of experiments in which random subsets of fixed size were sampled 100 times and alignment of alphaDS and cdc15DS were performed. The columns are as follows: number of genes used, stretch/squash parameter, standard deviation of this parameter, offset parameter, and standard deviation of this parameter. This analysis shows that the variance in the parameters decreases as more genes are used and there is convergence to the a and b settings found with the full set of genes.

analysis shows that the variance in the parameters decreases as more genes are used and there is convergence to the a and b settings found with the full set of genes. Interestingly, our algorithm is usually able to find the "actual" a and b parameter settings even when relatively small numbers of genes are used.

Thus, these analyses give evidence that our algorithm can reliably align the cell-cycle datasets. These results compare favorably with those in Aach *et al* [3] using the same data. In their case, they found that their actual alignment score was not at a low percentile when compared against alignments using randomized data (gene values shuffled). Further, they indicate that poor results were obtained with small cluster sizes (an analysis over a wide range of sizes was not presented in their paper). The fact that our method uses a continuous representation and fits only two parameters to all the genes helps to explain its good performance on the cell-cycle data. However, one must be careful in extrapolating these results, since they are clearly dependent on the underlying dataset.

3.4 Summary

In this section we have described an algorithm that aligns a continuous representation of two or more time series expression experiments to overcome problems related to the timing of biological processes, and differences in the starting times of the measurements. Our algorithm uses cubic splines to represent time series expression profiles, and the alignment

is performed by fitting a temporal mapping function that minimizes the square integral between the two curves. Using three wild type cell cycle expression dataset, we have shown that this algorithm obtains both biological and computational significant results. This method allows the direct comparison of two gene expression profiles, and in the next section we use this alignment algorithm as a pre-processing step for identifying genes that are differentially expressed between two expression experiments.

Chapter 4

Identifying differentially expressed genes in time series expression data[1]

One of the key issues in time series gene expression analysis is the identification of genes with altered expression between samples. For instance, one would like to identify genes that have changed significantly after an experimental treatment or that differ between normal and diseased cells. In clinical research, such differentially expressed genes can serve as disease specific markers or as predictors of the clinical outcome of a treatment [99, 104, 78, 62]. In knockout experiments, differentially expressed genes represent first or second order downstream effects of the knocked out gene [107, 82], and their identification allows the discovery of genetic interaction networks. Other experiments have looked at temporal expression changes when cells are treated with various agents [49] in order to identify different expression programs in the cell.

Recently, a number of algorithms for analyzing various features of time series expression data have been introduced, but none of these algorithms are directly applicable to detecting genes that are differentially expressed. As mentioned in Sections 2 and 3, a number of algorithm for interpolating, aligning and clustering time series expression data have been described in the past. Clustering algorithms are useful for identifying patterns in a single expression experiment, but they cannot be directly used to compare two time series

[1]This chapter is based on reference [19]

experiments. Interpolation and alignment algorithms are important pre-processing steps, however, it is not clear how they can be used to identify differentially expressed genes. In this section we use the interpolation and alignment methods from Sections 2 and 3 as a pre-processing step.

4.1 Overview of our algorithm

We present a general algorithm that fully exploits information in time-series gene expression data to detect differentially expressed genes. Our algorithm represents the two expression profiles to be compared with aligned continuous curves, and computes a global difference measure between these two curves. This allows us to use a difference measure that takes into account the entire curves, and not just the sampled points. In addition, such a method requires a minimum of unwarranted assumptions, since we do not assume anything about the underlying form of the data. In order to determine the significance of this global difference, we use a noise model for individual samples to find a curve that best explains this difference. By basing our comparison on a maximum likelihood assignment instead of the original profile, our algorithm will err on the conservative side. That is, only global difference values that cannot be explained by the "best" (maximum likelihood) curve will be considered significant. Another advantage of our algorithm is that it combines a noise model for individual samples (which is easy to compute) with a global error measurement that captures the temporal difference between two expression profiles. Thus, it can assign significance to temporal expression differences, while requiring only a minimum number of expression measurements repeats. This latter point is important, since repeating time-series experiments can be prohibitively expensive and in most publicly available data sets there are no, or very few, full repeats [94, 107, 78].

We applied our algorithm to the comparison of time-series data of cell-cycle dependent gene expression in wild-type [94] and knockout [107] yeast. Our algorithm identified a larger set of 56 differentially expressed genes than reported in the previous analysis by Zhu *et al* [107]. In addition, our algorithm was able to identify 22 non-cell cycle regulated genes as differentially expressed. We showed that this set of genes is significantly correlated in a set of independent expression experiments. Interestingly, some of these genes were found to be cycling in the knockout experiments while they are flat in the wild type experiment.

4.2 Related work

Many algorithms have already been introduced for identifying genes differentially expressed between two experiments in the static expression case [52, 41]. However, due to differences in sampling rates, and variations in the timing of biological processes (see Table 1.1), such methods cannot be directly applied to most time series expression datasets. Previously reported algorithms for identifying differentially expressed genes in time series data either essentially applied static analysis methods, used ad-hoc methods that are not generally applicable, or were highly tailored for a specific data set. Previously reported methods include cluster analysis [107, 49], generalized SVD [7], point wise comparison [78, 62] and custom tailored models [104]. While these methods have achieved some success, they suffer from many problems. As we demonstrate in the results section, cluster analysis fails to detect differentially expressed genes that belong to clusters for which most genes do not change. Generalized SVD (presented by Alter *et al* [7]) can be used to detect differences between sets of genes, but they are not appropriate for comparing individual genes. Further, these methods require that the datasets being compared contain the same number of experiments (or time points), which is clearly not the case in general (see Table 1.1). In addition, these algorithm cannot deal directly with differences that result from the difference in the timing of biological processes, which requires us to align the datasets prior to comparing them. Direct point-wise comparison of samples, essentially a static analysis method, is problematic because it does not take into account the dynamic nature of the experiments, and is unable to distinguish between systematic changes and random noise. Further, due to the inconsistencies in time series data obtained from different sources mentioned above, in many cases direct point-wise comparison is not possible. Finally, custom tailored models clearly do not present a general solution, as these require significant assumptions about the shape of the expression profiles being compared (e.g., linear or quadratic models). In most cases, *a priori* knowledge that would justify using highly specific models is unavailable. Even in cases in which some genes are known to change in a certain way over time (e.g., a sinusoidal model for the cell cycle), using a highly specific model for the shape of expression profiles will result in missing changes in many genes that are not behaving according to the assumed model.

4.3 Hypothesis testing for differentially expressed genes

Following spline assignment and alignment (as discussed in Sections 2 and 3), each gene is represented by two continuous curves (one for each experiment). Denote the first (reference) curve as C_1 and the second (test) curve as C_2 (for example, C_1 could be the wild-type expression profile, while C_2 is a knockout profile). Given C_1 and C_2, we would like to answer the following question: Is the difference between the two expression profiles for a certain gene significant or not? This problem can be formulated as a hypothesis testing problem, with the following two hypothesis:

H_0: C_2 is a noisy realization of C_1.

H_1: The two curves are independent.

Under the null hypothesis, we assume that C_2 can be represented by the same spline curve as C_1, and that any difference between the two profiles is a result of noise in the measurement of the test experiment. Under the alternative hypothesis we assume that both C_1 and C_2 can be represented by a spline curve, though we do not assume anything about the relationship between the two curves.

The hypothesis test can be performed by looking at the ability of each hypothesis to explain the difference between the two curves. Using log a likelihood ratio test this could be written as:

$$2\log\frac{p(C_2|C_1,H_1)}{p(C_2|C_1,H_0)} \tag{4.1}$$

While there are established statistical methods for comparing two curves, these methods will not work well in our case. Unlike regular curves, the expression profile curves were derived from very few sample points, and this fact should be taken into account when computing the significance of the difference between the curves. In addition, most methods require additional information about the curves (such as a noise model for the entire curve), which is not available in our case due to the small number of full repeats. Thus, we present a method that allows us to compute these conditional probabilities, even when only few repeats exist.

4.3.1 Noise model for individual samples

Here we assume that noise in individual measurements is normally distributed with mean 0 and variance σ^2 (see Section 4.3.5 for a discussion on how to incorporate expression-value dependent variance into our algorithm). Since noise in *individual* expression measurements is assumed to be independently varying, σ^2 can be computed even if few repeats exist. Denote by Y_1 and Y_2 the actual expression values measured in the reference and test experiments respectively. Let Y_1^t the expression value at time t in the reference experiment. Then $p(x|Y_1^t, \sigma^2)$ is the probability of obtaining expression measurement x at time t. Comparing Y_1 and Y_2 directly is not possible because of their different sampling rates and temporal expression variations. However, we can sample C_2 at the same time points as the actual reference experiment to obtain a set of values that are comparable to Y_1. Let $t_1 \ldots t_m$ be the set of time points that were measured in the reference experiment. For a curve C, Denote by $C(t)$ the value of C at time t. Set $Y_2' = \{C_2(t_1) \ldots C_2(t_m)\}$, i.e., Y_2' is the vector of points sampled from C_2 at the reference experiment points. To compute the conditional probability under hypothesis one ($p(C_2|C_1, H_1)$), we use Y_2', and recall that under H_1, C_1 and C_2 are independent. Thus, we can set

$$p(C_2|C_1, H_1) = p(Y_2'|\sigma^2, H_1) = \frac{1}{(2\pi\sigma^2)^{m/2}}$$

where we set the means at the sampled points, and the number of sampled points determine the number of degrees of freedom.

While a sample based method works well for the alternative hypothesis, under the null hypothesis (H_0), using samples from C_2 suffers from a number of drawbacks. First, such a method does not capture *global* differences in expression, since it ignores systematic differences between the curves (for example, if one were always higher or lower). Second, in many cases sampling rates for time-series data are non-uniform. Using a sampling based method, we assign an equal weighting to each sampled point, which does not reflect the actual time each point represents. In the next subsection, we introduce a method for computing $p(C_2|C_1, H_0)$ that overcomes these difficulties.

4.3.2 Combining a sample noise model with global difference

Instead of directly using samples from C_2 to compute $p(C_2|C_1, H_0)$, we use the global difference between the two expression curves C_1 and C_2, which is defined as:

$$D(C_2, C_1) = \frac{\int_{v_s}^{v_e} [C_2(t) - C_1(t)]^2 dt}{V}$$

Here, v_s and v_e are the start and end of the interval in which the two curves can be compared (the alignment interval). and $V = v_e - v_s$. Note that $D(C_2, C_1)$ is proportional to the averaged squared distance between the two curves (similar to Equation 3.1 we used for our alignment algorithm). This is a suitable difference measure for the following reasons. First, $D(C_2, C_1)$ depends on the actual duration over which the curves can be compared, and is thus less sensitive to sampling rates. In addition, $D(C_2, C_1)$ can distinguish between consistent differences and random oscillations around the reference samples, and is thus sensitive to actual systematic differences (see Section 4.4). Finally, by relying on a scalar global difference measure, we reduce to minimum the number of unwarranted assumptions, since we do not assume anything about the underlying form of the data. While hypothesis testing on multiple parameters would be advantageous if we had strong *a priori* knowledge about a model for expression changes (e.g., quadratic profiles), in the absence of such information the single global difference measure we introduce requires a minimum number of unwarranted assumptions.

We now discover a new curve, C, that best explains the difference between C_1 and C_2. Let $e^2 = D(C_1, C_2)$. Setting $p(C_2|C_1, H_0) = p(e^2|Y_1, \sigma^2, H_0)$ leads to framework that combines the individual error model (σ^2) with a global difference measurement (e^2) which, as discussed above, correctly captures the differences between the two curves. In other words, $p(e^2|Y_1, \sigma^2, H_0)$ is an induced probability measure over deviations $e^2 = D(C_1, C_2)$ for all possible realizations of C_2.

For a curve C, set $Y_C = \{C(t_1) \ldots C(t_m)\}$. In order to find the maximum likelihood assignment of $p(e^2|Y_1, \sigma^2, H_0)$, we need to find a curve C with the same global distance (e^2) from C_1 that maximizes the probability that C is a noisy realization of C_1. Formally, this

could be stated as the following maximization problem:

$$max_{Y_C}(p(Y_C|Y_1,\sigma^2)) \quad s.t. \quad D(C,C_1) = e^2 \tag{4.2}$$

That is, we are looking for a curve C which satisfies the global error constraint ($D(C,C_1) = D(C_1,C_2)$), such that when sampling from C at time points that where used in the reference experiment (Y_C), we get values that are as close as possible to the values measured in the reference experiment. Using the maximum likelihood assignment instead of simply the original C_2, guarantees that the computations of the significance of differential expression will err on the conservative side. That is, only a global error value e^2 that cannot be adequately explained by the "best" (maximum likelihood) curve C will be considered significant.

4.3.3 Solving the maximization problem

The maximization problem in equation 4.2 can be solved by working on the spline representation for each curve, and re-writing equation 4.2 in terms of the spline control points. In order to show how to solve this problem we introduce additional notation. Recall that the expression curves consist of sets of cubic splines. As mentioned in Section 2.2, splines can be fully specified by a set of control points. Let F_1 be the set of control points for C_1, and F_C be the set of control points for a curve C. Since F_1 and F_C fully specify C_1 and C, we can use the notation $D(F_C,F_1)$ instead of $D(C,C_1)$. In addition, based on our model we can write $Y_1 = SF_1$ [1], and $Y_C = SF_C$. Since individual noise terms are normally distributed, it can be shown (by taking the log, and ignoring constant terms) that maximizing $p(Y_C|Y_1,\sigma^2)$ is equivalent to minimizing $(Y_1 - Y_C)^T(Y_1 - Y_C)$. Thus, the above maximization problem can be written as the following quadratic minimization problem:

$$min_{F_C}(S(F_1 - F_C))^T(S(F_1 - F_C)) \quad s.t. \quad D(F_C,F_1) = e^2 \tag{4.3}$$

This problem can be re-written as a quadratic minimization problem, as shown in the following lemma.

[1] Note that we omit the noise term from this equation since we are now using the predicted reference splines.

Lemma 4.3.1 *The following minimization problem:*

$$min_{F_C}(S(F_1 - F_C))^T(S(F_1 - F_C)) \quad s.t. \quad D(F_C, F_1) = e^2$$

can be written as:

$$min_{\delta}(S\delta)^T(S\delta) \quad s.t. \quad \delta^T A\delta = 1$$

where A is a positive definite matrix.

Proof: First, we need to explicitly represent $D(C, C_1)$ using F_C and F_1. This could be done as follows:

$$D(C, C_1) = D(F_C, F_1) = \frac{\int_{v_s}^{v_e}(s(t)[F_1 - F_C])^2 dt}{V}$$

where $s(t)$ is the spline basis function evaluated at time t. Set $\delta = F_1 - F_C$, then we have $(S(F_1 - F_C))^T(S(F_1 - F_C)) = (S\delta)^T(S\delta)$. As for the integration, denote the number of splines (or the number of piecewise polynomials) by q, and let $p_0 \ldots p_q$ be the set of start and end points for the individual splines (that is, the first spline starts at $p_0 = v_s$ and ends at p_1 and so on). Thus, the integration part can be re-written as:

$$\int_{v_s}^{v_e}(s(t)[F_1 - F_C])^2 dt = \sum_{i=0}^{q-1}\int_{p_i}^{p_{i+1}}(s(t)\delta)^2 dt = \sum_{i=0}^{q-1}\delta^T(\int_{p_i}^{p_{i+1}}(s(t)s(t)^T)dt)\delta$$

Since $s(t)$ is continuous polynomial between p_i and p_{i+1}, we can evaluate the integral in the above equation. Note that this integral is actually a matrix, and, as just mentioned, each entry in that matrix can be evaluated. Set $B_i = (\int_{p_i}^{p_{i+1}}(s(t)s(t)^T)dt$ and let $B = \sum_i B_i$, then $\int_{v_s}^{v_e}(s(t)\delta)^2 dt = \delta^T B\delta$. Note that B is positive semi definite, since the integral on the right hand side measures the squared distance between two splines. To show that B is positive definite, we note that if $\delta \neq 0$, then the two sets of control points differ in at least one entry. Using a B-spline basis function, each polynomial is only supported by four control points. Since the basis for a cubic polynomial is of degree four, two identical cubic polynomials cannot be represented by two different sets of four control points. Thus, at least in the segment corresponding top this polynomial the two curves differ, and since the integral measures the squared distance between the two curves, the result will be greater than 0.

Setting $A = B/(V + e^2)$ proves the lemma since both V and e^2 are positive values. ■

Next, we use Lagrange multipliers to show that the solution is a vector δ, such that δ is the eigenvector with the smallest eigenvalue for the matrix $A^{-1}S^TS$, as we prove in the following lemma.

Lemma 4.3.2 *Let δ he the eigenvector with the smallest eigenvalue for the matrix $A^{-1}S^TS$, then δ (appropriately scaled), is the solution to the following minimization problem:*

$$min_\delta (S\delta)^T(S\delta) \quad s.t. \quad \delta^T A\delta = 1 \tag{4.4}$$

Proof: Using Lagrange multipliers we can write: $L = (S\delta)^T(S\delta) - \lambda\delta^T A\delta - \lambda$. Taking the derivative w.r.t. δ, and setting $\frac{\partial L}{\partial \delta} = 0$ we get: $S^TS\delta = \lambda A\delta \Rightarrow A^{-1}S^TS\delta = \lambda\delta$. Thus, δ is an eigenvector of $A^{-1}S^TS$. Multiplying both sides of the above equation by $\delta^T A$ we get: $\delta^T S^TS\delta = \lambda\delta^T A\delta = \lambda$, where the last equality results from our constraint ($\delta^T A\delta = 1$). Since $\delta^T S^TS\delta$ is the quantity we wish to minimize, λ must be the smallest eigenvalue of $A^{-1}S^TS$, and δ should be the eigenvector corresponding to that eigenvalue, appropriately scaled so that $\delta^T A\delta = 1$. ■

Now, set $F_\delta = F_1 - \delta$ and $Y_\delta = SF_\delta$. Based on the discussion above, we can now compute $p(C_2|C_1, H_0) = p(Y_\delta|Y_1, \sigma^2)$.

4.3.4 The complete algorithm

Figure 4-1 presents the complete algorithm we use for identifying differentially expressed genes in time-series data. The input to the algorithm is the set of genes G, the two expression datasets, E_1 and E_2 and a significance threshold ϵ. Following spline assignment and alignment, we solve equation 4.4 for each gene, and use the solution Y_δ to compute the log likelihood ratio as follows. First, as discussed above we set $p(C_2|C_1, H_1) = p(Y_2'|\sigma^2, H_1)$, which can be written as $p(Y_2'|Y_2', \sigma^2)$ since under H_1, C_2 represents the mean curve for the second experiment. For H_0 we have $p(C_2|C_1, H_0) = p(Y_\delta|Y_1, \sigma^2)$, and thus the log likelihood ratio evaluates to:

$$2\log\frac{p(C_2|C_1, H_1)}{p(C_2|C_1, H_0)} = 2\log\frac{e^{-\frac{(Y_2'-Y_2')^T(Y_2'-Y_2')}{2\sigma^2}}}{e^{-\frac{(Y_1-Y_\delta)^T(Y_1-Y_\delta)}{2\sigma^2}}} = \frac{(Y_1 - Y_\delta)^T(Y_1 - Y_\delta)}{\sigma^2} \tag{4.5}$$

Next, to perform a significance test, we use the chi-square distribution with q degrees of freedom (where q is the number of spline control points used by the curves). It is also possible to incorporate our confidence in the reference curve into the p-value computation, as we show in Appendix A.1.

DiffExp(G, E_1, E_2, ϵ) {
  For all genes $i \in G$
    Compute spline assignment (C_1^i, C_2^i) and control points(F_1^i, F_2^i) for i in both experiments
  Align the two datasets
  For all genes $i \in G$\{
    $e_i^2 = D(C_1^i, C_2^i)$
    Let Y_δ be the solution for equation 4.4 using e_i^2 and F_1^i
    Set $r = \frac{(Y_1 - Y_\delta)^T (Y_1 - Y_\delta)}{\sigma^2}$
    $s = 1-$ cdf of the chi-square distribution for r with q d.o.f
    If $s < \epsilon$ output i
  }
}

Figure 4-1: Complete algorithm for identifying differentially expressed genes in time series data.

The computational complexity of this algorithm is linear in the number of genes we are testing. Spline assignment and alignment can be done in linear time (see Sections 2 and 3). Next, for each gene we need to solve the maximization problem from equation 4.2. As shown in Lemma 4.3.2 this can be done by finding eigenvectors for the matrix $A^{-1}S^TS$. The dimensions of this matrix depend only on the number of points sampled for each gene, and the number of spline control points used. Both these values can be considered as constants with respect to the number of genes (typical numbers are less than 30 time points and 10 control points, while there are over 6000 yeast genes). Thus, we can solve the maximization problem in a constant time, and the total running time is linear in the number of genes.

As we show below, the above algorithm can be modified so that it can use variances that depend on expression value magnitudes. It is also possible to present a symmetric version of this algorithm. See Appendix A.1 for details.

4.3.5 Value specific variance

So far we have assumed that all expression values have the same variance, σ^2. In practice, we have found that the variance of expression measurements depends on the magnitude of

expression values or fold changes (see Figure 4-2). Taking this fact into account is especially

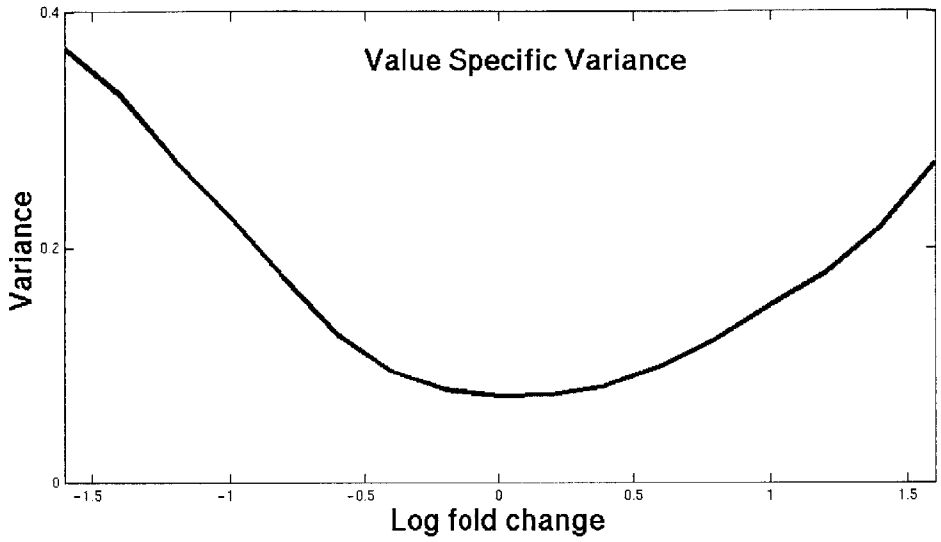


Figure 4-2: The variance associated with different log fold change values. This variance was computed from the 2 repeats in the fkh1/2 [107]. As can be seen, the higher the absolute log fold change, the higher the associated variance. Since most values are relatively low, using the same variance for all values results in an under estimate of the variance for genes that have large changes during the cell cycle. Instead, we incorporate the value specific variance into our algorithm, as discussed in this appendix.

important for time series data, since small shifts in the magnitude of expression values can result in large global differences for genes with high fold change values. Thus, taking value specific variance into account reduces the effect of experimental artifacts and associated high variance, allowing us to accurately detect significant changes and ignore changes that are a result of noise.

Our framework for combining an individual error model with a global difference measurement (equation 4.2) can be modified to use variances that depend on expression value magnitudes. Instead of maximizing $p(Y_C|Y_1,\sigma^2)$ we maximize $p(Y_C|Y_1,\sigma_1^2\ldots\sigma_m^2)$ where $\sigma_1^2\ldots\sigma_m^2$ are the m expression value specific variances for the samples in Y_1. Recall that the rows of S (the spline basis function matrix) correspond to the time points that were sampled in the reference experiment. Denote by S_i the i^th row of S. Let $S_i' = S_i/\sigma_i$. Then maximizing $p(Y_C|Y_1,\sigma_1^2\ldots\sigma_m^2)$ is equivalent to minimizing $S'(F_1 - F_C))^T(S'(F_1 - F_C)$, and we proceed by replacing S with S' in equation 4.4. This results in the differential weighting of the individual errors around Y_1, leading to a reduction in the effect that experimental

artifacts and associated high variance can play in determining differential gene expression.

In order to compute the value specific variance for a value x we use the following method. Let R_1 and R_2 be two repeats of the same experiment, and let θ denote a weighting coefficient (for the results presented in this chapter we use $\theta = 0.25$), which allows us to control the range of values that will contribute to the computation of the variance. For $r_1^i \in R_1$ let $r_2^i \in R_2$ be the corresponding repeat in the second experiment. Set

$$p(r_1^i) = \frac{1}{\sqrt{2\pi\theta^2}} e^{(r_1^i - x)^2/(2\theta^2)}$$

and $P = sum_i p(r_1^i)$. Then the value specific variance for x, v_x is computed by setting:

$$v_x = \sum_i \frac{(r_1^i - r_2^i)^2}{P}$$

That is, v_x is computed by using a Gaussian bump around the selected value (x), and weighting the contribution of the different repeats based on their distance from x.

4.4 Results

We have tested our algorithm using synthetic data and cell cycle expression data. As we show below, our algorithm generated correct results for the synthetic data and biologically meaningful results for the expression datasets we used. In both cases, our algorithm improved upon prior methods for comparing time series expression datasets.

4.4.1 Synthetic data

In order to test our algorithm, to determine significance thresholds under a variety of expression profiles and noise models, and to compare it to previous methods that work by comparing individual points we first tested our algorithm on synthetic data. We generated four sets of samples $Y1, Y2, Y3, Y4$ as follows (see Figure 4-3). $Y1$ consisted of a set of uniform samples from a sinusoid between 0 and 4π. $Y2$ was generated by adding random noise (normally distributed with mean 0) to $Y1$. $Y3$ was generated by adding a positive value, b, to the values in $Y1$. Finally, we set $Y4 = aY1$ where $0 < a < 1$. The parameters a and b were selected such that the mean absolute error of all sets with respect to $Y1$ was equal. While $Y2$ is a noisy realization of $Y1$, $Y3$ and $Y4$ represent consistent

additive or multiplicative differences. If the input set is normalized appropriately, such consistent differences over time might represent real biological change. For example, in cell cycle experiments, genes that show a reduced cycling profile are probably effected by the experimental condition and should be detected.

We repeated the process of generating $Y2, Y3$, and $Y4$ 1000 times for each of several sampling rates (sampling rates were chosen to be similar to those used in prior actual expression experiments), and was various noise variances. For all the different sampling rates and noise models, our algorithm correctly identified the $Y2$ samples as a noisy realization of $Y1$. As the sampling rate increased, so did the ability of our algorithm to correctly detect consistent changes ($Y3$ and $Y4$) as differentially expressed (see [16]). To compare our results with methods that work directly on the input samples, we have also performed the hypothesis testing with the actual samples by replacing Y_δ with $Y2, Y3, Y4$. In Figure 4-4 we present the results of the two methods using 24 samples (similar to the sampling rates of [94]) and a noise model derived from time series repeats from Zhu *et al* [107]. While in all cases our algorithm correctly identified $Y2$ as a noisy realization of $Y1$, the sample based method identified 90% of the $Y2$ samples as differentially expressed. Since in time series experiments most genes do not change, such a high false detection rate cannot be tolerated. In addition, our algorithm was able to detect more of the $Y3$ and $Y4$ curves, since the number of degrees of freedom it uses is smaller than the number used by the sample based method. Similar results where obtained with other sampling rates and different noise models.

4.4.2 Yeast cell cycle and knockout data

To test our algorithm on biological datasets and to compare our algorithm with algorithms that have been used in the past, we used a dataset from Zhu *et al* [107]. These authors performed an experiment in which two yeast transcriptional factors (Fkh1 and Fkh2) involved in regulating the cell cycle were knocked out and a time-series of gene expression levels was measured in synchronized cells. Focusing on 800 previously identified cell cycle regulated genes (from Spellman *et al* [94]), the authors used hierarchical clustering to compare their results with a time series wild-type (WT) dataset from Spellman *et al* [94]. Using this method Zhu *et al* identified two clusters (Clb2 and Sic1) that contain genes affected by the knockout. They were able to demonstrate direct binding of Fkh2 only to promoters of

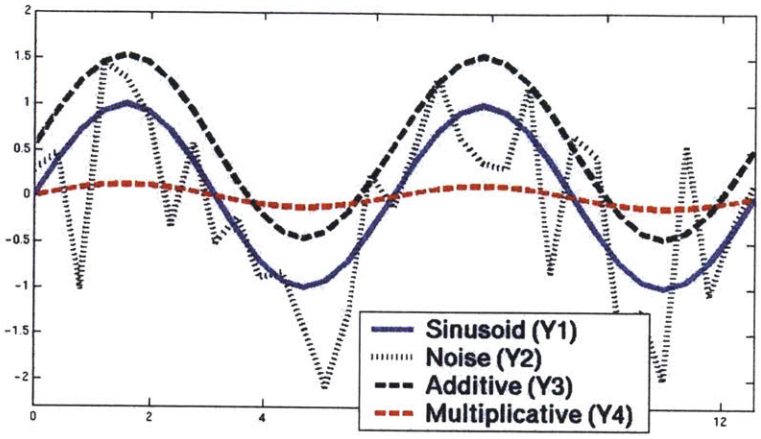


Figure 4-3: The four different sets we generated to test our algorithm. See text for details.

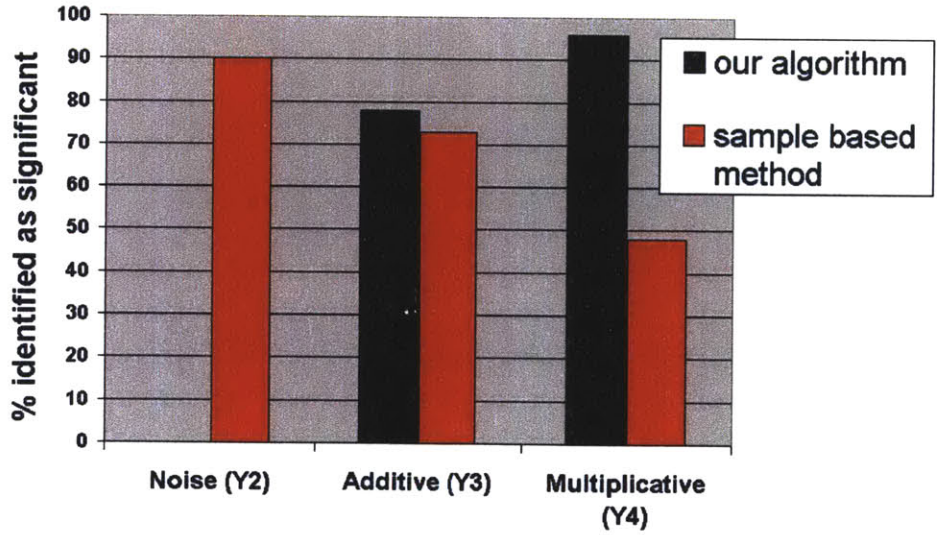


Figure 4-4: Comparison with methods that work directly on the input samples. In all cases our algorithm correctly identified $Y2$ as noisy realization of $Y1$. In contrast, sampling based methods detected 90% of the $Y2$ samples as differentially expressed. Such high false detection rate cannot be tolerated in time series experiments. In addition, our algorithm detected more of the multiplicative curves ($Y4$) as differentially expressed, indicating that the low false positive rate does not come at the expense of high true positive rate.

genes from the clb2 cluster and therefore they suggest that the forkhead proteins affect the Sic1 cluster genes indirectly through Swi5 and Ace2. Note that since the two experiments used different sampling rates, direct point-wise comparison of the samples is not possible. Independently, Simon *et al* [92] used DNA-binding experiments to identify genes that are regulated by nine cell cycle transcription factors, including Fkh1, Fkh2, Swi5 and Ace2. We applied our algorithm to the two time series data sets (WT and mutant) and have used the binding data to verify our results.

Identifying differentially expressed genes controlled by Fkh1 and Fkh2

We used our algorithm to identify genes that are differentially expressed in the Fkh1/Fkh2 knockout experiment, when compared to the alpha WT experiment. Of the 800 cell cycle regulated genes, our algorithm identified 56 genes as differentially expressed with a p-value $< .005$ (see Appendix 4.3.5 for a discussion about the value specific variance used).

In Table 4.1 we present the top 30 genes (in decreasing significance) identified by our algorithm. As can be seen, many of the genes identified by our algorithm are confirmed by independent binding experiments, and prior literature. While many of the genes come from the two primary phases that are either directly controlled by Fkh1/Fkh2 (G2/M) or indirectly controlled (M/G1), there are a number of genes from other cell cycle phases suggesting a role for the forkhead transcription factors in G1 and S phases. These results are supported by the genome-wide binding data that describes association of Fkh1 and Fkh2 with genes expressed in G1 and S (Simon *et al* [92]). In order to verify our results on a global scale, we computed the percentage of genes in our list that are bound by each of the nine cell cycle factors (using a binding p-value cutoff of 0.005), and compared this result to the percentage obtained from using the entire set of 800 cell cycle regulated genes. We then computed a p-value for the enrichment of each factor for differentially expressed genes using the hyper-geometric distribution. We expected to find enrichment for three types of factors:

1. Fkh1 and Fkh2 which should bind directly to regulated genes.
2. Swi5 and Ace2 which should bind the promoters of the indirectly regulated genes.
3. Mcm1 and Ndd1 which are co-factors of Fkh2 in the regulation of G2/M genes (Koranda *et al.* [70]) and therefore should bind at least a subset of the Fkh2 target genes.

As can be seen in Figure 4-5, the set of genes identified by our algorithm agree very well

Differentially expressed cycling genes

Gene name	P-value	Phase	Previously identified?	comments
Cwp1	$1 * 10^{-16}$	S/G2	No	bound by Fkh2 and Ace2
Cts1	$1.4 * 10^{-15}$	G1	Yes	bound by Fkh1 and Fkh2
Ole1	$3.5 * 10^{-14}$	M/G1	No	bound by Swi5
Egt2	$1.7 * 10^{-10}$	M/G1	Yes	bound by Ace2 and Swi5
Scw11	$1.8 * 10^{-10}$	G1	Yes	bound by Fkh1, Fkh2, Ace2 and Swi5
YER124C	$8 * 10 * -9$	G1	Yes	bound by Fkh1, Fkh2 and Ace2
Ald6	$1.2 * 10^{-8}$	S/G2	No	
YHR143W	$2.7 * 10^{-8}$	G1	Yes	bound by Fkh1, Fkh2 and Ace2
Pho5	$3.4 * 10^{-8}$	G2/M	No	
YLR194C	$3.5 * 10^{-8}$	M/G1	No	bound by Swi5
YBR158W	$5.5 * 10^{-8}$	M/G1	Yes	bound by Fkh1, Fkh2, Ndd1, Ace2 and Swi5
YNL058C	$9.7 * 10^{-8}$	G2/M	Yes	bound by Ndd1 and Mcm1
Clb2	$4.4 * 10^{-7}$	G2/M	Yes	bound by Fkh1, Fkh2, Ndd1 and Mcm1
Dip5	$5.1 * 10^{-7}$	G2/M	No	
YPL158W	$5.3 * 10^{-7}$	M/G1	No	bound by Swi5
YNL078W	$8.5 * 10^{-7}$	M/G1	No	bound by Ace2 and Swi5
Pry3	$9.1 * 10^{-7}$	G1	Yes	bound by Fkh1, Fkh2, Ace2 and Swi5
Utr2	$1.34 * 10^{-6}$	M/G1	No	bound by Fkh1, Fkh2 and Mcm1
YDR055W	$1.36 * 10^{-6}$	M/G1	No	bound by Swi5
YGL184C	$1.5 * 10^{-6}$	S	No	
Clb1	$1.8 * 10^{-6}$	G2/M	Yes	
Pbi2	$2.6 * 10^{-6}$	S	No	
Sic1	$7.5 * 10^{-6}$	M/G1	No	bound by Swi5
Pho11	$1.53 * 10^{-5}$	G2/M	No	bound by Fkh1
Pho12	$1.77 * 10^{-5}$	G2/M	No	
Mnn1	$2.1 * 10^{-5}$	G1	No	
Bud9	$2.4 * 10^{-5}$	G1	Yes	bound by Fkh1, Fkh2, Mcm1, Acc2 and Swi5
Pry1	$4.6 * 10^{-5}$	G2/M	No	bound by Fkh2 Ndd1, Mcm1 and Ace2
Pir1	$4.7 * 10^{-5}$	M/G1	Yes	bound by Mcm1 and Swi5
Ash1	$5.3 * 10^{-5}$	M/G1	Yes	bound by Swi5

Table 4.1: Top 30 differentially expressed cell cycle genes identified, ordered by significance p-value. Phase based on assignment by Spellman *et al* [94]. Previously identified column based on a list of genes extracted from Zhu *et al* [107]. Binding information taken from Simon *et al* [92] using a p-value cutoff of 0.005. Note that many of the genes in this list that are bound by one of the effected factors were not identified using cluster based method.

with the binding data. All of the above factors were significantly enriched for genes in the identified set; on the other hand, there was no enrichment for binding of Swi4, Mbp1 and Swi6. Overall, the expression changes of 37 out of the 56 genes (66%) can be explained by the binding of the six factors listed above (p-value $= 4 * 10^{-}11$). A number of other genes can be explained using prior knowledge (see Table 4.1). In order to examine the different

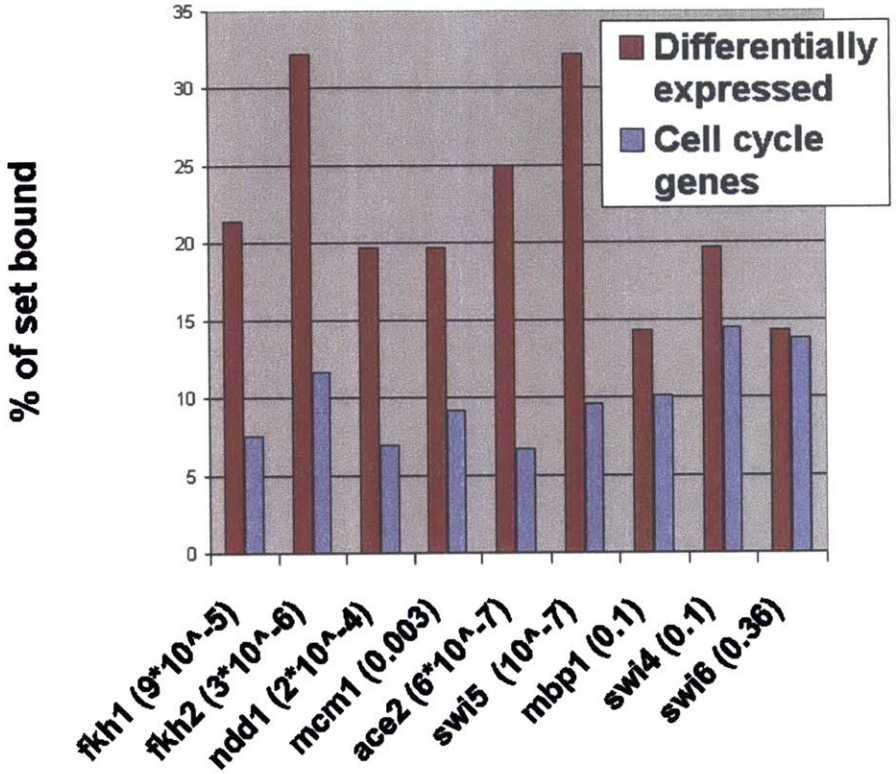


Figure 4-5: Differentially expressed cell cycle genes. The y axis is the percentage of genes bound by the nine factors in the entire set of 800 cell cycle regulated genes and in the set identified by our algorithm. Values in parenthesis following factors names are the p-value for the enrichment of each factor for differentially expressed genes. These values were computed using the hyper-geometric distribution. As can be seen, all the relevant factors are significantly enriched for genes in the selected set.

ways in which these 56 genes were affected, we clustered their expression profiles in the knockout experiment (using the K-means algorithm) into three different groups. Most of the genes (45 out of 56) belonged to cluster 1. As can be seen in Figure 4-6, this cluster had a flat profile, indicating that genes in this cluster lost their cycling ability in the knockout experiment. The second and third clusters contained fewer genes (8 and 3), and represent

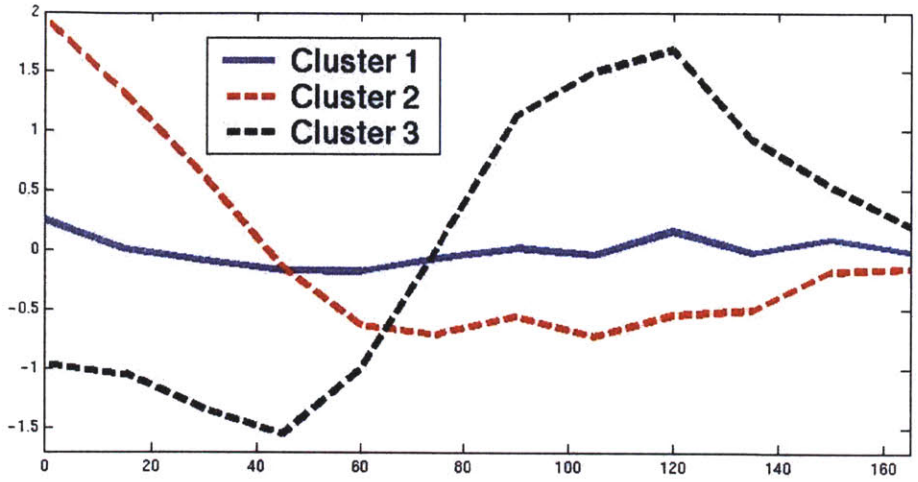


Figure 4-6: Results of clustering the 56 cell cycle genes identified by our algorithm as differentially expressed. Cluster 1 is composed of genes that are directly affected, while clusters 2 and 3 contain genes with second order effects. See [16] for the list of genes in each of these clusters, and for their expression profiles in the knockout experiment. See text for discussion.

either loss of cycling early on (cluster 2) or seeming participation in only one cycle, instead of the two that are covered by the experiment duration. Interestingly, 8 out of the 11 genes in the second and third clusters (over 70%) were bound by Swi5 or Ace2 (compared with only 10% in the entire set of 800 cycling genes), suggesting that these clusters are composed of genes with expression changes caused by second order downstream effects of the knockout of Fkh1/2. In contrast, genes in the first cluster were more likely to be bound by Fkh1/2 or one of there 2 co-factors (44% versus 20% in the entire set of 800 cycling genes).

Comparison with clustering based methods

As mentioned above, due to the differences in sampling rates, and to the different cell cycle durations (see Table 1.1) in the two time series experiments, all previously reported methods for identifying differentially expressed genes other than clustering based analysis cannot be applied to these data sets. Thus, we compared our results with the list of 42 genes identified in the original paper by Zhu *et al* using cluster analysis. The two lists overlapped in 21 genes. As can be seen in Table 4.1 many of the genes identified by our algorithm and not detected by hierarchical clustering seem to be controlled by one of the factors effected by

the knockout, indicating that they were correctly identified by our method. In addition, many of them seem to be losing there cycling ability (see top row of Figure 4-7). It is likely

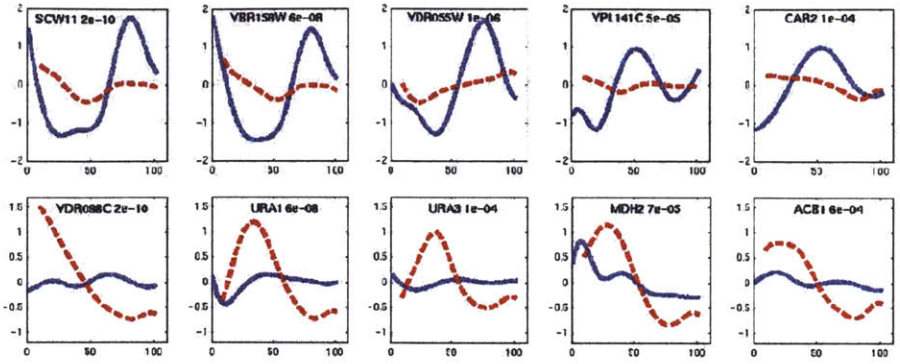


Figure 4-7: Genes identified by our algorithm that were missed by the clustering method used in [107]. Top: Five of the cell cycle regulated genes. WT expression is represented by the solid line, and knockout by the dashed line. A P-value appears to the right of the gene name. As can be seen, all of these genes displayed significant reduction in their cycling ability. In addition, all of the above genes are bound by either Fkh1/2 or by Ace2 or Swi5, indicating that our algorithm identified a relevant set of genes. Bottom: Five of the non-cycling genes. Note that some of these genes seem to be cycling in the knockout experiment while they are not cycling in the WT experiment, see text for discussion.

that these differences are missed when using cluster analysis because most of the genes in their clusters did not significantly change. We have also looked at the 21 genes that were detected by Zhu *et al* and were not detected by our algorithm. Most of these genes belonged to two clusters (CLB2 and SIC1) that were identified by Zhu *et al* as composed of differentially expressed genes. However, unlike other genes in these clusters (that were also detected by our algorithm), following alignment, many of the genes that were not identified by our algorithm do not seem to be changing in expression between the two experiments (see [16] for figures). Further, unlike the set identified by our algorithm, these genes did not show significant binding enrichment for the five factors mentioned above.

Analysis of non-cell cycle regulated genes

Though the main function of the Fkh1/2 transcription factors is in regulating the cell cycle, they are also involved in other functions including mating type switching and cell morphology. We have used our algorithm to identify differentially expressed genes in the set of 5000 genes that are not cell cycle regulated. Due to the size of this set, we have

used a more stringent p-value of .0001 (thus, in random data we would expect only 5 genes to be identified as significantly differentially expressed). Our algorithm identified 22 genes as significantly changing in the knockout data. We note that Zhu *et al* did not identify any non cycling genes as differentially expressed, perhaps because of the limitation of the cluster analysis method. In a sharp contrast with the cycling genes, all except one

Differentially expressed non cycling genes

Gene name	**P-value**	**comments**
YDR098C	$1.9*10^{-}10$	
YER037W	$4.7 * 10^{-8}$	
Ura1	$6.4 * 10^{-8}$	
YCR007C	$2 * 10^{-7}$	
Gdh2	$3.6 * 10^{-7}$	
Cit1	$1.8 * 10^{-6}$	
YOL164W	$1.4 * 10^{-5}$	
YFR026C	$1.4 * 10^{-5}$	
Ade5,7	$4 * 10^{-5}$	
Mdh2	$7.5 * 10^{-5}$	
Ura3	$1.5 * 10^{-4}$	
Ctr1	$1.8 * 10^{-4}$	
Cit2	$1.9 * 10^{-4}$	
YNL279W	$2.2 * 10^{-4}$	bound by Dig1, Ste12
YLR162W	$4.3 * 10^{-4}$	
Fig1	$5.1 * 10^{-4}$	bound by Swi5
Hxt6	$5.2 * 10^{-4}$	
Fre7	$5.4 * 10^{-4}$	
Pot1	$5.5 * 10^{-4}$	
Acb1	$6.2 * 10^{-4}$	
YMR040W	$9.1 * 10^{-4}$	
Ade1	$9.9 * 10^{-4}$	

Table 4.2: The 22 non cycling genes identified by our algorithm as differentially expressed, ordered according to their significance p-value. Binding data for Dig1 is from Lee *et al* [73]. As discussed in the text, most of these genes are significantly correlated with yeast response to stress in a set a stress related expression experiments.

of the promoters of the effected non cycling genes are not bound by any of the cell cycle transcription factors (see table 4.2), suggesting that these genes are not controlled directly by the forkhead proteins or by their direct targets (Swi5 and Ace2). Fkh1/2 double null mutation has global affects on cell growth; the cells shows pseudohyphal and invasive growth phenotypes, unusual cell morphology and slow growth rates (Hollenhorst *et al.* [58]; Zhu et al. [107]). Thus, some of the changes in gene expression in the mutant cells may be due

to the overall changes in the cell rather then to the direct effects of Fkh1/2. Following the method of Hughes *et al* [63] we searched a dataset of over 500 expression experiments for those experiments in which the set of genes identified by our algorithm were significantly correlated (either down or up regulated). We have identified 20 such experiments (see Appendix A.2). We have carried out a randomization analysis to test the significance of these results, and concluded that if random sets of equal size are used, no experiments are selected (see Appendix A.2). This indicates that the set of genes identified by our algorithm are significantly associated with the experiments selected by this method. The experiments in which these genes where significantly correlated fall mainly into the following categories: Red/Ox stress, response to alpha factor, response to zinc depletion and starvation. These finding suggests that:

1. Our algorithm finds relevant sets of differentially expressed genes.
2. Fkh1/2 may be involved in cellular pathways that are connected to the conditions under which these experiments were carried out.
3. Since some of the non cycling genes are cycling in the knockout experiment while they are flat under WT conditions (see Figure 4-7), the effects of the identified conditions may vary along the cell cycle.

Our ability to raise such hypothesis indicates the importance of principled algorithms for the analysis of time series gene expression data.

4.5 Summary

In this section we have described a principled algorithm for identifying genes with altered expression between two time series datasets. This algorithm is useful for identifying interactions between genes, selecting genes that can serve as markers for a certain disease and for determining subsets of genes that react differently to changes in experimental conditions.

One of the main advantages of this algorithm is its ability to work with few (or no) experimental repeats. By combining a noise model for individual samples with a global error difference we are able to assign significance to differences in expression profiles without computing complete statistics for these expression curves.

We have used transcription factor knockout data to test our algorithm, and to show that it correctly detects differentially expressed genes that were not detected using prior methods.

In addition, we have shown that by focusing on the set detected by our algorithm, we can correctly detect first and second order effects of the experimental condition. The main advantage of the dataset we used is that our results could be validated using protein-DNA binding data. Our algorithm can be used to analyze many biological systems, including infectious and other diseases, and cell behavior under different treatments which have been studied using time series expression data. For such systems, there is usually no independent high-throughput data source that can be used to validate sets of differentially expressed genes. Thus, it is essential to use principle computational methods that were shown to produce correct results, such as the algorithm described in this paper, when analyzing such systems.

Part II

Clustering and Ordering

A major challenge in gene expression analysis is effective data organization and visualization. Because of the large number of genes that are profiled in each experiment, clustering is needed to provide a global overview of the experiment results.

Time series expression data introduces a number of new challenges. First, when clustering non uniformly sampled time series data we should take into consideration the actual duration each point represents. In addition, when clustering such data we are also interested in the relationships between the clusters, and not only in the clusters themselves.

In this section we present a number of algorithms that address these challenges. In Chapter 5 we present a clustering algorithm that operates directly on the continuous representations of gene expression profiles. As we demonstrate, this algorithm is particularly effective when applied to non-uniformly sampled data. In Chapter 6 we discuss an algorithm for optimally ordering the leaves of a hierarchical clustering tree. This algorithm allow us to correctly determine gene expression programs that cannot be revealed by using the random ordering of the hierarchical clustering algorithm. Finally, we extend our optimal leaf ordering algorithm to address a number of problems related to the binary hierarchical clustering algorithm. More specifically, hierarchical clustering is very sensitive to noise, it usually lacks of a method to actually identify distinct clusters, and produces a large number of possible leaf orderings of the hierarchical clustering tree. In Chapter 7 we propose a new hierarchical clustering algorithm which reduces susceptibility to noise, permits up to k siblings to be directly related, and provides a single optimal order for the resulting tree. Our algorithm constructs a k-ary tree, where each node can have up to k children, and then optimally orders the leaves of that tree. By combining k clusters at each step our algorithm becomes more robust against noise and missing values. By optimally ordering the leaves of the resulting tree we maintain the pairwise relationships that appear in the original method, without sacrificing the robustness.

Chapter 5

Clustering continuous curves[1]

When clustering time-series expression data one should pay spacial attention to non uniformly sampled datasets such as in [94, 32, 43]. Unlike static data in which the samples are independent measurements taken from different individuals [52], in time series expression data, the different points are all measurements of the same continuous process. When clustering non uniformly sampled datasets that were performed to study uniform processes (such as the cell cycle [94, 82] or circadian clock [95, 80]) it is important to take into account the duration each point represents, since points representing longer durations should be weighted higher than those representing shorter ones.

Many clustering algorithms have been suggested for gene expression analysis (see [84] for a review of a number of these algorithms). However, as far as we are aware, all these algorithms treat their input as a vector of data points, and do not take into account the actual times at which these points were sampled. For example, K-means and self organizing maps [96] place the same weight on all input points, and treat them as independent samples. In contrast, our algorithm weights time points differently according to the sampling rate.

In this section we extend our splines framework from Section 2 so that it can be used when the class information is not given. This allows us to cluster the continuous representation of each expression profile. Instead of operating on the set of sampled points, our clustering algorithm uses the spline control points. These control points are uniformly spaced, and each one is influenced by the sampled points based on the duration these points represent. Thus, our clustering algorithm implicitly assigns weights to the sampled points

[1]This chapter is based on references [18] and [20]

based on the duration they represent.

We have used both synthetic and real data to test our algorithm. As we show in Section 5.2, working on the continuous representation of the input data allows our algorithm to improve upon previous clustering algorithms.

5.1 Model based clustering algorithm for temporal data

The algorithm described in Section 2 assumes that the class information is known in advance. Though this information is sometimes available, either from previous knowledge or from a classification algorithm [94], in most cases such information is not known (for example, see [78, 49]). In this section we describe a new clustering algorithm that simultaneously solves the parameter estimation and class assignment problems.

TimeFit(Y, S, c, n) {
  For all classes j {
    choose a random gene i
    $\mu_j = (S_i^T S_i)^{-1} S_i^T Y_i$ // initialize class center with a random gene
  }
  Initialize Γ, σ^2, γ arbitrarily
  Repeat until convergence {
    E step:
      for all genes i and classes j
        $p(j|i) \leftarrow \frac{p_j p(Y_i|\mu_j, \Gamma_j, \gamma_{i,j}, \sigma^2)}{\sum_k p_k p(Y_i|\mu_k, \Gamma_k, \gamma_{i,k}, \sigma^2)}$
    M step:
      for all genes i and classes j, Find the MAP estimate of $\gamma_{i,j}$
      Maximize Γ, σ^2, μ w.r.t. $P(j|i)$ // see text for complete details
      for all classes j, $p_j \leftarrow \frac{1}{n} \sum_i^n p(j|i)$ // update the class probability
  }
}

Figure 5-1: Estimating the model parameters without class information. The posterior probabilities $P(j|i)$ can be used for clustering as described in the text.

Instead of the fixed class model described in Section 2.4, here we assume a mixture model. Thus, we can model the gene expression vector for gene i in the following way. First, we select a class j for gene i uniformly at random. Next, we sample γ_i using class j's covariance matrix Γ_j, and sample a noise vector ϵ_i using σ^2. Finally we construct Y_i by setting:

$$Y_i = S_i(\mu_j + \gamma_i) + \epsilon_i$$

In Figure 5-1 we present TimeFit, our spline fitting algorithm that performs class as-

signment. The number of desired classes, c, is an input to TimeFit. Initially all classes are assumed to have the same prior probabilities, though it is easy to modify this algorithm if one has a prior knowledge about the different classes. TimeFit begins by choosing for each class one gene at random, and using this gene as an initial estimate for the class center (or average of the spline coefficients). We now treat the class assignments as the missing variables, and iterate using a modified EM algorithm. In the E step we estimate for each gene i and class j the probability that i belongs to class j, $P(j|i)$, using the values we obtained for the rest of the parameters in the M step. In the M step, instead of equation 2.6 from Section 2 we now maximize our parameters for each class using the class probability ($P(j|i)$) as computed in the E step. In addition, we now treat the $\gamma_{i,j}$'s as parameters, and find their MAP (maximum a posteriori) estimate, which is then used in the E step. In the next section we describe in detail the modified EM algorithm we use in TimeFit. TimeFit increases the likelihood monotonically, and is terminated when the parameters converge. When algorithm converges, for each gene i we discover the class j such that $p(j|i) = max_{1 \leq k \leq c} P(k|i)$ and assign i to this class. Using this "hard" clustering, when we need to re-sample gene i's expression curve (either for missing values estimation or for new time points) we use the estimated class js parameters $(\gamma_{i,j}, \mu_j)$ and continue as described in section 2.

As with most EM algorithms, TimeFit is sensitive to the initial random assignment of class centers (μ_js). Thus, we recommend that the algorithm will be repeated a number of times (in the results presented in this chapter we have re-run it five times for each dataset). Out of these runs, we select the parameter assignment that achieves the highest log likelihood for the complete model (see below), and report it as the final result of the algorithm.

5.1.1 The modified EM algorithm

We now present the details of the modified EM algorithm simultaneously infers the class assignment and the parameters of our model (see also [50] for initial derivation of our spline assignment algorithm from Section 2). We start with the complete log likelihood of our

model given by:

$$\sum_i \log(\sum_j p_j Z(j|i) \frac{1}{\sigma^{n_i}} exp[-(Y_i - S_i(\mu_j + \gamma_{i,j}))^T (Y_i - S_i(\mu_j + \gamma_{i,j}))/2\sigma^2] \frac{1}{|\Gamma_j|^{1/2}} exp[-\frac{1}{2}\gamma_{i,j}^T \Gamma_j^{-1} \gamma_{i,j}]) \tag{5.1}$$

where j is the class index and n_i is the number of observed values for gene i. $Z(j|i)$ is an (unobserved) binary indicator variable that assigns each gene to exactly one class.

In the E step we compute the expected values for $Z(j|i)$:

$$p(j|i) = E(Z(j|i)|Y_i) = \frac{p_j e^{-(Y_i - S_i(\mu_j + \gamma_{i,j}))^T (Y_i - S_i(\mu_j + \gamma_{i,j}))/\sigma^2} e^{-\frac{1}{2}\gamma_{i,j}^T \Gamma_j^{-1} \gamma_{i,j}}}{\sum_k p_k e^{-(Y_i - S_i(\mu_k + \gamma_{i,k}))^T (Y_i - S_i(\mu_k + \gamma_{i,k}))/\sigma^2} e^{-\frac{1}{2}\gamma_{i,k}^T \Gamma_k^{-1} \gamma_{i,k}}}$$

Where k goes over all possible classes, and p_k is the prior probability of class k.

In the M step we first find the MAP estimate for $\gamma_{i,j}$ by setting:

$$\gamma_{i,j} = (\sigma^2 \Gamma_j^{-1} + S_i^T S_i)^{-1} S_i^T (Y_i - S_i \mu_j)$$

Next, we maximize σ^2, μ and Γ w.r.t. the class assignment probabilities computed in the E step:

$$\sigma^2 = \frac{\sum_i \sum_j p(j|i)(Y_i - S_i(\mu_j + \gamma_{i,j}))^T (Y_i - S_i(\mu_j + \gamma_{i,j}))}{\sum_i n_i}$$

μ_j is computed by setting:

$$\mu_j = (\sum_i p(j|i) S_i^T S_i)^{-1} (\sum_i p(j|i) S_i^T (Y_i - S_i \gamma_{i,j}))$$

Then we set Γ_j to:

$$\Gamma_j = \frac{\sum_i p(j|i)[\gamma_{i,j} \gamma_{i,j}^T + (\hat{\Gamma_j}^{-1} + S_i^T S_i/\sigma^2)^{-1}]}{\sum_i p(j|i)}$$

For TimeFit, the complexity of each iteration of the EM algorithm is $O(cq(n+q))$ since we now estimate $c + cq$ parameters for each gene (q for each class).

5.2 Results

In order to explore the effect that non-uniform sampling can have on clustering we generated two synthetic curves as follows. The first curve, f_1 is obtained using the equation $f_1(x) = \sin x$. The second curve is given by the following equation:

$$f_2(x) = \begin{cases} \sin x & : \quad x \leq \pi \\ \sin x + (x - \pi)/(20\pi) & : \quad x > \pi \end{cases}$$

We sampled each curve 64 times between $-\pi$ and π and then sampled between π and 5π (the remaining portion of the curve) at different rates of either every $\pi, \pi/2, \pi/4$ or $\pi/16$. Note that since all curves were sampled between $-\pi$ and 5π, the maximal difference between the sampled values (amplitude of the curves) is at most 0.2. For each different sampling we generated 100 vectors from each curve, and added random noise (normally distributed with mean 0 and variance 0.2). Next we used our TimeFit algorithm, and compared the results to those of the Gaussian mixture clustering algorithm (that is, the probabilistic version of K-mean). Gaussian mixtures is a clustering algorithm that assumes a mixture model and tries to assign genes to classes using the class centers (see [84] for details). Unlike our algorithm, this Gaussian mixtures algorithm treats all points in the same way, and does not use the actual times they represent. As can be seen in Figure 5-2, the lower the sampling rate, the larger the difference between the performance of TimeFit and Gaussian mixtures. For example, for the sampling rate of π, Gaussian mixtures does only slightly better than chance, while TimeFit has a much higher classification success.

Next we tested TimeFit on the cdc15 dataset (see Table 1.1) and compared the results to Gaussian mixtures. For both algorithms we generated five classes. When analyzing the results we used the Spellman clusters as the gold standard, and determined how many clusters in our results correspond to these clusters. The results are presented in Table 5.1. As can be seen, four out of the five clusters that were generated by TimeFit correspond to Spellmans' clusters, containing at most two neighboring phases with 10 or more genes (the fifth contained genes from three consecutive phases). Since we are dealing with cell cycle data, the clusters defined in [94] can only have arbitrary boundaries and thus joining two of them is reasonable. On the other hand, in the k-means clustering result, cluster 4 contains more than 20 genes from four different phases, while cluster 5 contains more

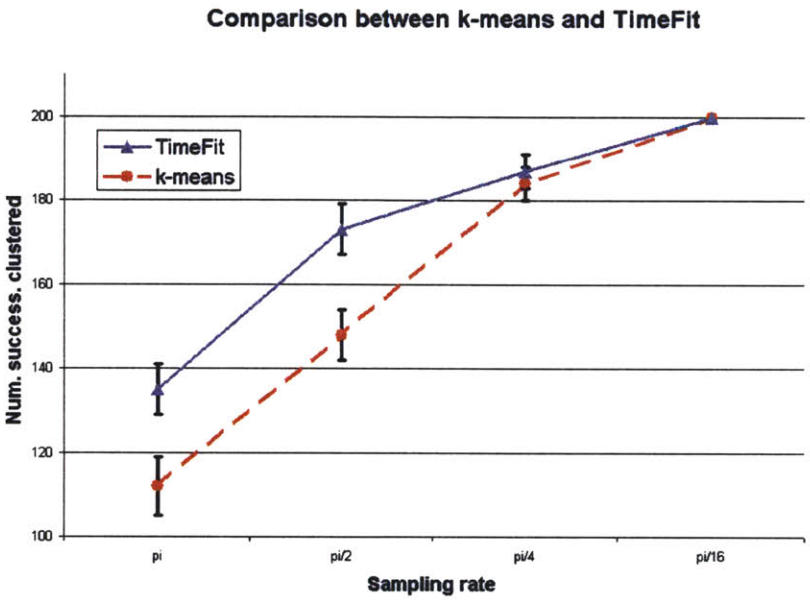


Figure 5-2: A comparison between Gaussian mixture and TimeFit for clustering the vectors from f_1 and f_2. The success rate was determined by the total number of correctly clustered vectors out of the 200 vectors. For both algorithms we have used the result that obtained the highest likelihood for each of the sampling rates. As can be seen, the lower the sampling rate, the greater the advantage of using our algorithm.

than 20 genes from three different phases. Thus, the results of our clustering algorithm are in better correspondence with existing biological knowledge than those of the Gaussian mixtures algorithm.

	TimeFit					Gaussian mixtures				
Phase / Cluster	1	2	3	4	5	1	2	3	4	5
G2/M	121	52	4	0	8	117	6	1	60	1
M/G1	2	62	33	8	1	6	51	3	45	1
G1	0	9	166	40	71	0	24	125	104	33
S	0	1	2	0	65	0	0	3	7	58
S/G2	30	3	4	0	81	18	0	0	22	78

Table 5.1: Comparison between Gaussian mixtures and TimeFit using the cdc15DS. For both algorithms we have used the result that obtained the highest likelihood. As can be seen, in most cases each of the clusters generated by TimeFit contains genes from 1 or 2 neighboring phases. For k-means there is one cluster (4) that contains more than 20 genes from 4 different phases.

5.3 Summary

In this section we have discussed an algorithm that extends our splines framework to clustering. Unlike previous clustering algorithms, our algorithm works on the continuous representation of the expression profile. This allows us to overcome problems related to non uniform sampling rates, and to produce clustering results that are superior to methods that work directly on the input points. In the next chapter we address another issue related to clustering time series expression data: Determining the relationships between clusters.

Chapter 6

Optimal leaf ordering for hierarchical clustering trees[1]

Hierarchical clustering is one of the most popular methods for clustering gene expression data. Hierarchical clustering assembles input elements into a single tree, and subtrees represent different clusters. Thus, using hierarchical clustering one can analyze and visualize relationships in scales that range from large groups (clusters) to single genes. However, the ordering of the leaves, which plays an important role in analyzing and visualizing hierarchical clustering results, is not defined by the algorithm. Thus, for a binary tree, any one of the 2^{n-1} orderings is a possible outcome.

Using hierarchical clustering, genes that are placed close to each other in the linear representation of the results, are assumed to share a common function [43]. Thus, the order in which the genes are presented can have an effect on the biological analysis of the results. In addition, when analyzing time series data one is not only interested in the clusters, but also in the relationships between different clusters. To this end, ordering the clusters can help in analyzing the results.

In this section we present an algorithm that re-orders the resulting hierarchical clustering tree so that an optimally ordered tree is presented. This ordering maximizes the sum of the similarity of adjacent leaves in the tree. As we show in Section 6.5 such ordering allows us to recover expression programs that are not visible using the random ordering of the hierarchical clustering procedure.

[1]This chapter is based on references [22] and [14]

6.1 Hierarchical clustering

Hierarchical clustering starts by computing the similarity matrix (S) between all pairs of genes, based on their expression levels. Using the similarity matrix S we combine at each step the two most similar clusters (initially each gene corresponds to a single cluster), and form a new cluster which contains both clusters. There are many different ways in which to choose the different clusters to combine at each step, and these differ based on the distance metric used between clusters. Popular methods include: average linkage, single linkage and complete linkage (see [84] for details). Since our algorithm works for all of these methods, we do not discuss the details of these distance measures here. After combining the two clusters, we compute the similarity of the new cluster to all remaining clusters. For n genes, this step is repeated $n-1$ times until we are left with a single cluster that contains all genes. See [43] for a detailed description of the hierarchical clustering algorithm used in this section. The running time of this algorithm is $O(n^3)$ since for each of the $n-1$ merging steps we perform we need to search the S matrix for the highest similarity. Though there are faster algorithms for hierarchical clustering [44], the above algorithm is the one that is implemented in Cluster [43], and is the one most papers use.

Unlike most other clustering methods, in hierarchical clustering the clusters are not uniquely determined. The only available information is the subtrees in which genes reside. Thus, the user has to determine which of the subtrees are clusters, and which are only a part of a bigger cluster. Any improvement to the basic algorithm (such as the leaf ordering algorithm) helps the user in identifying the different clusters and interpreting the data in the right way. In addition, one might be interested not only in the clusters but also in the relationships between them (for example, when studying time series data). This information is not available in the random order generated by hierarchical clustering.

6.2 Related work

The application of hierarchical clustering to gene expression data was first discussed by Eisen [43]. Hierarchical clustering has become the tool of choice for many biologists, and it has been used to both analyze and present gene expression data [43, 94, 79].

The problem of ordering the leaves of a binary hierarchical clustering tree dates back to 1972 [53]. Due to the large number of applications that construct trees for analyzing

datasets, over the years, many different heuristics have been suggested for solving this problem (cf. [53, 37, 48, 43]). These heuristics either use a 'local' method, where decisions are made based on local observations, or a 'global' method, where an external criteria is used to order the leaves. For the local methods, decisions made in an early stage of the ordering are irreversible, and thus can lead to less than optimal order in the following steps. For the global methods, fitting an external criterion that was generated in a different way is infeasible in most cases, since there are $n!$ possible global orderings, while only 2^{n-1} of them are consistent with the tree.

Recently it has come to our attention that the optimal leaf ordering problem was also addressed by Burkard *et al* [29]. In that paper the authors present an $O(n^3)$ time, $O(n^2)$ space algorithm for optimal leaf ordering of PQ-trees. For binary trees, their algorithm is essentially identical to the basic algorithm we present in section 6.4, except that we propose a number of heuristic improvements. Although these do not affect the asymptotic running time, we experimentally observed that they reduce the running time by 50-90%. In addition, we present several results on synthetic and real data they show that optimal leaf ordering helps in analyzing biological datasets.

6.3 Problem statement

First, we formalize the optimal leaf ordering problem, using the following notations. For a tree T with n leaves, denote by $z_1, \cdots, z_n$ the leaves of T and by $v_1 \cdots v_{n-1}$ the $n-1$ internal nodes of T. A **linear ordering consistent with** T is defined to be an ordering of the leaves of T generated by flipping internal nodes in T (that is, changing the order between the two subtrees rooted at v_i, for any $v_i \in T$). See Figure 6-1 for an example of node flipping.

Since there are $n-1$ internal nodes, there are 2^{n-1} possible linear orderings of the leaves of a binary tree. Our goal is to find an ordering of the tree leaves that maximizes the sum of the similarities of adjacent leaves in the ordering. This could be stated mathematically in the following way. Denote by Φ the space of the 2^{n-1} possible orderings of the tree leaves. For $\phi \in \Phi$ we define $D^{\phi}(T)$ to be:

$$D^{\phi}(T) = \sum_{i=1}^{n-1} S(z_{\phi_i}, z_{\phi_{i+1}})$$

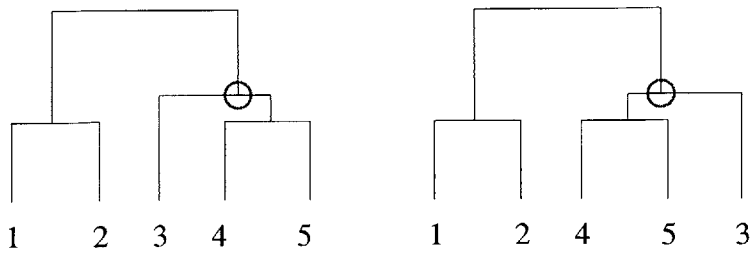


Figure 6-1: When flipping the two subtrees rooted at the red circled node we obtain different orderings while maintaining the same tree structure. Since there are $n-1$ internal nodes there are 2^{n-1} possible orderings of the tree leaves.

where $S(u,w)$ is the similarity between two leaves of the tree. Thus, our goal is to find an ordering ϕ that maximize $D^{\phi}(T)$. For such an ordering ϕ, we say that $D(T) = D^{\phi}(T)$.

6.4 An $O(n^3)$ algorithm for binary trees

Assume that a hierarchical clustering in form of a tree T has been fixed. The basic idea is to create a table M with the following meaning. For any node v of T, and any two genes i and j that are at leaves in the subtree defined by v (denoted $T(v)$), define $M(v,i,j)$ to be the cost of the best linear order of the leaves in $T(v)$ that begins with i and ends with j. $M(v,i,j)$ is defined only if node v is the least common ancestor of leaves i and j; otherwise no such ordering is possible. If v is a leaf, then $M(v,v,v) = 0$. Otherwise, $M(v,i,j)$ can be

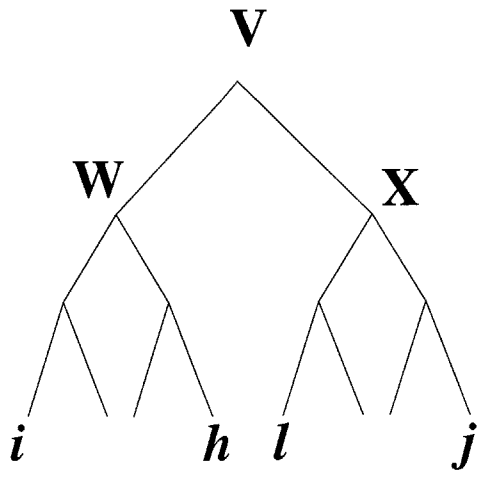


Figure 6-2: Computing $M(v,i,j)$ for a binary tree rooted at V.

computed as follows, where w is the left child and x is the right child of v (see Figure 6-2):

$$M(v,i,j) = \max_{\substack{h \in T(w), \\ l \in T(x)}} M(w,i,h) + S(h,l) + M(x,l,j) \tag{6.1}$$

Let $F(n)$ be the time needed to compute all defined entries in table $(M(v,i,j))$ for a tree with n leaves. We analyze the time to compute Equation 6.1 as follows: Assume that there are r leaves in $T(w)$ and p leaves in $T(x)$, $r+p=n$. We must first compute recursively all values in the table for $T(w)$ and $T(x)$; this takes $F(r)+F(p)$ time.

To compute the maximum, we compute a temporary table $Temp(i,l)$ for all $i \in T(w)$ and $l \in T(x)$ with the formula

$$Temp(i,l) = \max_{h \in T(w)} M(w,i,h) + S(h,l); \tag{6.2}$$

this takes $O(r^2p)$ time since there are rp entries, and we need $O(r)$ time to compute the maximum. Then we can compute $M(v,i,j)$ as

$$M(v,i,j) = \max_{l \in T(x)} Temp(i,l) + M(x,l,j). \tag{6.3}$$

This takes $O(rp^2)$ time, since there are rp entries, and we need $O(p)$ time to compute the maximum.

Thus the total running time obeys the recursion $F(n) = F(r) + F(p) + O(r^2p) + O(rp^2)$ which can be shown easily (by induction) to be $O(n^3)$, since $r^3+p^3+r^2p+rp^2 \leq (r+p)^3 = n^3$.

The required memory is $O(n^2)$, since we only need to store $M(v,i,j)$ once per pair of leaves i and j.

We use some more notations to present a precise bound on the running time of our ordering algorithm. Intuitively, if $f(n) = \Omega(g(n))$ then we can think of $f(n)$ as being 'not less than' $g(n)$ (just as $f(n) = O(g(n))$ means that $f(n)$ is 'not more than' $g(n)$). If $f(n)$ is both upper and lower bounded by $g(n)$ (that is, $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$) then we say that $f(n) = \Theta(g(n))$. Based on the analysis above, for a balanced binary tree with n leaves we need $\Omega(n^3)$ time to compute Equation 6.1; hence the algorithm has running time $\Theta(n^3)$.

We can further improve the running time of the algorithm in practice by using the following techniques:

6.4.1 Early termination of the search.

We can improve the computation time for Equation 6.2 (and similarly Equation 6.3) by pruning the search for maximum whenever no further improvement is possible. To this end, set $s_{\max}(l) = \max_{h \in T(w)} S(h, l)$. Sort the leaves of $T(w)$ by decreasing value of $M(w, i, h)$, and compute the maximum for Equation 6.2 processing leaves in this order. Note that if we find a leaf h for which $M(w, i, h) + s_{\max}(l)$ is bigger than the maximum that we have found so far, then we can terminate the computation of the maximum, since all other leaves cannot increase it.

6.4.2 Top-level improvement.

The second improvement concerns the computation of Equations 6.2 and 6.3 when v is the root of the tree. Let w and x be the left and right children of v. Unlike in the other cases, we do not need to compute $M(v, i, j)$ for all combinations of $i \in T(w)$ and $j \in T(x)$. Rather, we just need to find the maximum of all these values $M_{\max} = \max_{i,j} M(v, i, j)$.

Define $\mathrm{Max}(v, i) = \max_{h \in T(v)} M(v, i, h)$. From Equation 6.1 we have

$$\begin{aligned} & M_{\max} \\ = & \max_{i \in T(w), j \in T(x)} \max_{h \in T(w), l \in T(x)} M(w, i, h) + \\ & S(h, l) + M(x, l, j) \\ = & \max_{h \in T(w), l \in T(x)} \mathrm{Max}(w, h) + S(h, l) + \mathrm{Max}(x, l) \end{aligned}$$

Therefore we can first precompute values $\mathrm{Max}(w, h)$ for all $h \in T(w)$ and $\mathrm{Max}(x, l)$ for all $l \in T(x)$ in $O(n^2)$ time, and then find $M_{\max}$ in $O(n^2)$ time. This is in contrast to the $O(n^3)$ time needed for this computation normally.

While the above two improvements do not improve the theoretical running time (and in fact, the first one increases it because of the sorting step), we found in experiments that on real-life data this variant of the algorithm is on average 50–90% faster.

6.5 Results

We tested our algorithm on several different inputs. First, we used randomly generated datasets to test the effect that optimal ordering has on the sum of similarities of neigh-

boring leaves (the function our algorithm maximizes). Next, we generated datasets to test the effect optimal ordering has on the visual representation of the hierarchical clustering results. We conclude this section by presenting biological results that were obtained using the cell cycle data of [94].

For random data, we chose 60 values (representing 60 time points) at random for each leaf. Next, we computed the resulting similarity matrix, and then hierarchically clustered these data points. Denote by $S(T^r)$ the sum of the similarity between adjacent leaves in the linear ordering of the tree T^r that is generated by the clustering algorithm (for example, from Eisen's Cluster [43]). Denote by $D(T^r)$ the sum of the similarity between adjacent leaves after optimally ordering the leaves of T^r. Set $I = (D(T^r) - S(T^r))/S(T^r)$. I is the increase in similarity of $D(T^r)$ compared with $S(T^r)$. We found that even for a large number of random data points ($n = 1500$), I is on average 20%, indicating that optimal ordering has a big impact on the similarity of neighboring leaves in the linear ordering (see figure 6-3).

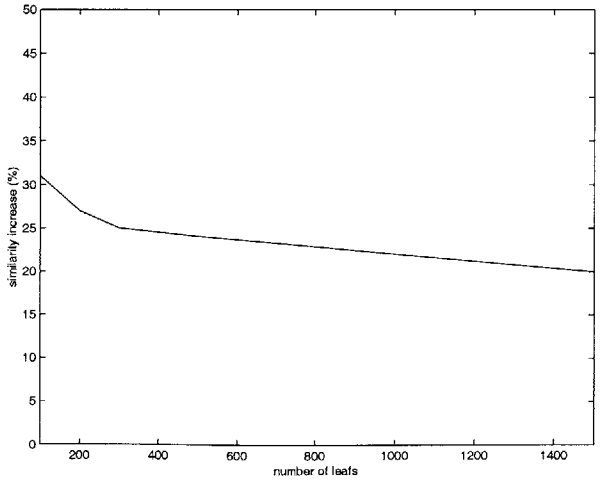


Figure 6-3: Average increase in the sum of similarities of neighboring leaves for a randomly generated dataset, as a function of the number of leaves. Note that even for a large number of leaves the increase is large.

6.5.1 Generated data

In order to test the effect of leaf ordering on the presentation of the data, we generated several input data sets. Each data set had some structure which we permuted, and then we ran the hierarchical clustering algorithm to cluster the data set. In this section we compare the results of the Eisen clustering algorithm [43] with our ordering algorithm. The Eisen results where generated using Cluster (the software package that implements Eisen's algorithm). All the figures in this section where generated using TreeView.

In Figure 6-4 we present a dataset that were manually generated. As can be seen, although there is some structure in the Cluster output, it is obvious that the ordering algorithm captures the true underline structure.

Cluster result Input (permuted) Ordering result

Figure 6-4: Comparison between the ordered and non ordered hierarchical clustering on a generated dataset. Green corresponds to decrease in value (-1) and red to increase (1).

Next, we performed a larger scale test in the following way. We generated 100 data points for each leaf. For each leaf we set 40 of these points to -1 with high probability (living a small probability of generating a different value for these 40 data points). The rest of the points where generated at random. The results of the initial ordering (which is similar to the random ordering of Cluster), and of the similarity maximizing ordering are shown in Figure 6-5. As can be seen, the clusters where identified correctly by the hierarchical clustering algorithm. However, the ordering of the clusters is much better when using our algorithm. This can be important for time series data in which one is not only interested in the actual clusters, but also in the relationship between the clusters (such as which genes are repressed early in the time series and which are repressed at the end, etc.). In addition, as we show in the two cluster enlargements, the ordering algorithm orders the clusters themselves, and not only the relationships between them.

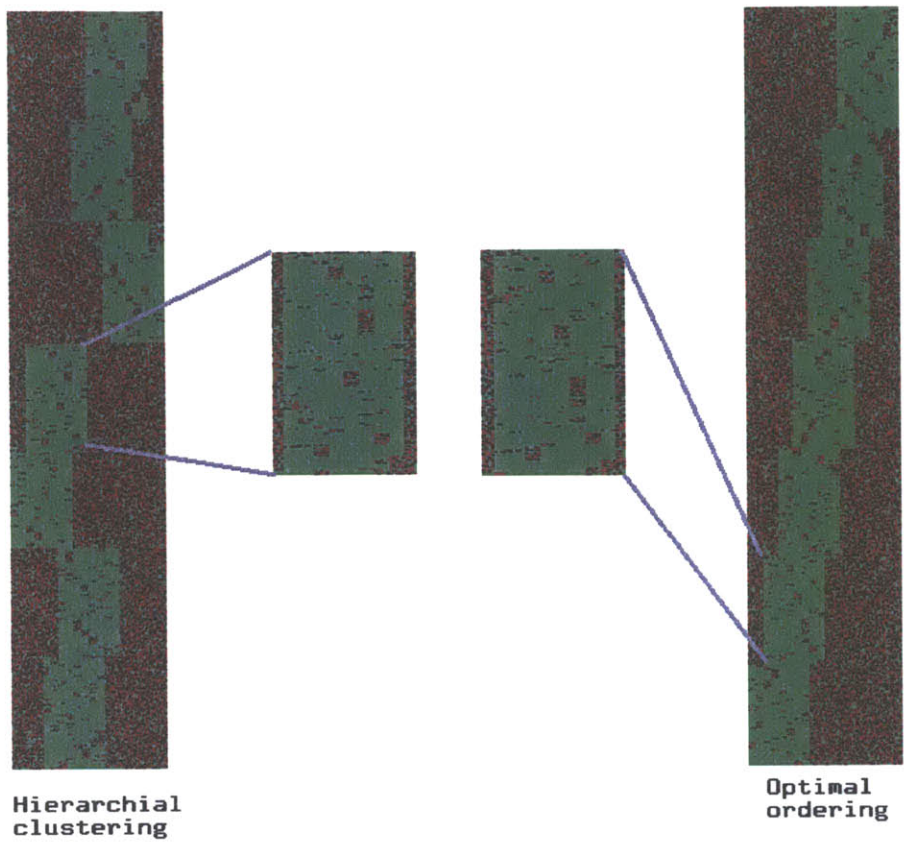


Figure 6-5: Comparison between the ordered and non ordered hierarchical clustering. The small images are enlargements of the same cluster in both results. As can be seen, the ordering algorithm orders the clusters so that the relationship between them is apparent. In addition, it also orders the genes inside each cluster as can be seen in the small images.

6.5.2 Biological results

We tested our algorithm using the cell cycle data presented in [94]. In that paper, the authors identify 800 genes which are cell cycle regulated in *Saccharomyces cerevisiae*. The authors assigned these genes to five groups termed $G1$, S, $S/G2$, $G2/M$, and $M/G1$. These groups approximate the commonly used cell cycle groups in the literature (see [94] and http://cellcycle-www.stanford.edu for the complete data set and gene assignment to the different groups). The assignment to different groups was performed by what the authors

call a 'phasing' method. This method compares the 'peak expression' for each unknown gene with the expression of genes that where known to belong to each of these group. Genes where assigned to the group to which their peak expression was the most similar.

After performing the phasing algorithm the authors clustered these 800 genes using hierarchical clustering. Most of the genes that belong to the same group where clustered together. However, some clusters where a mix of two or more groups, and the relationship between the clusters was not apparent

We used our ordering algorithm to generate an optimal ordering of these 800 genes in the 24 cdc15 experiments. As can be seen in Figure 6-6, our algorithm was able to correctly recover the cell cycle order (starting with $G2/M$ and going through $M/G1$, $G1$, S and $S/G2$). In addition, our algorithm was able to reorder some of the clusters so that the different gene groups (clusters) are correctly separated (as can be seen in the $G1$, S and $S/G2$ groups which are mixed in the hierarchical clustering results but are correctly separated in the optimal ordering results). Thus, while still using unsupervised learning, our algorithm was able to correctly identify the cell cycle groups and the order of these groups, achieving a high correlation with the phasing method (which is a supervised algorithm) that was used by the authors of [94].

6.6 Summary

We presented an $O(n^3)$ algorithm for computing the optimal ordering of hierarchical clustering trees. The algorithm we presented is general, and works for any binary tree when a metric is defined on the leaves. We have also presented a number of improvements which drastically reduce the running time of this algorithm in practice. Using optimal ordering one can arrive at results that are better than the original hierarchical clustering results. We presented several examples in which the results of the ordering algorithm are superior to the original clustering results. The ordering allows the user to determine not only the clusters, but also the relationship between different clusters. This could be very important in time series data analysis.

In addition to an optimal ordering algorithm, we have recently presented an algorithm for optimal level ordering, which solves a variant of the optimal ordering problem in $O(n)$ time and space (see Section 7.5). While $O(n^3)$ is a reasonable running time for most

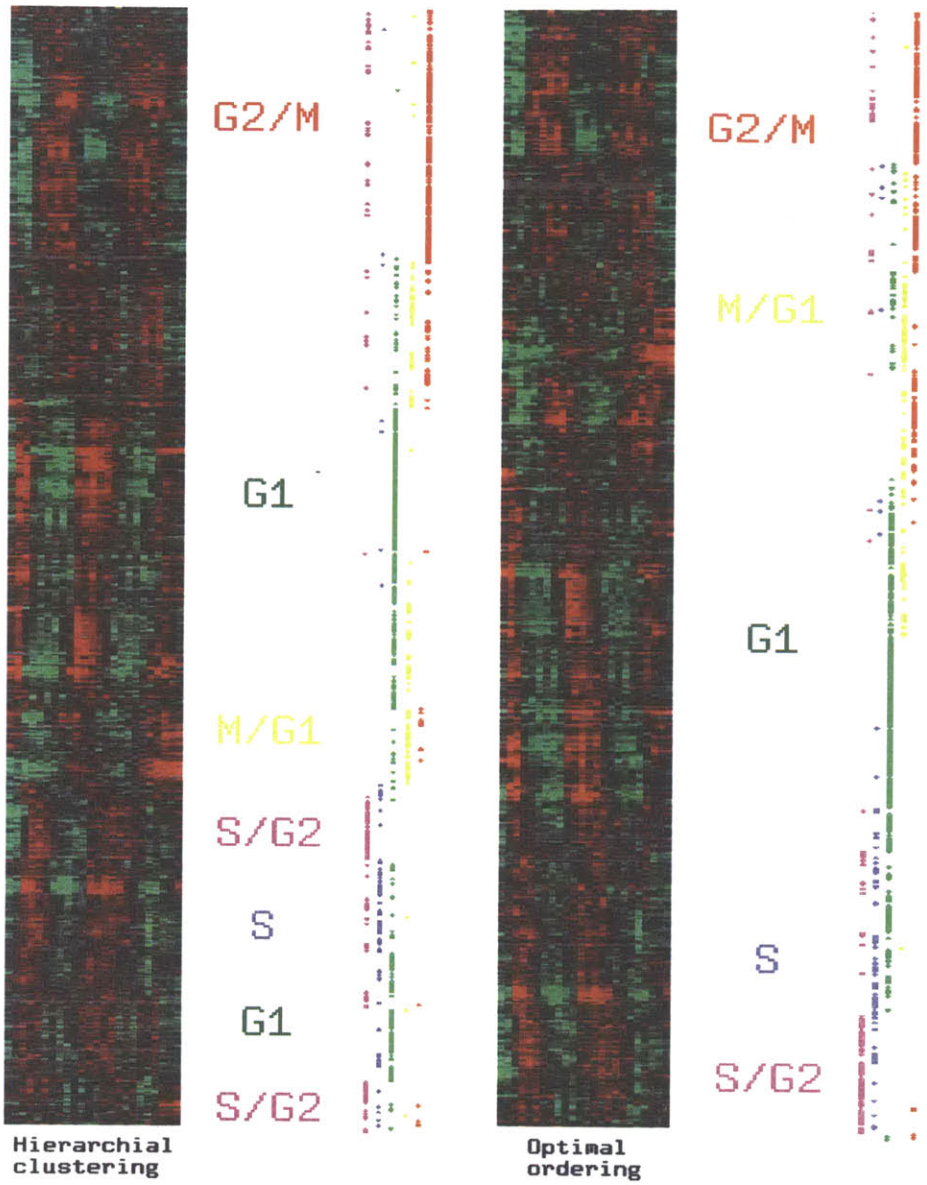


Figure 6-6: Comparison between hierarchical clustering (left) and optimal ordering (right) using the cell cycle data of [94]. For each of the genes in these figures we plotted a dot on its right which represents the group to which it belongs according to [94] (red for $G2/M$ yellow for $M/G1$ etc.). We have indicated areas in the clustering trees that contain a significant number of genes from a specific phase by placing the name of that phase to the left of the plotted dots. As can be seen , the cell cycle phasing is much more apparent in the optimal ordering result, which correctly recovers the order of the groups in the cell cycle. In addition, using optimal ordering one can better reconstruct the groups themselves as can be seen in the case of the $G1$, S and $S/G2$ groups.

expression datasets, for very large ones (over 10000) this time might be prohibitive. The level ordering algorithm allows users to achieve a reasonable ordering of such datasets in a very fast time. In addition, we have recently shown [13] how to use the level ordering algorithm for a completely different task: Improving lossless image compression. Since many algorithms construct trees as part of their solution to various problems, we anticipate several more uses of our optimal ordering and level ordering algorithms for many different computer science problems, including text compression, image compression and database visualization (see also Section 7.5).

While our ordering algorithm solves one of the major problems of hierarchical clustering, ordering the leaves, there are a number of problems it does not address. Hierarchical clustering is very sensitive to noise and it usually lacks of a method to actually identify distinct clusters. In the next section we present an algorithm that solves these problems by constructing a k-ary tree instead of a binary tree, and uses an extension of the optimal ordering algorithm discussed in this section to restore binary relationships that are lost when using more than two children per node.

Chapter 7

K-ary clustering and ordering[1]

As mentioned in the previous section, hierarchical clustering is one of the most popular methods for clustering gene expression data. However, hierarchical clustering is very sensitive to noise, since in a typical implementation if two clusters (or genes) are combined they cannot be separated even if farther evidence suggests otherwise [84]. In addition, hierarchical clustering does not specify the components of the tree that comprise a distinct cluster, making it hard to distinguish between internal nodes that are roots of a cluster and nodes which only hold subsets of a cluster. Finally, as discussed in the previous section, the ordering of the leaves, which plays an important role in analyzing and visualizing hierarchical clustering results, is not defined by the algorithm.

In this section we propose a new hierarchical clustering algorithm which reduces susceptibility to noise, permits up to k siblings to be directly related, and provides a single optimal order for the resulting tree, without sacrificing the tree structure, or the pairwise relationships between neighboring genes and clusters in the result. Our solution replaces the binary tree of the hierarchical clustering algorithm with a k-ary tree. A k-ary tree is a tree in which each internal node has at most k children. When grouping k clusters (or genes) together, we require that all the clusters that are grouped together will be similar to one another. It has been shown (e.g. in CLICK [84]) that relying on similarities among large groups of genes helps reduce the noise effects that are inherent in expression data. Our algorithm utilizes this idea for the hierarchical case. In our case, we are interested in groups of genes that are similar, where the notion of similarity depends on the scale we are

[1]This chapter is based on reference [14]

looking at.

The number of children of each internal node is not fixed to k. Rather, k is an upper bound on this number, and if the data suggests otherwise this number can be reduced. An advantage of such a method is that it allows us to highlight some of the actual clusters since nodes with less than k children represent a set of genes that are similar, yet significantly different from the rest of the genes. Such a distinction is not available when using a binary tree.

Finally, our algorithm re-orders the resulting tree so that an optimally ordered tree is presented. This ordering maximizes the sum of the similarity of adjacent leaves in the tree, allowing us to obtain the best pairwise relationships between genes and clusters, even when $k > 2$.

The running time of our clustering algorithm (for small values of k) is $O(n^3)$, which is similar to the running time of currently used hierarchical clustering algorithms. As for our ordering algorithm, for $k = 2$ the running time is $O(n^3)$ as discussed in the previous chapter. For $k > 2$ our ordering algorithm runs in $O(4^k n^3)$ time and $O(kn^2)$ space which is feasible even for a large n (when k is small).

The rest of this chapter is organized as follows. In Section 7.2 we present an algorithm for constructing k-ary trees from gene expression data. In Section 7.3 we present the optimal leaf ordering algorithm for k-ary trees. In Section 7.4 we present our experimental results, and Section 7.6 summarizes this chapter and suggests directions for future work.

7.1 Related work

The application of hierarchical clustering to gene expression data was first discussed by Eisen [43]. Hierarchical clustering has become the tool of choice for many biologists, and it has been used to both analyze and present gene expression data [43, 94, 79]. A number of different clustering algorithms, which are more global in nature, where suggested and applied to gene expression data. Examples of such algorithms are K-means, Self organizing maps [96] and the graph based algorithms Click [83] and CAST [26]. These algorithms generate clusters which are all assumed to be on the same level, thus they lack the ability to represent the relationships between genes and sub clusters on different scales as hierarchical clustering does. In addition, they are usually less suitable for large scale visualization tasks,

since they do not generate a global ordering of the input data. In this chapter we try to combine the robustness of these clustering algorithms with the presentation and flexible groupings capabilities of hierarchical clustering.

Recently, Segal and Koller [88] suggested a probabilistic hierarchical clustering algorithm, to address the robustness problem. Their algorithm assumes a specific model for gene expression data. In contrast, our algorithm does not assume any model for its input data, and works with any similarity/distance measure. In addition, in this chapter we present a method that allows not only to generate the clusters but also to view the relationships between different clusters, by optimally ordering the resulting tree.

The optimal leaf ordering problem for k-ary trees was previously addressed by Burkard *et al* [29]. In that paper the authors present an $O(2^k n^3)$ time, $O(2^k n^2)$ space algorithm for optimal leaf ordering of PQ-trees. Unlike their algorithm, the algorithm we present in this section uses a different search strategy over the children of a node. Burkard *et al.* suggest a dynamic programming approach which is more computationally efficient ($O(2^k n^3)$ vs. $O(4^k n^3)$), while we propose a divide and conquer approach which is more space-efficient ($O(kn^2)$ vs. $O(2^k n^2)$). The number of genes (n) in an expression data set is typically very large, making the memory requirements very important. In our experience, the lower space requirement, despite the price in running time, enables using larger ks.

7.2 Constructing K-ary Trees

In this section we present an algorithm for constructing k-ary trees. We first formalize the k-ary tree problem, and show that finding an optimal solution is hard (under standard complexity assumptions). We then present a heuristic algorithm for constructing k-ary trees for a fixed k, and extend this algorithm to allow for nodes with at most k children.

7.2.1 Problem statement

As is the case in hierarchical clustering, we assume that we are given a gene similarity matrix S, which is initially of dimensions n by n. Unlike binary tree clustering, we are interested in joining together groups of size k, where $k > 2$. In this paper we focus on the average linkage method, for which the problem can be formalized as follows. Given n

clusters denote by C the set of all subsets of n of size k. Our goal is to find a subset $b \in C$ s.t. $V(b) = max\{V(b')|b' \in C\}$ where V is defined in the following way:

$$V(b) = \sum_{i,j \in b, i<j} S(i,j) \tag{7.1}$$

That is, $V(b)$ is the sum of the pairwise similarities in b. After finding b, we merge all the clusters in b to one cluster. The revised similarity matrix is computed in the following way. Denote by i a cluster which is not a part of b, and let the cluster formed by the merging the clusters of b be denoted by j. For a cluster m, let $|m|$ denote the number of genes in m, then:

$$S(i,j) = \frac{\sum_{m \in b} |m| S(m,i)}{\sum_{m \in b} |m|}$$

which is similar to the way the similarity matrix is updated in the binary case. This process is repeated $(n-1)/(k-1)$ times until we arrive at a single root cluster, and the tree is obtained.

Finding b in each step is the most expensive part of the above problem, as we show in the next lemma. In this lemma we use the notion of $W[1]$ hardness. Under reasonable assumptions, a $W[1]$ hard problem is assumed to be *fixed parameter* intractable, i.e. the dependence on k cannot be separated from the dependence on n (see [40] for more details).

Lemma 7.2.1 *Denote by $MaxSim(k)$ the problem of finding the first b set for a given k. Then $MaxSim$ is NP-hard for arbitrary k, and $W[1]$ hard in terms of k.*

Proof: The NP-completeness is proved by a reduction from MAX-CLIQUE. Given a graph G, we construct the similarity matrix S_G in the following way. For any two nodes $i, j \in G$ we set $S_G(i,j) = 1$ if there is an edge between i and j in G, and 0 otherwise. Denote by b_k the b set generated by $MaxSim(k)$ using S_G. For a given k we can test if $V(b_k) = k(k-1)/2$, and if so there is a clique of size k in G. repeating this for all $k = 2 \ldots n$ possible values of k solves the MAX-CLIQUE problem, so $MaxSim(k)$ is NP-complete.

As for the $W[1]$ completeness, the same reduction holds since Clique is $W[1]$ complete for a fixed k (see [40] pages 248-249). ■

7.2.2 A heuristic algorithm for constructing k-ary trees

As shown in the previous section, any optimal solution for the k-ary tree construction problem might be prohibitive even for small values of k, since n is very large. In this section we present a new heuristic algorithm, which has a running time of $O(n^3)$ for any k, and reduces to the standard average linkage clustering algorithm when $k = 2$. The algorithm is presented in Figure 7-1 and works in the following way. Starting with a set of n clusters (initially each gene is assigned to a different cluster), we generate L_i which is a linked list of clusters for each cluster i. The clusters on L_i are ordered by their similarity to i in descending order. For each cluster i we compute $V(b_i)$ where b_i consists of i and the clusters that appear in the first $k-1$ places on L_i, and V is the function described in equation 7.1. Next, we find $b = argmax_i\{V(b_i)\}$, the set of clusters that have the highest similarity among all the b_i sets that are implied by the L_i list. We merge all the clusters in b to a single cluster denoted by p, and recompute the similarity matrix S. After finding b and recomputing S we go over the L_i lists. For each such list L_i, we delete all the clusters that belong to b from L_i, insert p and recompute b_i. In addition, we generate a new list L_p for the new cluster p, and compute b_p.

```
KTree(n, S) {
    C = {1 ... n}
    for all j ∈ C // preprocessing step
        L_j = ordered linked list of genes based on similarity to j
        b_j = j ⋃ first k − 1 genes of L_j
    for i = 1 : (n − 1)/(k − 1) { // main loop
        b = argmax_{j∈C}{V(b_j)}
        C = C \ b
        Let p = min{m ∈ b}
        for all clusters j ∈ C
            S(p, j) = (Σ_{m∈b} |m| S(m, j)) / (Σ_{m∈b} |m|)
            remove all clusters in b from L_j
            insert p into L_j
            b_j = j ⋃ first k − 1 cluster of L_j
        C = C ⋃ p
        generate L_p from all the clusters in C and find b_p
    }
    return C // C is a singleton which is the root of the tree
}
```

Figure 7-1: Constructing k-trees from expression data

Note that using this algorithm, it could be that even though j and i are the most similar

clusters, j and i will not end up in the same k group. If there is a cluster t s.t. b_t includes i but does not include j, and if $V(b_t) > V(b_j)$, it could be that j and i will not be in the same k cluster. This allows us to use this algorithm to overcome noise and missing values since, even when using this heuristic we still need a strong evidence from other clusters in order to combine two clusters together.

The running time of this algorithm is $O(n^3)$. Generating the L_js lists can be done in $O(n^2 \log n)$, and finding b_j for all genes j can be done in kn time. Thus the preprocessing step takes $O(n^2 \log n)$.
For each iteration of the main loop, it takes $O(n)$ to find b, and $O(nk)$ to recompute S. It takes $O(kn^2 + n^2 + kn)$ to delete all the members of b from all the L_js, insert p into all the L_js and recompute b_j. We need another $O(n \log n + k)$ time to generate L_p and compute b_p. Thus, the total running time of each iteration is $O(kn^2)$. Since the main loop is iterated $(n-1)/(k-1)$ time, the total running time of the main loop is $O(k(n-1)n^2/(k-1)) = O(n^3)$ which is also the running time of the algorithm.

The running time of the above algorithm can be improved to $O(n^2 \log n)$ by using a more sophisticated data structure instead of the linked list. For example, using Heaps, the preprocessing step has the same complexity as we currently have, and it can be shown that the (amortized) cost of every step in the main loop iteration becomes $O(n \log n)$. However, since our ordering algorithm operates in $O(n^3)$, this will not reduce the asymptotic running time of our algorithm, and since the analysis is somewhat more complicated in the Heap case we left the details out.

7.2.3 Reducing the number of children

Using a fixed k can lead to clusters which do not have a single node associated with them. Consider for example a dataset in which we are left with four internal nodes after some main loop iterations. Assume $k = 4$ and that the input data is composed of two real clusters, A and B such that three of the subtrees belong to cluster A, while the fourth subtree belongs to cluster B (see Figure 7-2). If k was fixed, we would have grouped all the subtrees together, which results in a cluster (A) that is not associated with any internal node. However, if we allow a smaller number of children than k we could have first grouped the three subtrees of A and later combine them with B at the root. This can also highlight the fact that A and B are two different clusters, since nodes with less than k children represent a set of genes

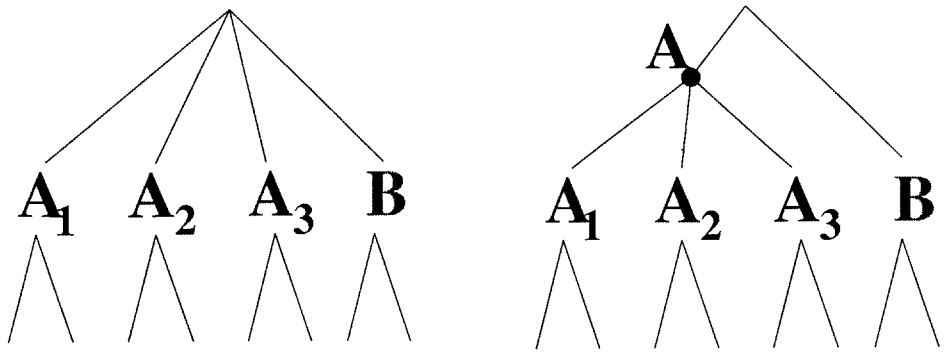


Figure 7-2: Fixed vs. non fixed k. In the left hand tree k is fixed at 4. This results in a cluster (A) which does not have any internal node associated with it. On the right hand side k is at most 4. Thus, the three subtrees that form cluster A can be grouped together and then combined with cluster B at the root. This results in an internal node that is associated with A.

that are similar, yet significantly different than the rest of the genes.

We now present a permutation based test for deciding how many clusters to combine in each iteration of the main loop. There are two possible approaches one can take in order to perform this task. The first is to join as many clusters as possible (up to k) unless the data clearly suggests otherwise. The second is the opposite, i.e. to combine as few clusters as possible unless the data clearly suggests otherwise. Since we believe that in most cases more than 2 genes are co-expressed, in this paper we use the first approach, and combine all k clusters unless the data clearly suggests otherwise.

Let $k = 3$ and assume $b = b_c$, i.e. $b_c = argmax_i\{V(b_i)\}$ where i goes over all the clusters we have in this step. Let d be the first cluster on L_c and let e be the second cluster. Since d is the first on L_c, it is the most similar to c. We now wish to decide whether to combine the first two clusters (c and d) or combine all three clusters. Let $max_e = max\{S(c,e), S(d,e)\}$, that is max_e is the maximal similarity between e and one of the two clusters we will combine in any case. In order to test the relationship between max_e and $S(c,d)$, we perform the following test. In our case, each cluster c is associated with a profile (the average expression values of the genes in c). Assume our dataset contains m experiments, and let p_c, p_d and p_e be the three clusters profiles. Let p be the 3 by m matrix, where every row of p is a profile of one of the clusters. We permute each column of p uniformly and independently at random, and for the permuted p we compute the best (s_1) and second best (s_2) similarities among its rows. We repeat this procedure r times, and in each case test if s_2 is bigger than

max_e or smaller. If $s_2 > max_e$ at least αr times (where α is a user defined value between 0 and 1) we combine c and d without e, otherwise we combine all three clusters. Note that if c and d are significantly different from e then it is unlikely that any permutation will yield an s_2 that is lower than max_e, and thus the above test will cause us to separate c and d from e. If c and d are identical to e, then all permutations will yield an s_2 that is equal to max_e, causing us to merge all three clusters. As for the values of α, if we set α to be close to 1 then unless e is very different from c and d we will combine all three clusters. Thus, the closer α is to 1, the more likely our algorithm is to combine all three clusters.

For $k > 3$ we repeat the above test for each $k' = 3 \ldots k$. That is, we first test if we should separate the first two clusters from the third cluster, as described above. If the answer is yes, we combine the first two clusters and move to the next iteration. If the answer is no we apply the same procedure, to test weather we should separate the first three clusters from the fourth and so on. The complexity of these steps is rk^2 for each k' (since we need to compute the pairwise similarities in each permutations), and at most rk^3 for the entire iteration. For a fixed r, and $k << n$ this permutation test does not increase the asymptotic complexity of our algorithm. Note that if we combine $m < k$ clusters, the number of main loop iteration increases. However, since in this case each the iteration takes $O(n^2 m)$ the total running time remains $O(n^3)$.

7.3 Optimal leaf ordering for k-ary trees

In this section we discuss how we preserve the pairwise similarity property of the binary tree clustering in our k-ary tree algorithm. This is done by performing optimal leaf ordering on the resulting k-ary tree. We present an algorithm that extends our binary optimal leaf ordering algorithm from Section 6 to general k-ary trees.

The Optimal order of the leaves of a k-ary tree is defined in the same way as the optimal ordering for binary trees. That is, we are looking for an ordering that maximizes the sum of the similarities of neighboring leaves in the ordering. However, unlike binary trees, for which we have 2^{n-1} possible leaf orderings, for a k-ary tree there could be as many as $k!^{n/(k-1)}$ possible orderings. Even for a relatively small value of k, this is a much higher number if n is big.

For a k-ary tree, denote by $v_1 \ldots v_k$ the k subtrees of v. Assume $i \in v_1$ and $j \in v_k$, then

any ordering of $v_2 \ldots v_{k-2}$ is a possibility we should examine. For a specified ordering of the subtrees of v, $M(v,i,j)$ can be computed in the same way we computed M for binary trees by inserting $k-2$ internal nodes that agree with this order (see figure 7-3). Thus, we first

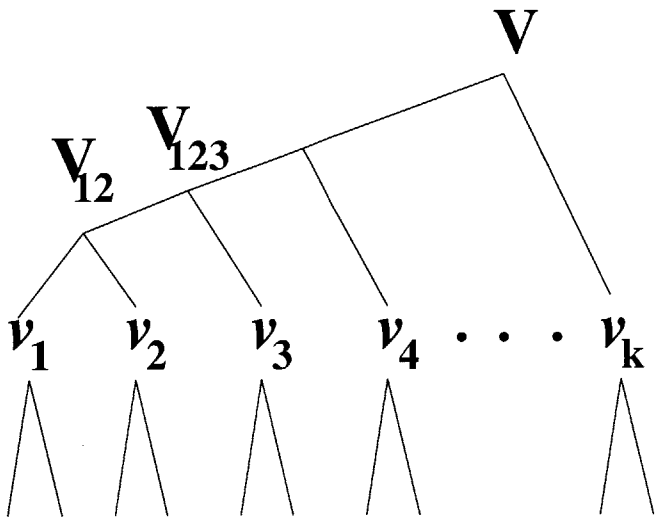


Figure 7-3: Computing $M(v,i,j)$ for the subtrees order $1 \ldots k$. For each possible ordering of $1 \ldots k$ we can compute this quantity by adding internal nodes and using the binary tree algorithm.

compute $M(v_{1,2},h,l)$ for all h and l leaves of v_1 and v_2. Next we compute $M(v_{1,2,3},*,*)$ and so on until we compute $M(v,i,j)$ for this order. This results in the optimal ordering of the leaves when the subtrees order is $v_1 \ldots v_k$. Since there are $k!$ possible ordering of the subtrees, going over all $k!$ orderings of the subtrees in the manner described above gives rise to a simple algorithm for finding the optimal leaf ordering of a k-ary tree. Denote by $p_1 \ldots p_k$ the number of leaves in $v_1 \ldots v_k$ respectfully. Denote by Θ the set of $k!$ possible orderings of $1 \ldots k$. The running time of this algorithm is $O(k!n^3)$ as can be seen using induction from the following recursion:

$$
\begin{aligned}
&F(n) = \sum_i F(p_i) + \\
&\sum_{\theta\in\Theta}\sum_{i=1}^{k-1}\left((\sum_{j=1}^{i} p_{\theta(j)})^2 p_{\theta(i+1)} + (\sum_{j=1}^{i} p_{\theta(j)}) p_{\theta(i+1)}^2\right) \\
&\leq k!\sum_i p_i^3 + \sum_{\theta\in\Theta}((\sum_i p_i)^3 - \sum_i p_i^3) \\
&= k!(p_1 + p_2 + \ldots p_k)^3 = k!n^3
\end{aligned}
$$

Where the inequality uses the induction hypothesis. As for space complexity, for two leaves, $i \in v_1$ and $j \in v_k$ we need to store $M(v, i, j)$. In addition, it might be that for two other leaves $m \in v_2$ and $l \in v_{k-1}$ i and j are two boundary leaves in the internal ordering of the subtrees of v, and thus we need to store the distance between them for this case as well. The total number of sub-distances we need to store for each pair is at most $k-2$, since there are only $k-2$ subtrees between the two leftmost and rightmost nodes, and thus by deleting all sub-paths which we do not use we only need $O(kn^2)$ memory for this algorithm.

7.3.1 Improving the running time using a divide and conquer algorithm

Though $O(k!n^3)$ is a feasible running time for small ks, we can improve upon this algorithm using the following observation. If we partition the subtrees of v into two groups, v' and v'', then we can compute $M(v)$ for this partition (i.e. when the subtrees of v' are on the right side and the subtrees of v'' on the left) by first computing the optimal ordering on v' and v'' separately, and then combining the result in the same way discussed in Section 6.4. This gives rise to the following divide and conquer algorithm. Assume $k = 2^m$, recursively compute the optimal ordering for all the $\binom{k}{k/2}$ possible partitions of the subtrees of v to two groups of equal size, and merge them to find the optimal ordering of v.

We now prove that the running time of the divide and conquer algorithm for ordering k-ary trees presented above is $O(4^k n^3)$. In order to compute the running time of this algorithm we introduce the following notations: We denote by $\Gamma(v)$ the set of all the possible partitions of the subtrees of v to two subsets of equal size. For $\gamma \in \Gamma(v)$ let $v_{\gamma(1)}$ and $v_{\gamma(2)}$ be the two subsets and let $p_{\gamma(1)}$ and $p_{\gamma(2)}$ be the number of leaves in each of these subsets. The running time of this algorithm can be computed by solving the following recursion:

$$F(n) = F(p_1 + p_2 \ldots + p_k) = \sum_{i=1}^{k} F(p_i) + D(v)$$

Where

$$\begin{aligned} D(v) &= \sum_{\gamma \in \Gamma(v)} p_{\gamma(1)}^2 p_{\gamma(2)} + p_{\gamma(1)} p_{\gamma(2)}^2 + \\ &\quad + D(v_{\gamma(1)}) + D(v_{\gamma(2)}) \end{aligned}$$

and $D(i) = 0$ if i is a singleton containing only one subtree. The following lemma discusses

the maximal running time of the divide and conquer approach.

Lemma 7.3.1 *Assume $k = 2^m$ then*

$$D(v) \leq \frac{(2^m)!}{\prod_{i=0}^{m-1}(2^i)!}(\sum_{j=1}^{k} p_j)^3 - \sum_{j=1}^{k} p_j^3$$

Proof: By induction on m.

For $m = 1$ we have $k = 2$ and we have already shown in Section 6.4 how to construct an algorithm that achieves this running time for binary trees. So $D(v) = p_1^2 p_2 + p_1 p_2^2 < 2(p_1 + p_2)^3 - p_1^3 - p_2^3$.

Assume correctness for $m - 1$. Let $k = 2^m$. Then for every $\gamma \in \Gamma(v)$ we have

$$\begin{aligned}
&p_{\gamma(1)}^2 p_{\gamma(2)} + p_{\gamma(1)} p_{\gamma(2)}^2 + D(v_{\gamma(1)}) + D(v_{\gamma(2)}) \leq \\
&p_{\gamma(1)}^2 p_{\gamma(2)} + p_{\gamma(1)} p_{\gamma(2)}^2 + \frac{(2^{m-1})!}{\prod_{i=0}^{m-2}(2^i)!}(\sum_{j\in\gamma(1)} p_j)^3 \\
&- \sum_{j\in\gamma(1)} p_j^3 + + \frac{(2^{m-1})!}{\prod_{i=0}^{m-2}(2^i)!}(\sum_{j\in\gamma(2)} p_j)^3 - \sum_{j\in\gamma(2)} p_j^3 \\
&\leq \frac{(2^{m-1})!}{\prod_{i=0}^{m-2}(2^i!)}(\sum_{j=1}^{k} p_j)^3 - \sum_{j=1}^{k} p_j^3
\end{aligned}$$

The first inequality comes from the induction hypothesis, and the second inequality arises from the fact that $\gamma(1)$ does not intersect $\gamma(2)$ and $\gamma(1) \bigcup \gamma(2) = \{1 \ldots k\}$. Since $|\Gamma(v)| = \binom{k}{k/2} = \frac{(2^m)!}{(2^{m-1})!(2^{m-1})!}$, summing up on all $\gamma \in \Gamma$ proves the lemma. ∎

It is now easy to see (by using a simple induction proof, as we did for the binary case) that the total running time of this algorithm is: $O(\frac{k!}{\prod_{i=0}^{m-1}(2^i)!}n^3)$ which is faster than the direct extension discussed above. If $2^m < k < 2^{m+1}$ then the same algorithm can be used by dividing the subtrees of v to two groups of size 2^m and $k - 2^m$. A similar analysis shows that in this case the running time is $O(\frac{k!}{\prod_{i=0}^{\lfloor \log k \rfloor - 1}(2^i)!}n^3)$, which (using the Sterling approximation) is $O(4^{k+o(k)}n^3)$.

Thus, we can optimally order the leaves of a k-ary tree in $O(4^k n^3)$ time and $O(kn^2)$ space, which are both feasible for small ks.

7.4 Experimental results

First, we looked at how our heuristic k-ary clustering algorithm (described in Section 7.2.2) compares with the naive k-ary clustering which finds and merges the k most similar clusters in each iteration. As discussed in Section 7.2.1, the naive algorithm works in time $O(n^{k+1})$, and thus even for $k = 3$ it is more time consuming than our heuristic approach. We have compared both algorithms on a 1.4 GHz Pentium machine using an input dataset of 1000 genes and setting $k = 3$. While our k-ary clustering algorithm generated the 3-ary tree in in 35 seconds, the naive algorithm required almost an hour (57 minutes) for this task. Since a dataset with 1000 genes is relatively small, it is clear that the naive algorithm does not scale well, and cannot be of practical use, especially for values of k that are greater than 3. In addition, the results of the naive algorithm where not significantly better when compared with the results generated by our heuristic algorithm. The average similarity of the descendants of an internal node in the naive algorithm tree was only 0.8% higher than the average similarity in the tree generated by our algorithm (0.6371 vs. 0.6366).

Next we compared the binary and k-ary clustering using synthetic and real datasets, and show that in all cases we looked at we only gain from using the k-ary clustering algorithm.

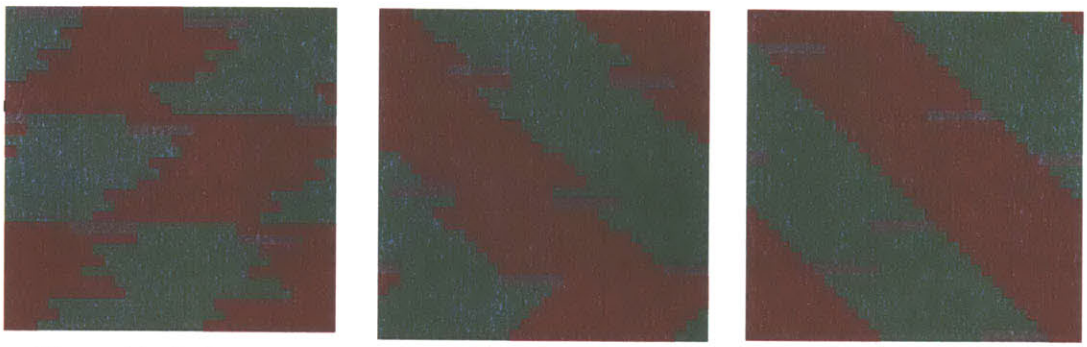
Choosing the right value for k is a non trivial task. The major purpose of the k-ary algorithm is to reduce the influence of noise and missing values by relying on more than that most similar cluster. Thus, the value of k depends on the amount of noise and missing values in the input dataset. For the datasets we have experienced with, we have found that the results do not change much when using values of k that are higher than 4 (though there is a difference between 2 and 4 as we discuss below). Due to the fact that the running time increases as a function of k, we concentrate on $k = 4$.

For computing the hierarchical clustering results discussed below we used the correlation coefficients as the similarity matrix (S).

7.4.1 Generated data

To test the effect of k-ary clustering and ordering on the presentation of the data, we generated a structured input data set. This set represents 30 temporally related genes, each one with 30 time points. In order to reduce the effect of pairwise relationships, we chose 6 of these genes, and manually removed for each of them 6 time points, making these

time points missing values. Next we permuted the genes, and clustered the resulting dataset with the three methods discussed above. The results are presented in Figure 7-4. As can be seen, using optimal leaf ordering (o.l.o.) with binary hierarchical clustering improved the presentation of the dataset, however o.l.o. was unable to overcome missing values, and combined pairs of genes which were similar due to the missing values, but were not otherwise similar to a larger set of genes. Using the more robust k-ary tree algorithm, we where able to overcome the missing values problem. This resulted in the correct structure as can be seen in Figure 6-4.



Hierarchical clustering | Binary clustering with o.l.o. | 4-tree clustering with o.l.o.

Figure 7-4: Comparison between the three different clustering methods on a manually generated dataset. Green corresponds to decrease in value (-1) and red to increase (1). Gray represents missing values.

7.4.2 Visualizing biological datasets

For the biological results we used two datasets. The first was a dataset from [79] which looked at the chicken immune system during normal embryonic B-cell development and in response to the overexpression of the *myc* gene. This dataset consists of 13 samples of transformed bursal follicle (TF) and metastatic tumors (MT). These samples where organized in decreasing order based on the *myc* overexpression in each of them. In that paper the authors focused on approximately 800 genes showing 3 fold change in 6 of 13 samples. These genes were clustered using Eisen's Cluster ([43]), and based on manual inspection, 5 different clusters where identified. In Figure 7-5 we present the results of the three clustering algorithms on this dataset. The left hand figure is taken from [79], and contains the labels for the 5 clusters identified. [79] discuss the five different classes, and

separate them into two groups. The first is the group of genes in clusters A,B,E and D which (in this order) contain genes that are decreasingly sensitive to *myc* overexpression. The second is the cluster C which contains genes that are not correlated with *myc* overexpression. As can be seen, when using the k-ary clustering algorithm (right hand side of Figure 7-5) these clusters are displayed in their correct order. Furthermore, each cluster is organized (from bottom to top) based on the required level of *myc* overexpression. This allows for an easier inspection and analysis of the data. As can be seen, using binary tree clustering with o.l.o. does not yield the same result, and the order that is generated by this algorithm divides E into two parts that are separated by D.

The resulting 4-ary tree for the *myc* dataset is presented in the left hand side of Figure 7-5. Each node is represented by a vertical line. We highlighted (in red) some of the nodes that contain less than 4 children. Note that some of these nodes correspond to clusters that where identified in the original paper (for example, the top red node corresponds to cluster C). Had we used a fixed $k = 4$, these clusters might not have had a single node associated with them.

7.4.3 Clustering biological datasets

The second biological dataset we looked at is a collection of 79 expression experiments that where performed under different conditions, (from [43]). In order to compare our k-ary clustering to the binary clustering we used the MIPS complexes categories (http://www.mips.biochem.mpg.de) We focused on the 979 genes that appeared in both the dataset and the MIPS database. In order to compare the binary and 4-ary results, we have selected a similarity threshold (0.3, though similar results were obtained for other thresholds), and used this threshold to determine the clusters indicated by each of the trees. Starting at the root, we went down the tree and in each internal node looked at the average similarity of the leaves (genes) that belong to the subtree rooted at that node. If the average similarity was above the selected threshold the subtree rooted at that node was determined to be a cluster. Otherwise, we continued the same process using the sons of this internal node.

This process resulted in 36 distinct clusters for the binary tree and 35 for the 4-ary tree. Next, we used the MIPS complexes to compute a p-value (using the hyper geometric test) for each of the clusters with each of the different complexes, and to chose the complex for

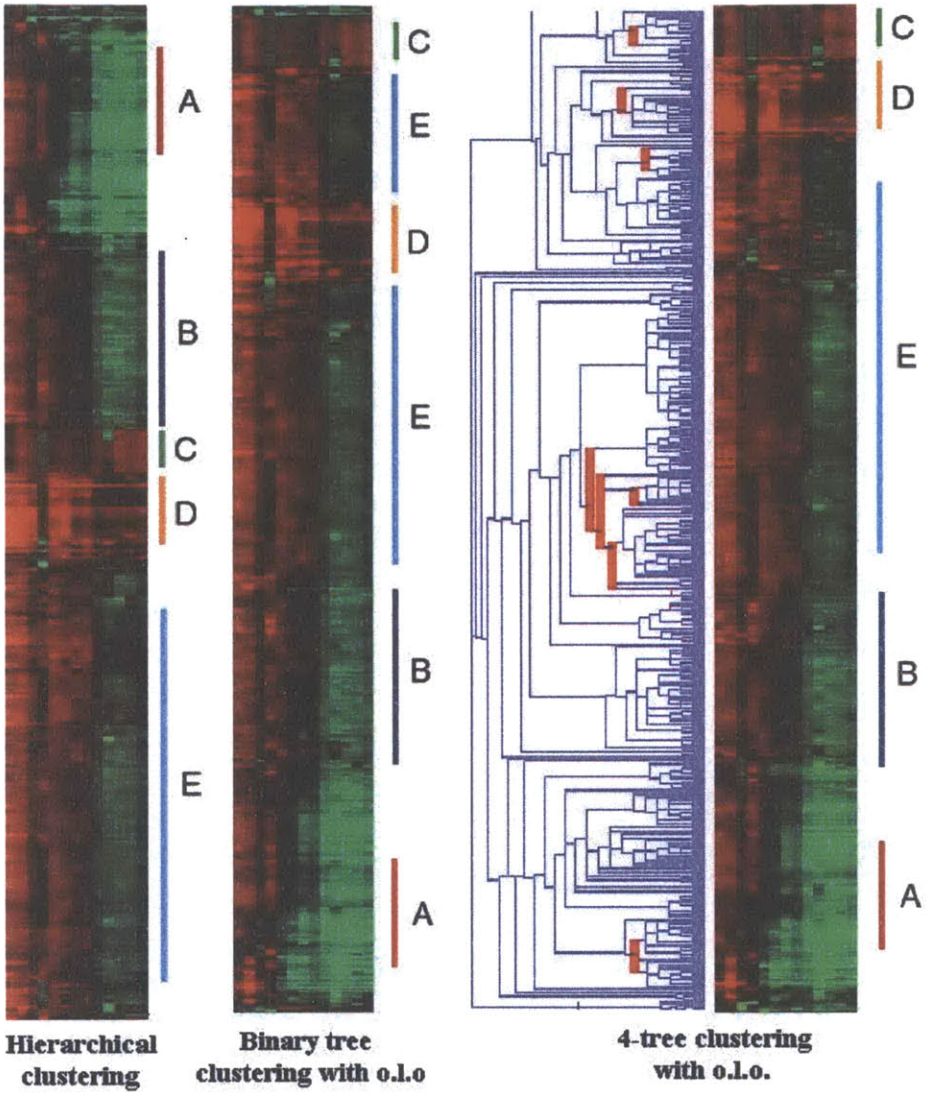


Figure 7-5: Color comparison of the three different clustering algorithms using the *myc* overexpression dataset. The left hand side is the figure that appears in the original paper containing the five different clusters that where identified. Note that by using the k-ary clustering algorithm we where able to both identify and order the clusters correctly (based on the required *myc* gene expression). The tree in the figure is the 4-ary tree where some of the nodes that contain less than 4 children are highlighted. See text for more details

which the cluster obtained the lowest p-value. Finally, we looked at all the clusters (in both trees) that had both more than 5 genes from their selected complex and a p-value below 10^{-3}.

The results seem to support our assumption that the 4-ary tree can generate more meaningful biological results. The above process resulted in 10 significant clusters in the binary tree, while the 4-ary tree contained 11 significant clusters. Further, as shown in Table 7.4.3, the clusters generated by the 4-ary algorithm where, on average, more specific than the binary clusters. Out of the 7 complexes that where represented in both trees, the 4-ary tree contained 4 clusters for which more than 50% of their genes belonged to a certain complex, while the binary tree contained only two such clusters. In particular, for the Proteasome complex, the 4-ary tree contained a cluster in which almost all of its genes (28 out of 29) belonged to this complex, while for the corresponding cluster in the binary tree was much less specific (28 out of 39 genes). These results indicate that our k-ary clustering algorithm is helpful when compared to the binary hierarchical clustering algorithm. Note that many of the clusters in both the binary and 4-ary algorithms do not overlap significantly with any of the complexes. This is not surprising since we have only used the top level categorization of MIPS, and thus some of the complexes should not cluster together. However, those that do cluster better when using the 4-ary algorithm.

Complex	**# genes**	**binary tree**			**4-ary tree**		
		#∈ complex	**cluster size**	**p-value**	**#∈ complex**	**cluster size**	**p-value**
Nucleosomal	8	8	48	$2 * 10^{-11}$	8	15	$3 * 10^{-16}$
Respiration chain	31	11	67	$2 * 10^{-6}$	14	27	$5 * 10^{-16}$
Proteasome	35	28	39	$8 * 10^{-39}$	28	29	0
Replication	48	27	88	$5 * 10^{-18}$	30	106	$3 * 10^{-19}$
Cytoskeleton	48	15	53	$3 * 10^{-9}$	9	28	$2 * 10^{-6}$
RNA processing	107	12	26	$4 * 10^{-6}$	13	46	$7 * 10^{-4}$
Translation	208	152	231	0	144	195	0

Table 7.1: Comparison between the binary tree and 4-ary clustering algorithms for the complexes that where identified in both trees. In most cases (5 out of 7) the 4-ary results where more significant than the binary results. See text for complete details and analysis.

7.4.4 Comparing the sum of similarities for different trees

For each of the three datasets described above, we have also computed the sum of similarities of adjacent leaves in the orderings of the tree ($M(v, *, *)$ for the root node v), for the three different ordering strategies (binary clustering, binary with o.l.o. and 4-ary with ordering). In all cases the ordering of the trees (both in the binary and 4-ary cases) significantly improved this sum when compared to the random order (by 4% on average). However, the difference between the binary and 4-ary orderings was not significant. In two of the three datasets the 4-ary algorithm achieved a higher value, while for the third the binary ordering had a higher value. This could be attributed to the fact that the 4-ary algorithm can place highly similar genes in different subtrees, if their similarity is the results of noise. This increases the robustness of the clustering, but at the same time decreases the sum of similarities of neighboring leaves.

7.5 Extensions: optimal ordering for lossless image compression

While the optimal leaf ordering algorithm discussed in this section was motivated by problems in computational biology, it can also be applied to other problems in computer science. Here we present an example of such a problem, which benefits from introducing optimal ordering for trees.

In image processing and computer graphics, trees are used to represent, store and retrieve images, and to synthesize new textures from a sample input [15]. One of the most popular lossless image compression algorithms, GIF, works on a linearized version of the input image. Since GIF uses the Lempel-Ziv (LZ) compression algorithm [74], any improvement to the scanning method used can improve GIFs' compression rates. Many algorithm have been suggested for this task [35], but, as we have shown in a recent paper [13], a scanning method that represents the image as a tree, and reorders the pixels using a variant of the optimal leaf ordering algorithm from Section 6 outperforms all previous methods. The algorithm we present for this task solves a restricted version of the optimal ordering problem which we call level ordering. Unlike optimal ordering, the level ordering problem asks to optimally order each level in the tree while holding the levels above it fixed. This allows us to reduce the running time while still achieving a good ordering of the leaves. We present an

algorithm that solves the level ordering algorithm (for quadtrees) in $O(n)$ time and space. This algorithm allows us to order any image, regardless of its size.

To retrieve the original image, a context-based pixel ordering (such as the optimal level ordering) necessarily needs to associate the specific ordering with the given image. Encoding the associated ordering imposes a space overhead. To achieve a compact encoding of the ordering, we use the ordering of upper levels in the tree to *learn* an ordering for lower levels. Specifically, our context-based pixel ordering technique is shown to be effective for lossless image compression even when the cost of encoding the ordering is included.

In [9], we show that by ordering the leaves of the image tree we are able to present first context based scanning algorithm that actually improves GIF image compression. Our algorithm improves the compression rates by up to 9% when compared to the line scan result, and up to 4% compared to the best previous scan result. Since lossless compression rates are usually between 20-40%, this is a significant improvement.

7.6 Summary

In this section we have presented an algorithm for generating k-ary clustering tree, and ordering the leaves of this tree so that the pairwise relationships are preserved despite the increase in the number of children. Our k-ary clustering algorithm runs in $O(n^3)$. As for ordering, we presented an algorithm which extended our binary optimal leaf ordering algorithm to the task ordering k-ary trees. Our algorithm has a running time of $O(4^k n^3)$ and its space complexity time is $O(kn^2)$, which improves upon previous space requirements.

We presented several examples in which the results of our algorithm are superior to the results obtained using binary tree clustering, both with and without ordering. For synthetic data our algorithm was able to overcome missing values and correctly retrieve a generated structure. For biological data our algorithm is better in both identifying the different clusters and ordering the dataset.

While clustering algorithms are useful to draw initial conclusions, and to detect expression programs, these algorithm cannot determine the specific pathways and networks involved in the process being studied. Thus, in the next part we present a new algorithm which combines expression and binding data to discover complex biological networks for various systems in the cell.

Part III

Modeling Biological Systems

Chapter 8

Computational discovery of gene modules and regulatory networks[1]

One of the main goals of computational molecular biology is to determine the structure and function of complex networks in the cell. Using such networks we can accomplish many different objectives. For example, regulatory networks provide valuable information about the organization of the cell in the molecular level, which is one of the long term goals of molecular biology [5]. In addition, such networks can be used to determine possible causes of different diseases or developmental problems. For drug discovery, these networks allow us to determine proteins that can serve as potential drug targets. While such targets can be determined by a number of methods, using networks we can identify not only appropriate targets, but also the side effects of targeting a specific protein. This can both reduce the expenses and the time it takes to develop new drugs and to bring them to market.

While computational analyses of transcriptional profiling data have provided many insights into biological pathways by identifying genes with similar patterns of expression, these methods are unable to distinguish among genes that despite similar expression patterns differ in their cellular function and in their mechanisms of regulation. A variety of new high-throughput data sources are becoming available [97, 73, 91], and these hold the promise of revolutionizing molecular biology by providing a large-scale view of the regulation of genes in the cell. Fundamental goals at this scale involve discovering patterns of combinatorial regulation and how the activity of genes involved in related biological processes

[1]This chapter is based on references [21] and [73]

is coordinated and interconnected. However, each high-throughput data source measures only a particular aspect of cellular activity and suffers from limitations in accuracy. Thus, an important goal is to integrate information from multiple data sources, so that each type of data can compensate for the limitations of the others. A further goal is to develop methods that can aid in deducing abstractions that can conceptually reduce genetic network complexity without significant loss of explanatory power.

In this section we present a method that combines expression data with in vivo protein-DNA binding data for module discovery, and thus provides direct access to underlying biological mechanism.Modules are groups of genes that are both co-expressed and bound by the same set of transcription factors. We present the GRAM (Genetic RegulAtory Modules) algorithm, which identifies genetic modules that share common regulators as well as expression profiles by combining direct information from DNA binding experiments with functional information from expression experiments. Our method exhaustively and efficiently searches the entire combinatorial space of subsets of transcription factors to discover what factors control what genes.We discover that genes grouped together in previous work [65, 81] are, in fact, controlled by different transcription factors. This allows us to accurately reconstruct the underlying networks used by the cell to control gene expression programs under different conditions.

GRAM is both efficient in combining information from large complementary data sets and robust as it makes few assumptions about the underlying data. It uses genomic expression and transcription factor protein-DNA binding data sources to discover gene modules. Importantly, GRAM performs an efficient exhaustive search over all possible combinations of transcription factors implied by the protein-DNA interaction data, applying a stringent criterion for determining binding. Once a set of genes bound by a common set of transcription factors is found, the algorithm identifies a subset of these genes with highly correlated expression, which serves as a *seed* for the module. The algorithm then seeks to add additional genes to the module that are similarly expressed and would be considered bound by the same set of transcription factors if a more relaxed binding criteria were used.

We applied our GRAM algorithm to a collection of genomic binding experiments profiling 106 Saccharomyces cervisiae transcription factors in rich media conditions [73] and a second data set of over 500 expression experiments profiling yeast cells under a variety of conditions. Complete details on the data used are available on the supplementary web

site [17]. We use the discovered modules to build a regulatory network of transcription factors and modules, and also use modules to label transcription factors as activators or repressors and identify patterns of combinatorial regulation. Further, we present a method for using modules to build automatically genetic regulatory sub-networks for specific biological processes, and use this to reconstruct accurately key elements of the amino acid starvation response, the cell cycle system in yeast and the regulation of genes following a treatment with rapamycin. Finally, we validate the quality of the results obtained with our module discovery algorithm, by performing analyses using four independent data sources.

8.1 Related work

Recently, there have been many papers that focus on discovering genetic regulatory networks. While I will mention many of these papers in this section, this is by no means a comprehensive overview of systems biology references. See [61] for a recent review.

A number of papers focused on networks derived from one type of data, usually genes expression or protein interaction data. Friedman *et al* [47] used expression experiments performed on yeast cells to learn static Bayesian networks and subnetworks for the regulation of gene expression in yeast. Hartemink *et al* [55] used Gal related expression dataset, and extended Bayesian networks to model the types of relationships between proteins and their targets. While these approaches work in some cases, in general it is hard to infer the structure of the network by using only indirect (expression) information.

Other papers combined binding and expression data, though unlike the approach presented in this section, they have used a strict cutoff for the binding data, reducing it to binary relationships based on the computed p-value. For example, Hartemink *et al* [56] used binding data to constrain the structure of the learned static Bayesian network. Idekker *et al* [64] combined protein interaction and binding data to construct the network structure, and then used expression data to identify specific subnetworks in that network. Unlike these two approaches, we do not rely on the strict threshold alone. We use expression to allow us to overcome problems associated with a strict p-value cutoff (such as a low true positive rate [73]). Thus, we allow relationships that cannot be detected using these two approaches.

A number of methods have been recently suggested in order to combine DNA motifs

with gene expression to discover the different transcriptional modules that are activated in response to external conditions [65, 81]. These methods start with an initial set of genes that are selected using a certain criteria (DNA binding motif, functional category) and use expression data to refine the initial set. While these methods represent an important first step, the method presented in this paper improves upon them in several ways. First, our method exhaustively and efficiently searches the entire combinatorial space of subsets of transcriptional factors. This allows us to distinguish modules that were found to be identical in previous work but that are controlled by different transcription factors (see Section 8.4). Second, unlike previous methods, our method comprehensively combines the two data sources (binding and expression data) by revisiting the binding data after refining the initial set. Finally, our method focuses not only on the genes themselves but also on the relationships between the factors binding to these genes and the bounded set of genes. This allows us to discover the way in which the cell responds to the external conditions by assigning functional annotations to the different transcriptional factors.

Note that all the above approaches (and several others that where not mentioned above) generate networks that represent a static (or a steady state) version of gene expression. In contrast, as we show in the following section, our algorithm allows us to represent a dynamic network, by combining our module discovery algorithm with our interpolation and alignment algorithms from Section I. This is especially important for networks that model dynamic systems in the cell, such as the cell cycle system, which is one of the examples we present in section.

8.2 The GRAM algorithm

In this section we present the GRAM algorithm for discovering gene modules. As mentioned above, modules are sets of genes that are both co-expressed and regulated by the same set of transcription factors (TFs). Together with the factors that regulate these modules, they can serve as a basis for a network that represents general regulation of gene expression in the cell. The computational challenge is to integrate the large binding and expression datasets to identify such modules.

In order to reduce the assumption the algorithm makes, and to allow an exhaustive search over the space of potential sets of regulators, we do not assume that the expression

patterns of genes in a specific module are directly effected by the expression profiles of the factors regulating these modules. Indeed, in many cases TFs are post transcriptionally modified, and since we currently cannot examine the protein levels of these genes, we cannot eliminate regulation inference based on expression data alone. For example, the expression profiles of four out of the nine cell cycle TFs are not cycling [92], while there is evidence that by knocking out some of these non cycling factors the expression levels of many cell cycle regulated genes are altered(see [107] and Section 4). Thus, instead of basing our regulation decision on expression data, we base it on binding data. However, since the binding data is noisy, we complement it with expression data, by using a reduced p-value cutoff for genes that have similar expression patterns to genes that are bound by a set of factors (based on a strict p-value). Thus, in addition to discovering gene modules, our algorithm can discover new regulation relationships that are missed when using a strict p-value cutoff.

8.2.1 Identifying gene modules

The GRAM algorithm exhaustively searches the space of all possible sets of regulators (see below). For each such set F, our algorithm involves three stages (see Figure 8-1). In the first stage, the set of all genes G that are bound using a strict p-value by all of the factors in F are selected. In the second stage, we look at the expression profiles of the genes in G, and determine an expression center c for which the ball in expression space defined by c and a given radius r (denoted $B(c,r)$) contains as many of the genes in G as possible. Finally, in the third stage we look for genes with a *combined* p-value below a certain threshold, and an expression profile contained in $B(c,r)$. These are the set of genes that we include in the module regulated by F.

Below we discuss in detail if the the three stages of the algorithm.

Initial strict set

Let b be a matrix of binding p-values, where rows correspond to genes and column correspond to transcription factors, so that $b_{i,j}$ denotes the binding p-value of TF j to gene i. Let $T(i,p)$ denote the set of all transcription factors that bind to a gene i with p-value less than p, i.e., the list of factors j such that $b_{i,j} < p$. Let $F \subseteq T(i,p)$ denote a subset of the transcription factors that are bound to i. Let $G(F,p)$ be the set of all genes to which all the factors in F bind with a given significance threshold p. That is, all the genes k such that

```
GRAM(E, B, p1, p2) { // E and B are the expression and binding datasets respectfully
    For all genes i {
      Set T = T(i, p1) // see text
      // Exhaustively search the regulators space for subsets controlling modules
      For all F ⊆ T s.t. F has not been previously explored {
        // Step 1: Find the initial strict set
        Let G = G(F, p1)
        Let n = |G| and s_n be the similarity threshold for n
        // Step 2: Find the expression core set
        Find the expression center c for G and s_n // see text
        // Step 3: Expansion by relaxing the p-value cutoff and using expression core
        Set M(F) = genes j s.t. d(e_j, c) < s_n and p_{F,j} < p1
        // Report the module that was found
        if M(F) > min size, output F, M(F)
        Mark all genes in M(F) so that they are not considered for F' ⊂ F
      }
    }
}
```

Figure 8-1: GRAM algorithm for identifying gene modules.

$F \subseteq T(k,p)$. The GRAM algorithm begins by going over all genes i. Using a strict p-value cutoff p_1, we search over all subsets $F \subseteq T(i,p_1)$ that have not been explored in previous steps for modules regulated by the factors in F.

Note that we are working with a dataset containing over 100 TFs. Thus, there are potentially exponential number of subsets of factors of size 10. However, since we initiate our search at the genes themselves, the number of actual subsets we are searching is much smaller. In particular, this approach allows us to exhaustively search all possible subsets of factors that are present in the dataset we are looking at.

Expression core set

For every set of transcription factors F, the genes in $G(F,p_1)$ serve as candidates for a module regulated by F. For each such set $G(F,p_1)$ with a sufficient number of genes (we currently require at least five genes), the algorithm attempts to find an expression core set i.e. a subset of the genes in $G(F,p_1)$ that are co-expressed. Let $|G(F,p_1)| = n$, that is, there are n genes bound by all the factors in F with a p-value less than p_1. In order to find such a subset we look for a point c in expression space such that for an expression similarity threshold s_n, the ball centered at c of radius s_n contains as many genes in $G(F,p_1)$ as

possible. In order to reduce unwarranted assumptions, we do not assume any probability model for gene expression data. Instead, we rely on randomizations tests to determine a suitable radius for similar expression patterns. Randomization tests have been extensively used in computational biology, and in general seem to provide good results. Thus, s_n is computed by using a randomization test in the following way. We select at random n genes, compute c for this set (see below), and determine the distance r of the 5% closest genes that were not included in the random sampled set. This process is repeated many times (it is actually performed as a pre-processing step, for different possible sizes of n), and we set s_n to be the median r obtained in this randomization tests.

Let e_i denote the expression vector for gene i, and let $d(e_i, c)$ denote the distance between the expression profile of gene i and the selected center c. Denote by $C(F, p_1, c)$ the set of genes in $G(F, p_1)$ that are all at distance at most s_n from a given point c in expression space. In order to find the best center, we would like to solve the following maximization problem:

$$max_c |G(F, p_1, c)| \tag{8.1}$$

Unfortunately, the exact solution of this problem is exponential in the dimension of the expression space [10]. Since our algorithm is intended for large datasets of expression experiments, such a search is prohibitive. Instead we use an approximation algorithm, that has good theoretical guaranties, and runs in time $O(n^3)$. Note that since n is the number of genes in $G(F, p_1)$, $n << n_g$ the total number of genes (since only a small number of genes are bound by all the factors in F), and so such a running time is reasonable.

The algorithm we use is very simple. We go over all triplets of genes from $G(F, p_1)$, and for each such triplet we compute the center for this triplet c, and the number of genes in $G(F, p_1)$ that are contained in $C(F, p_1, c)$. We select the point c which maximizes $|C(F, p_1, c)|$ over all the centers we have looked at for all triplets.

In order to present the theoretical guarantees of this algorithm we cite a lemma from [10]. Let P be a set of points in a high dimensional space. For $S \subseteq P$ let $B(S)$ denote the smallest ball enclosing all the points in S, let $c_{B(S)}$ be the center of such ball and let $r_{B(S)}$ be its radius. Then the following lemma is proved in Badoiu and Clarkson [10].

Lemma 8.2.1 [Badoiu and Clarkson 03] *There exists a set $S \subseteq P$ of size $2/\epsilon$ such that the*

distance between $c_{B(S)}$ *and any point* $p \in P$ *is at most* $(1+\epsilon)r_{B(P)}$.

Set $\epsilon = 2/3$. According to the above lemma, there are three points in the subset we are looking for, for which the ball defined by the center of these three points and a radius of $(1+\epsilon)s_n$ contains all the points in the set we are looking for. Since we are going over all possible triplets of points, we are guaranteed to encounter the three points from the above lemma. In other words, our algorithm finds a solution which is at least as good as a solution that can be found for $n^* = s_n/(1+2/3)$.

Final module

Now that we have a center for the expression profile of the module, we expand the set of genes included by re-visiting the p-value table to detect genes that where omitted due to noise in the p-value measurements. This is done in the following way. For a gene i, we first check if $d(e_i, c) < n_s$, and if so we move on to test the binding of the factors in F to gene i. First, if there exists a factor $f \in F$ such that $b_{i,f} > p_2$, i will not be added to the module. p_2 is determined using a set of independent chromatin IP experiments. Next, for all genes that pass these two requirements (expression similarity and a p_2 or less p-value with all factors), we compute the combined p-value for the factors in F. This is done using the Fischer method [46] which can be described as fellows. If each p-value p_f has a uniform distribution between 0 and 1, then $-2\log p_f$ has a χ^2 distribution. Thus, if the tests for each factor are independent [1] then for gene i

$$-2\sum_{f\in F}\log p_{i,f}$$

has a χ^2 distribution with $|F|$ degrees of freedom. Using the χ^2 distribution we can compute a new combined p-value based on the above sum. Denote this p-value by $p_{i,F}$. Then we add i to the module $M(F)$ if $p_{i,F} < p_1$ (where p_1 is our initial strict p-value).

Following this step, we arrive at a module that contains similarly expressed and co-regulated genes. We repeat the above process for all possible subsets of TFs, until we determine the list of all modules contained in the datasets we are looking at.

[1] This assumption is not entirely true since experimental conditions can affect some genes in all experiments. However, this simplifying assumption makes computation easier, and allows us to capture the essence of the results

Note that if a gene i is included in a module controlled by a set of TFs F, it is likely to be included in any module controlled by a subset $F' \subset F$. Since we are interested in the complete set of factors controlling a gene, we do not gain anything from including i in $M(F')$. Thus, prior to computing the set of genes contained in $M(F')$, we filter out any genes that have already been found to included in a module controlled by a superset of F'. This reduces the number of overlapping modules, without reducing the explanatory power of this approach.

8.2.2 Sub-network discovery algorithm

Our sub-network discovery algorithm extends the GRAM to the task of inferring subnetworks for specific dynamic and static systems in the cell. For a given system, the GRAM algorithm is run on the full set of transcription factors and a large set of expression data (containing expression data that specifically studies this system, and expression data under other conditions). In the second step, genes in the generated modules are flagged if they are determined to be involved in a given biological process based on an objective criteria. For instance, the criteria could be a fold change above a threshold in a set of relevant expression experiments or a more complex criteria such as cycling behavior in a time series gene expression experiment. A statistical test based on the hypergeometric distribution is then used to determine which modules contain a significant number of flagged genes. The transcription factors that regulate these significant modules are then collected into a list. We then select a set of expression experiments in which the biological process of interest is expected to be particularly active. Finally in the third step, our module discovery algorithm is run using this expression data, the flagged genes, and the list of regulators determined in the previous step, producing a set of modules with genes and factors presumably directly involved in the process of interest.

In order to present a dynamic model for a subsystem, we combine the above algorithm with our interpolation and alignment algorithms from Sections 2 and 3. After determining the modules for the subnetwork, we interpolate the expression profiles of the genes in the discovered modules. Next, we select one module as an anchor (or time point 0), and align the rest of the modules to this module using our continuous alignment algorithm. Unlike the alignment discussed in Section 3, in this case we only allow a shift between the two sets that we align, and thus we can use the shift parameter to determine the actual starting

time (with respect to the selected time point 0 module) of the aligned module. We repeat this process for all modules, resulting in a temporal ordering of the discovered modules. Note that this way we can determine the actual activation time for factors, even if their expression profiles do not change under the experimental conditions, based on the time of the modules they regulate. This allows us to correctly assign factors to different stages in the system, without directly observing their protein levels.

8.3 Results

In this section we discuss results that are based on a number of applications of our GRAM algorithm. First, we have applied our module discovery algorithm to a large collection of gene expression experiments in order to identify modules for gene regulation in rich media conditions. Next, we have verified the results obtained by our modules discovery algorithm using a variety of other high throughput data sources (including sequence and literature data) and a set of independent biological experiments. Finally, we have used our subnetwork discovery algorithm to discover subnetworks for a number of systems in the cell, including the cell cycle system and amino acid starvation. In collaboration with the Young laboratory at the Whitehead Institute we have also explored a novel network for gene expression in the Tor pathway using a collection of binding experiments that where performed after treating cells with rapamycin.

8.3.1 Gene modules in rich media conditions and modes of regulatory control

The GRAM algorithm was applied to genome-wide binding data for 106 transcription factors and over 500 expression experiments (complete details on the data used are available on the supplementary web site [17]).

In order to determine the values for p_1 and p_2 we have performed independent experiments to test the prediction of the binding data at a number of different confidence levels. We used gene-specific PCR analysis with selected regulators to test the results predicted at each of the different p-value thresholds. We could then determine how frequently a regulator-gene interaction is correctly predicted at different p-values threshold.

We selected two different regulators, Nrg1 and Stb1, for further testing. For each reg-

ulator, we selected sets of genes with p-values closest to one of four thresholds (0.001, 0.005,0.01,0.05) and performed chromatin IP and gene-specific PCR. See http://web.wi.mit.edu/young/regulat for complete results.

Based on these independent experiments, we have used $p_1 = .001$ and $p_2 = 0.01$. For these values we have determined that p_1 achieves a 70% true positive rate and a 5% false postive rate, while p_2 achieves over 90% true positve rate, but over 50% false positive rates. Thus, it is important to start with p_1 in order to reduce the false positive rate, but using only the strict p-value of p_1 reduces the number of true positives that can be determined. By complementing binding with expression data, we can use the higher cutoff (p_2) and thus increase the true positive rate without increasing the false positive rate, as we show below.

One-hundred eight modules were found that correspond to a variety of biological functions (see Figure 8-2 and supplementary data [17]) and in many cases, modules that share a biological function have overlapping transcriptional regulators. The modules contain 659 distinct genes and are regulated by 68 of the transcription factors. The discovered modules are typically controlled by multiple transcription factors, and in a number of cases these factors were previously known to interact including Hap2/3/4/5, Hap4/Abf1, Ino2/Ino4, Hir1/Hir2, Mbp1/Swi6, and Swi4/Swi6. These results provide evidence that the GRAM algorithm identifies not only biologically related sets of genes, but also relevant factors that are interacting to control the genes. By integrating binding and expression data the GRAM algorithm improves on either data source alone as we show in Figures 8-3 and 8-4.

Identifying activators and repressors

While genomic binding data is very useful in helping to determine which transcription factors regulate a gene, it does not provide information on the type of regulation being exercised. In particular, it cannot tell us whether factors act as activators or repressors. Our approach to identifying activators/repressors was to search for transcription factors with expression profiles that are positively/negatively correlated with the profiles of regulated genes (activators/repressors). We determined the significance of these correlations by computing correlation coefficients between all factors and all modules and taking the 5% tail of the distribution of absolute correlation coefficients. Out of the 108 modules, 28 (26%) were bound by a factor determined to be either an activator or repressor. Tables 8.1 and 8.2 present the eleven activators and five repressors determined using the method de-

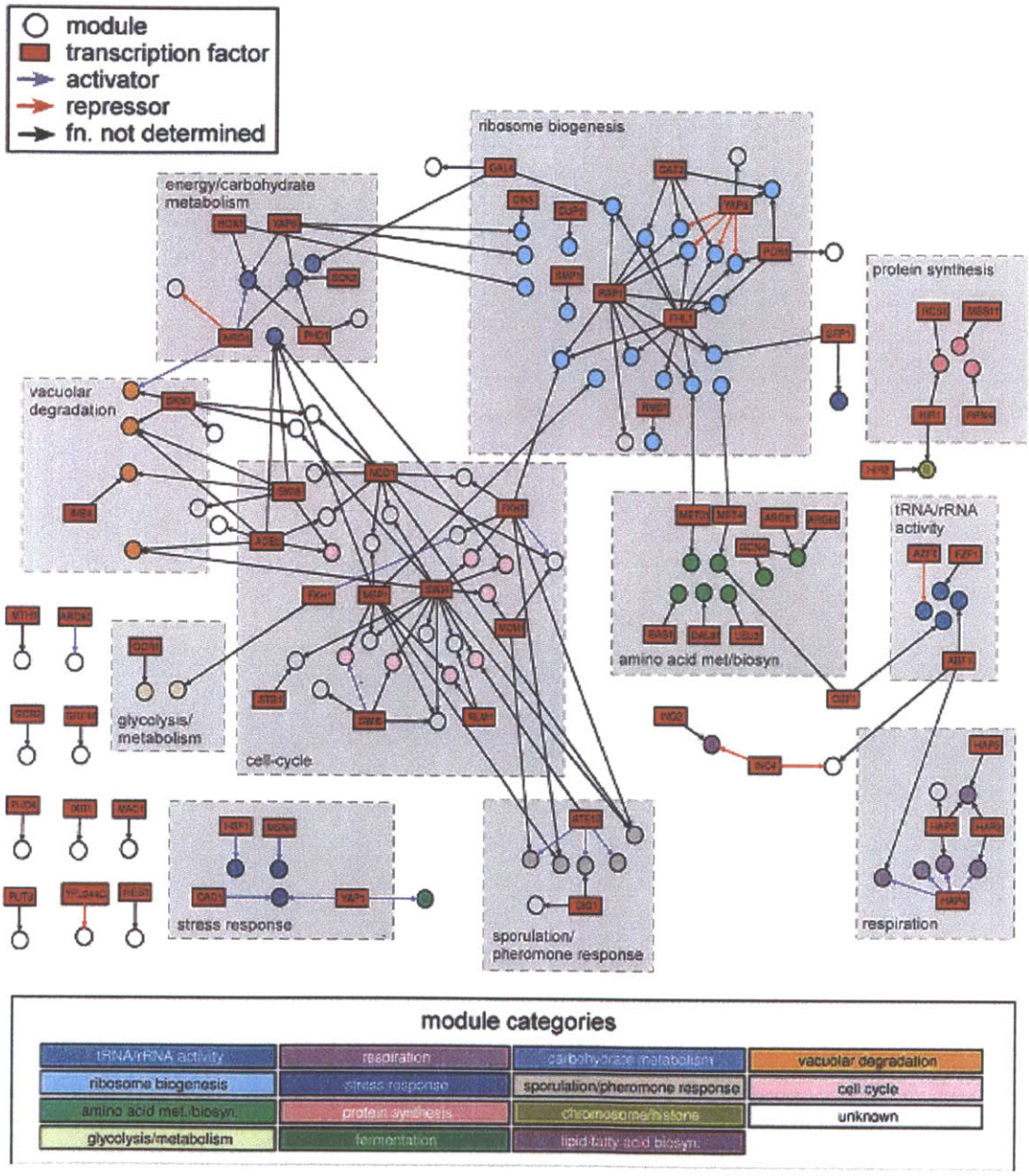

module
transcription factor
activator
repressor
fn. not determined
ribosome biogenesis
energy/carbohydrate metabolism
protein synthesis
vacuolar degradation
tRNA/rRNA activity
amino acid met/biosyn
glycolysis/ metabolism
cell-cycle
stress response
sporulation/ pheromone response
respiration
module categories
tRNA/rRNA activity
respiration
carbohydrate metabolism
vacuolar degradation
ribosome biogenesis
stress response
sporulation/pheromone response
cell cycle
amino acid met./biosyn.
protein synthesis
chromosome/histone
unknown
glycolysis/metabolism
fermentation
lipid/fatty acid biosyn.

Figure 8-2: Rich media gene modules network - Visualization of the network of modules and regulatory factors reveals that there are many groups of connected modules/regulators involved in similar biological processes. The network consists of 108 modules containing 659 distinct genes regulated by 68 factors. The directed arrows point from transcription factors to the modules that they regulate. Modules are colored according to the MIPS category to which a significant number of genes belong (significance test using the hypergeometric distribution $p < 0.005$). Modules containing many genes with unknown function or an insignificant number belonging to the same MIPS category are uncolored. When the discovered modules were compared to results generated using the binding data alone, the module discovery algorithm yielded an almost three-fold increase in modules significantly enriched for genes in the same MIPS category.

scribed above (see Also Figure 8-5). Ten of the eleven activators were already identified in the literature; the literature offered less information about the five repressors. One of these repressors was previously identified, while in at least one case, a factor may serve in both activator and repressor roles under different conditions. Ino4 and Ino2 are thought to dimerize and activate genes in low inositol conditions, but while Ino4 is required for binding it apparently does not affect activation. In our analysis, the highest degree of negative correlation occurs in stress condition expression experiments, suggesting that Ino4 might serve as a repressor under certain conditions and an activator under others.

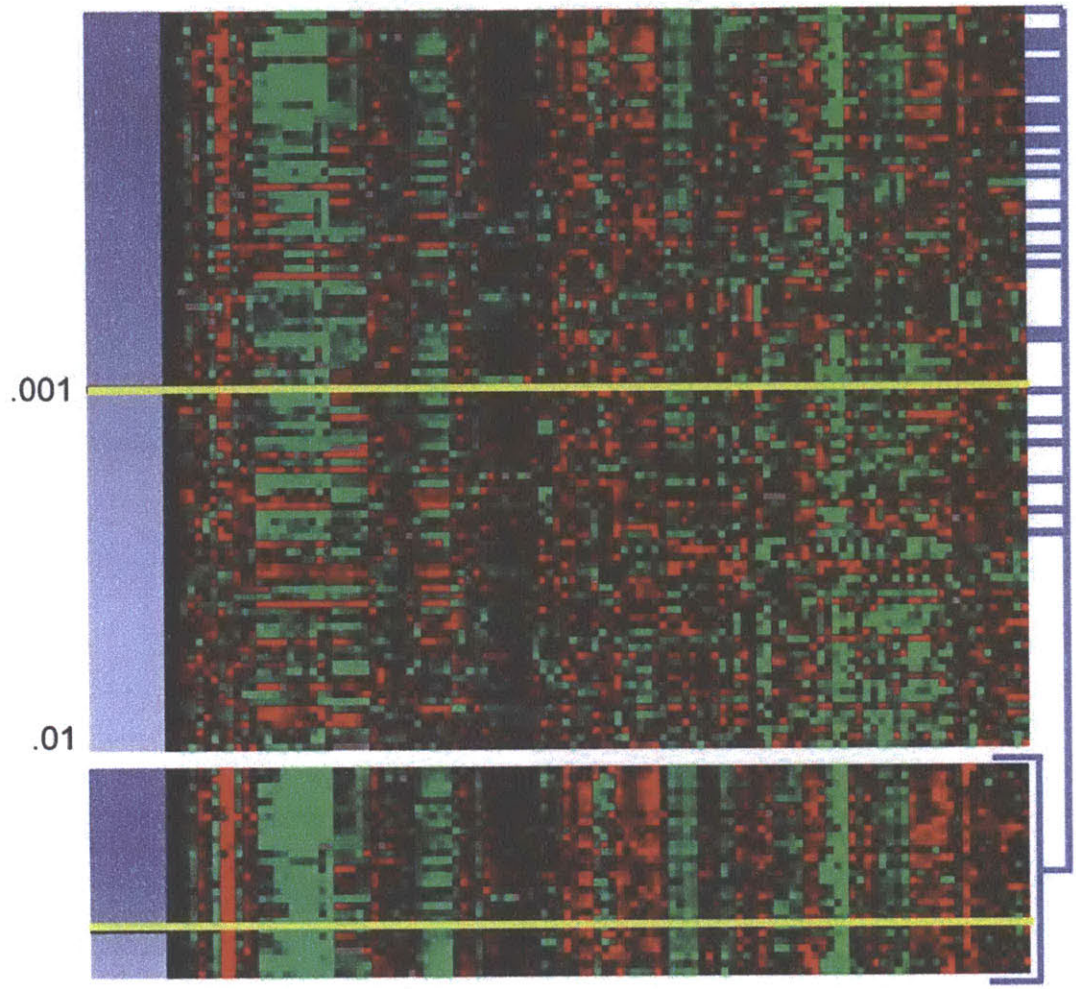


Figure 8-3: Binding data: the GRAM algorithm can improve the quality of DNA-binding information, since it uses expression data to avoid a strict statistical significance threshold. Shown is DNA-binding and expression information for the 99 genes bound by Hap4 with a p-value ¡ .01 using the statistical model in Lee et al. The blue-white column on the left indicates binding p-values, and the horizontal yellow line denotes the strict significance threshold of .001. As can be seen, the p-values form a continuum and a strict threshold is unlikely to produce good results. The blue horizontal lines on the right indicate genes that were selected for modules by the GRAM algorithm. As can be seen, most have a p-value ¿ .001, but some have p-values below this threshold. The lower portion of the figure shows the genes selected by the GRAM algorithm only, and it can be seen that they exhibit coherent expression. Further, all the selected genes are involved in respiration.

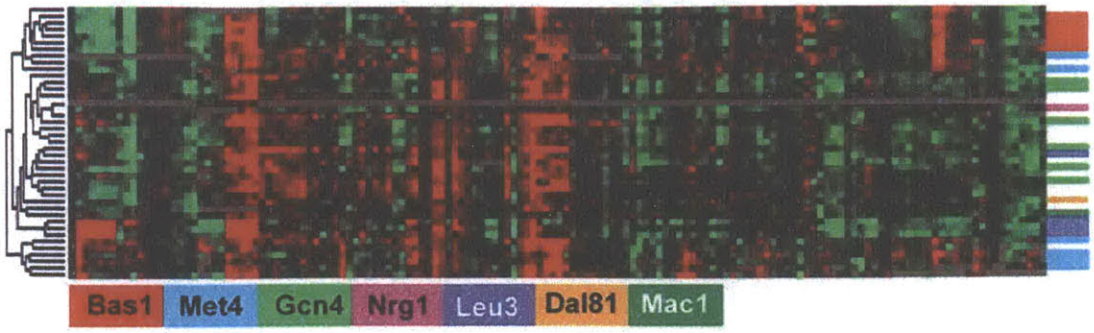


Figure 8-4: Expression data: the GRAM algorithm can assign different regulators to genes with similar expression patterns that cannot be distinguished using expression clustering methods alone. Hierarchical clustering of expression data was used to obtain the sub-tree on the left. On the right, the regulators assigned to genes by the GRAM algorithm are color coded. As can be seen, many genes with very similar expression patterns are regulated by different transcription factors.

Activators identified by our algorithm

Factor	**Module function**	**Corr. w/ module**	**Comments**
Ste12	Pheromone response	+0.64	Activator, required for pheromone response
Hap4	Respiration	+0.60	Activator of CCAAT box containing genes
Yap1	Detoxification	+0.53	Activator, possibly involved in oxidative stress response
Nrg1	Carbohydrate transport	+0.50	Previously identified as a repressor
Fkh1	Cell cycle	+0.49	Activator of cell cycle genes
Cad1	Detoxification	+0.47	Activator, involved in multidrug resistance
Aro80	Energy and metabolism	+0.40	Activator, involved in regulation of amino acid synthesis
Swi6	Cell cycle	+0.39	Activator of cell cycle genes
Msn4	Stress response	+0.38	Activator, involved in stress response
Fkh2	Cell cycle	+0.37	Activator of cell cycle genes
Hsf1	Stress response	+0.36	Activator of heat shock related genes

Table 8.1: Eleven activators were identified by computing the correlation between the expression patterns of genes in a module and the regulators. Ten of the eleven activators were previously identified in the literature.

Repressors identified by our algorithm

Factor	Module function	Corr. w/ module	Comments
YFL044C	Unknown	-0.44	Function unknown
Azf1	rRNA transcription	-0.41	Previously identified as an activator
Nrg1	Unknown	-0.40	Transcriptional repressor for glucose gene expression
Yap5	Ribosome biogenesis	-0.39	Function unknown
Ino4	Fatty acid biosynthesis	-0.39	Previous evidence of involvement in Ino2-Ino4 dimer that activates in low inositol conditions

Table 8.2: Five repressors were identified by our algorithm. One of them was previously reported in the literature, and two have not been studied before. In at least one case, a factor may serve in both activator and repressor roles under different conditions. Ino4 and Ino2 are thought to dimerize and activate genes in low inositol conditions, but while Ino4 is required for binding it apparently does not affect activation. In our analysis, the highest degree of negative correlation occurs in stress condition expression experiments, suggesting that Ino4 might serve as a repressor under certain conditions and an activator under others.

Modes of combinatorial regulation

A central feature of eukaryotic transcriptional regulation is combinatorial control, the ability of different combinations of transcription factors to specify distinct biological outcomes [93]. We sought to determine how combinations of regulators affect expression by examining the correlation between expression profiles of genes from pairs of modules that share at least one transcription factor but have no genes in common. This analysis suggests four distinct types of coordinate regulation:

- *Additive* - the binding of additional factors increases the expression levels of the bound genes. Our analysis suggested that this is the most common type of coordinate regulation. The complete set of pairs of modules that exhibit additive control appears on the supplementary website [17].

- *Negative* - a factor serves as an activator for one module, but addition of a partner factor for a second module abolishes activation or causes repression. This results in negative expression correlation between the two modules under all conditions. For example, the module controlled by the two cell cycle activators Swi4 and Mbp1 was strongly negatively correlated with a module controlled by Swi4 and Skn7. This is plausible, since there is evidence that Skn7 acts as a repressor in the oxidative stress response in yeast [72].

- *Delayed* - one factor regulates two or more modules in a similar way, but addition of a partner factor causes expression of the genes in the two modules to be offset temporally. Thus, the average expression of the modules can be aligned after an appropriate time shift. For example, Swi6, a cell cycle transcription factor, is known to partner with both Swi4 and Mbp1 to regulate genes in the G1/S cell cycle phase. However, since Swi4 itself is regulated by Swi6, expression of the set of genes regulated by Swi6/Mbp1 occurs earlier than that of those regulated by Swi6/Swi4 (see Figure 8-6). Many other cell cycle factors exhibit this type of delayed regulation.

- *Conditional* - addition of a partner factor causes expression changes in a subset of conditions. For example, our algorithm discovered a module regulated by Met4 alone and a second regulated by Met4 and Cbf1. As shown in Figure 8-6, under many conditions the average expression profile of genes in the module regulated by Met4 is

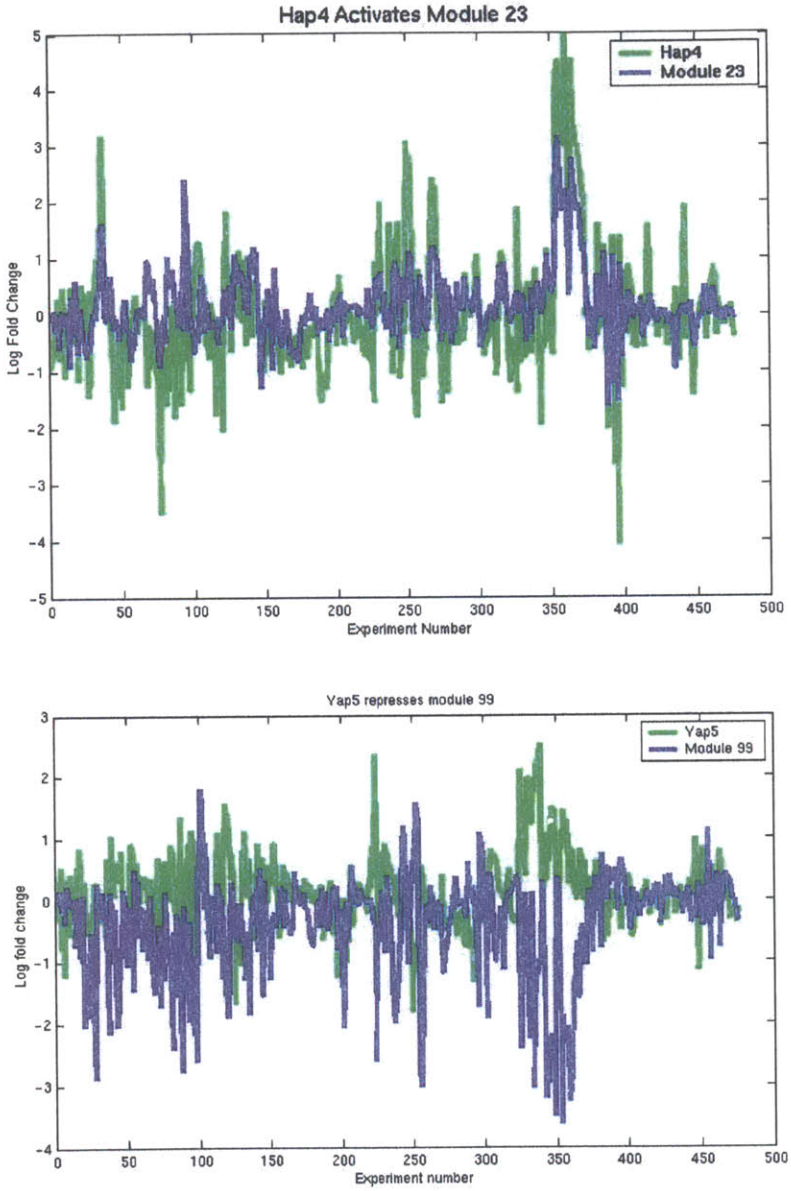


Figure 8-5: Examples of an activator and a repressor. Top: The expression levels of Hap4 correlates very well with the expression levels of the genes in the module in regulates, indicating that Hap4 is an activator. Bottom: Unlike Hap4, the expression levels of Yap5 are anti-correlated with the expression levels of the genes it regulates, indicating that it acts as a repressor. See [17] for the list of genes contained in these modules.

very similar to that of genes in the module regulated by Met4/Cbf1. However, there are some experiments in which the average expression profiles of genes in the two modules are anti-correlated, most notably under stress conditions.

8.3.2 Validation of the GRAM algorithm results using independent data sources

In this section, we investigate the GRAM algorithms ability to discriminate real transcription-factor binding events from erroneous ones, and hence improve on the results obtained using the genomic DNA-binding data alone. As with any high-throughput microarray method, considerable noise is inherent in genomic DNA-binding data. To deal with noise, Lee *et al* [73] used a statistical model for determining binding that required their trading off false-positive and negative rates. Their decision was to choose a relatively stringent p-value cutoff (.001) to reduce the false-positive rate, but this can unfortunately result in a low true positive rate. The GRAM algorithm presents a powerful alternative to using a single p-value threshold to predict binding events, since our method allows the p-value cutoff to be relaxed if there is sufficient supporting evidence from expression data (see also Figure 8-3).

Out of the 1560 unique factor-gene interactions detected by our algorithm, 627 (40%) would not have been detected using the DNA-binding data alone with a stringent p-value cutoff. One would like to show that these interactions are not erroneous by comparing them against some independent gold standard. Unfortunately, no such data source exists on a genomic scale for transcription factor-DNA interactions.

Our approach was to verify that the GRAM algorithm improves true positive rates without significantly increasing false-positive rates by using available data from four independent sources: transcription factor-gene interactions identified in the literature, conventional chromatin-IP (chIP) experiments, the MIPS database and DNA sequence motif information. We first focused on known transcription factor-gene interactions involved in the cell cycle, since this is an extensively studied system. We performed a literature search and found 51 previously identified binding relationships [17]. Seven of these binding relationships were not detected in the genomic binding assay using a stringent cutoff, but three of these were identified by the GRAM algorithm. Since our method added only 59 new factor-gene relationships for the cell cycle genes, this result was significant, with a p-value $< 3*10\text{-}5$. To further verify our results, we performed conventional chromatin-IP exper-

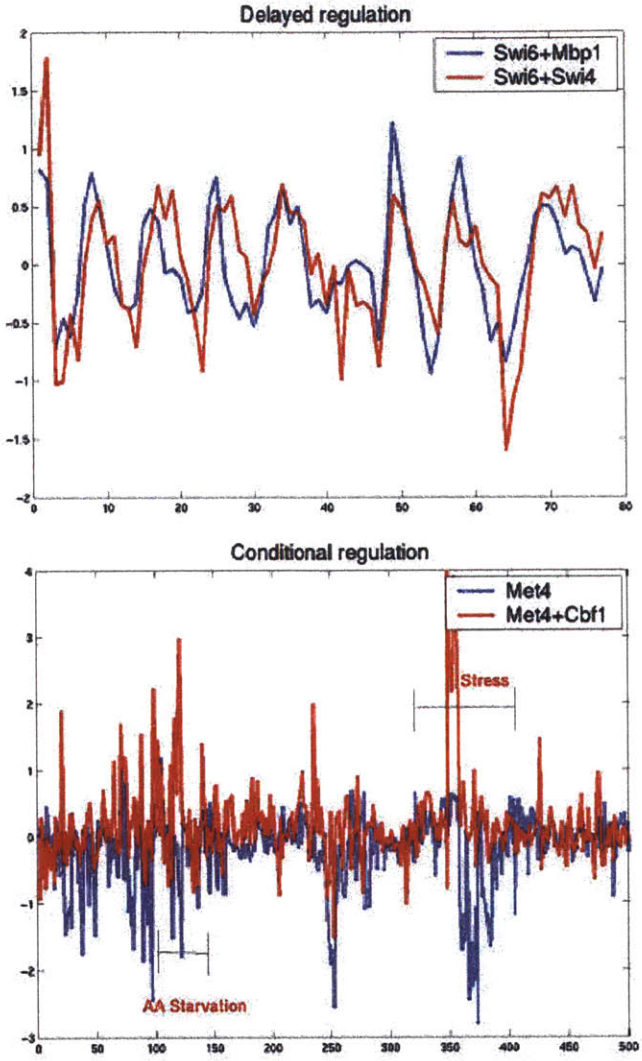


Figure 8-6: Combinatorial regulation modes - Our analysis of pairs of modules that share at least one transcription factor but have no genes in common revealed several distinct types of coordinate regulation. Examples of two such types are shown in the above figures. (a) In delayed control a factor regulates two or more modules in a similar way, but the expression of these sets of genes are temporally separated, an affect brought about by the activity of different bound partner factors. As can be seen, the average expression of genes in the module regulated by Swi6/Mbp1 lags that of genes in the module regulated by Swi6/Swi4, though both belong to the G1 phase. (b) In conditional control, a partner factor affects expression primarily in a subset of conditions. As can be seen, under many conditions the average expression profile of genes in the module regulated by Met4 is very similar to that of genes in the module regulated by Met4/Cbf1. However, there are some experiments in which the average expression profiles of genes in the two modules are anti-correlated, most notably under stress conditions.

iments for the factor Stb1 and 36 genes. The profiled genes were picked randomly from the full set of yeast genes. In these verification experiments, three additional genes were determined to be bound by Stb1 that were not detected in the genomic binding experiments using a stringent p-value cutoff. Our algorithm identified all three genes as bound by Stb1 without adding any additional genes that were determined not to be bound in the conventional chromatin-IP experiments (see Table 8.3 and [17]).

The Stb1-Swi4 Module

Orf name	**Gene name**	**Cell cycle phase**	**Binding p-value for Stb1**
YOR065W	**Hcm1**	**G1**	**0.0012**
YDR501W	YDR501W	G1	0.00002
YGR109C	Clb6	G1	0.0013
YGR221C	YGR221C	G1	0.0009
YIL140W	**Sro4**	**G1**	**0.008**
YIL141W	YIL141W	G1	0.008
YMR179W	**Spt21**	**G1**	**0.007**
YNL289W	Pcl1	G1	0.0000005
YPL256W	Cln2	G1	0.00007

Table 8.3: A module controlled by Swi4 and Stb1. Out of the nine genes contained in the module, five had a p-value higher than .001 for Stb1, and where thus not considered as bound by Stb1 in Lee *et al* [73]. However, independent chromatin-IP experiments for Stb1 confirmed the prediction of the GRAM algorithm for three out of the five genes included (HCM1, SRO4 and SPT21), while the two others were not tested. these results indicate that the GRAM algorithm can improve the true positive rate without increasing the false negative rate.

Next, we looked at the extent to which genes in the same module belonged to an identical MIPS [2] sub-category. For each module we have computed the overlap between the genes contained in that module and the different MIPS sub-categories. We have used the hyper-geometric distribution to compute a significance value for this overlap, and assigned module to their most significant category, provided that the overlap for such a category was significant with a p-value < 0.005. We repeated the same process using the binding data alone (that is, using the initial strict set instead of the modules identified by our algorithm). When these results were compared, we observed that for the modules discovered by the GRAM algorithm there was almost a three-fold increase in modules significantly enriched for genes in the same MIPS category. Thus, this results indicate that using expression we can arrive at much more coherent modules when compared to binding data on

its own.

Finally, we investigated the extent to which genes in our modules are enriched for DNA binding motifs as compared with results obtained using genomic binding data alone. We identified 34 transcription factors that regulate at least one module and have well-characterized DNA binding motifs in the Transfac database (http://transfac.gbf.de/TRANSFAC/). For each of these 34 factors, we generated a list of genes in modules bound by the factor and a second list of genes bound by the factor using the genomic binding data alone (stringent p-value cutoff of .001). We then computed the percentage of genes from each list that contained the appropriate known motif in the upstream region of DNA. As is evident from Figure 8-7, in almost all cases, the percentage of genes containing the correct motif was higher when gene lists were generated using the GRAM algorithm instead of from the genomic binding data alone.

8.3.3 Discovering networks for systems in the cell

Inspection of the global visualization of the network of modules and regulatory factors shown in Figure 8-2 suggests that there are many groups of connected nodes involved in similar biological processes. However, this global network was generated using expression data obtained under a variety of conditions and in some sense represents an average or consensus picture. One would like to be able to zoom in to obtain details on modules and factors involved in similar biological processes. For this task, we developed the sub-network discovery algorithm discussed in Section 8.2.2, that uses the globally discovered modules in combination with expression data specific to a certain biological process. This algorithm automatically infers relevant sub-networks and uses a minimum of prior biological knowledge.

We have applied our sub-networks discovery algorithm to discover three subnetworks for specific systems in the cell. We start with a sub-network for gene regulation in response to amino acid starvation. This network demonstrates the basic capabilities of our algorithm, as it uses publicly available datasets to automatically infer the network. Next, we describe a dynamic network for the regulation of gene expression during the cell cycle, which demonstrates the ability of our algorithm to infer dynamic sub-networks when high quality time series expression data is available. Finally, we present a novel network for the regulation of gene expression under rapamycin. For this network we collaborated with

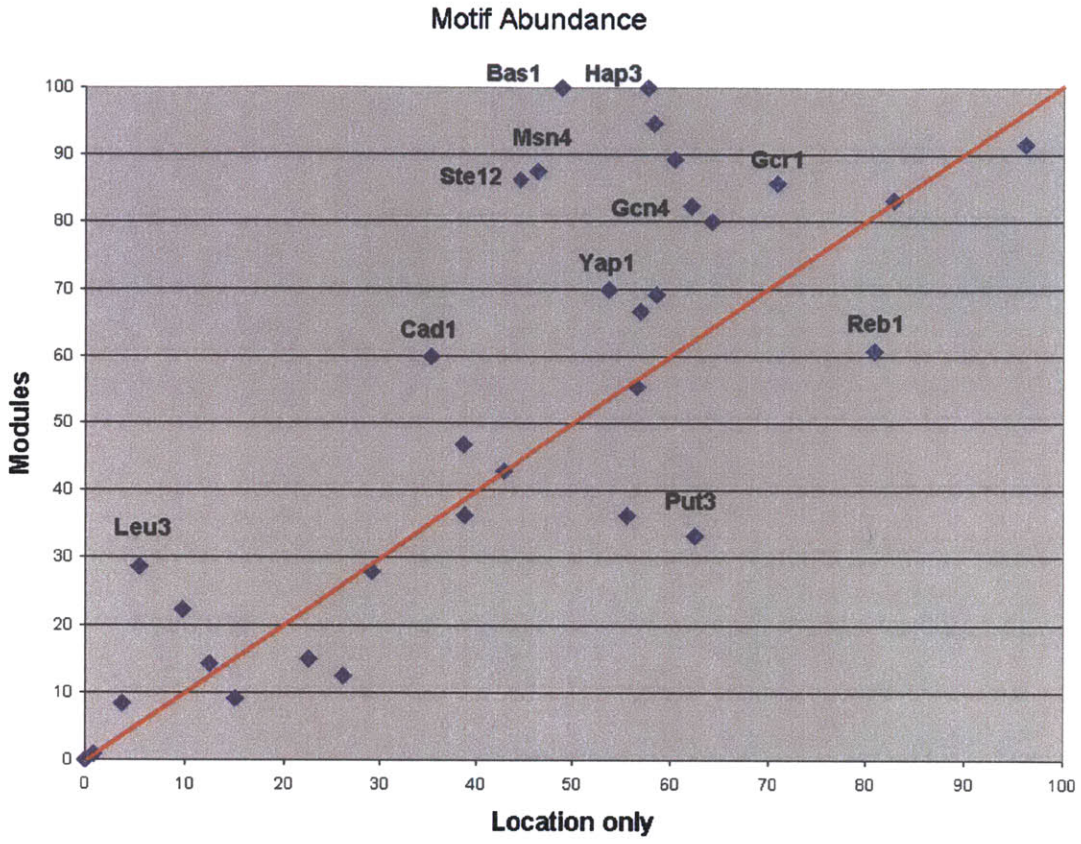


Figure 8-7: Motif discovery using the upstream regions of genes in modules discovered by the GRAM algorithm produces superior results to those obtained using genomic binding data alone. Shown is a comparison of the percentage of genes containing known regulatory motifs using modules or binding data alone. Thirty-four known motifs for factors regulating at least one module were determined using the Transfac database. For each factor with a known motif, two lists were generated: one containing genes determined bound by the factor using genomic binding data alone and the second using the GRAM algorithm results. The percentage of genes containing the appropriate motif was then calculated for each list. As can be seen, in most cases using the GRAM algorithm results produces a much better list, resulting in a higher percentage of genes containing the known motif. See supplementary web site [17] for complete results and factors names.

the Young laboratory at the Whitehead Institute to generate a new binding dataset under the appropriate condition. As we show, by using both expression and binding data that where specifically generated under these conditions, our algorithm is able to derive many new biological insights that cannot be discovered if one only uses binding in rich media conditions.

Amino acid starvation network

We applied the sub-network discovery algorithm to the task of reconstructing a regulatory network for the yeast amino acid starvation response and obtained biologically meaningful results. When yeast cells are starved of amino acids, a characteristic response occurs involving expression changes in hundreds of genes [54, 76]. Our analysis generated 16 modules containing 132 unique genes regulated by 16 transcription factors. As shown in Figure 8-8, all of the genes in the modules fall into one of six functional categories. Four of these categories involve biosynthesis of different amino acids or their components. Almost all the factors in the network and many of the genes in the modules have been previously reported in the literature to be involved in the amino acid starvation response. For instance, one module is regulated by both ARG80 and ARG81, a complex known to regulate amino acid biosynthesis. This module contains arginine biosynthesis genes such as ARG1,3,5,6 and CPA1. A module regulated by DAL81 contains genes involved in amino acid transport such as the permeases BAP2 and AGP1. Such genes are known to be induced during starvation conditions, presumably to increase uptake of amino acids. Seven of the modules contain genes involved in ribosomal function, which is not surprising, since a large number of ribosomal proteins are known to be strongly repressed under amino acid starvation conditions [76]. It is interesting to note that a number of the amino acid starvation modules discovered by our algorithm are a refined version of the transcriptional modules discovered by Ihmels *et al* algorithm [65] (see Section 8.4). This indicates that our algorithm can be used to construct the actual network employed by the cell under different conditions.

Given that most of the the binding experiments we used were performed on yeast growing in rich media, it is interesting that our algorithm was able to find a number of the modules, such as those associated with specific metabolic responses. One possibility is that the genomic binding assay is permissive, indicating not only the regions of DNA the factors bind to under ideal growth conditions but also where factors have the potential of binding.

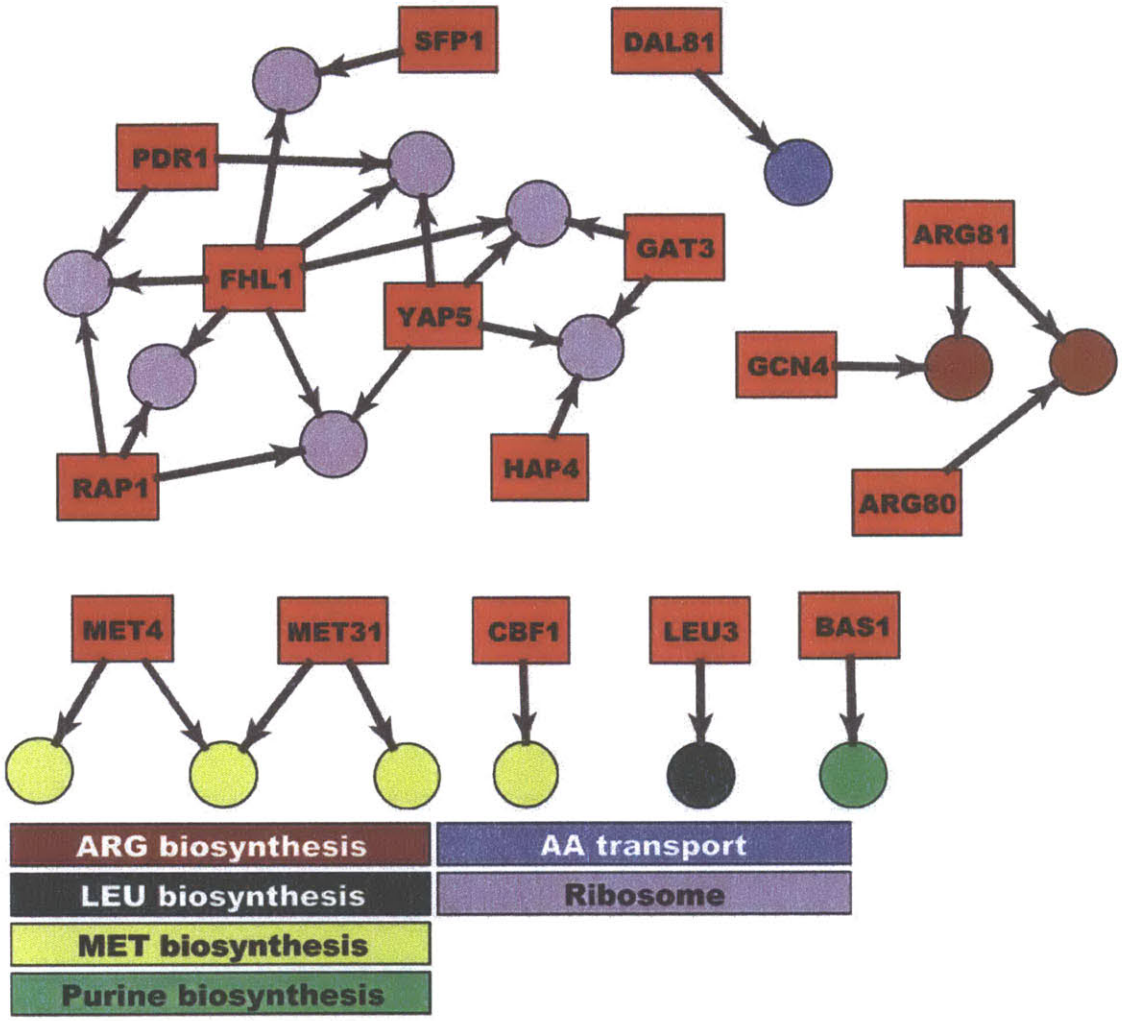


Figure 8-8: Amino acid starvation sub-network - An amino acid starvation network in yeast was automatically recovered using our sub-network discovery algorithm and a minimum of prior biological knowledge. The network consists of 16 modules containing 132 unique genes regulated by 16 transcription factors. All the genes in modules fall into one of six functional categories. Four of these categories involve biosynthesis of different amino acids or components. Almost all the factors in our network and many of the genes in the modules have been previously reported in the literature to be involved in the amino acid starvation response.

Another possibility is that some factors may bind constitutively to certain genes and require a second activating event under the appropriate conditions. It is also possible that many factors are involved in pathways that are operational in rich media conditions, but at lower levels.

Cell cycle network

When additional information is available, such as temporal expression data, even richer regulatory networks can be inferred. We applied our sub-network discovery algorithm to the yeast cell cycle in combination with a continuous temporal alignment algorithm to uncover not only modules and their regulating transcription factors, but also the temporal relationships among these modules. The cell cycle regulatory network was selected because of the importance of this biological process, the availability of extensive genome-wide expression data for the cell cycle (see Table 1.1), the extensive literature that can be used to explore features of a network model, and our interest in determining whether a principled computational approach can reproduce substantial portions of the simple network that was previously modeled using a more directed manual approach [92]. As mentioned in Section 8.2, to produce a dynamic cell cycle transcriptional regulatory network model, the modules are aligned around the cell cycle . Since the modules contain genes that are co-expressed in this process, the expression data can be used to instruct the assembly of the network to represent this temporal process. Fifteen modules containing 75 genes and regulated by 11 factors were found using the sub-network discovery algorithm (see Figure 8-9). We then applied a continuous alignment method in order to determine the phasing of the modules. A module regulated by the factors Swi5/Ace2 and containing genes known to be active at the G1/M boundary was chosen as the start of the cell cycle and the other modules were aligned against this, allowing us to localize all the modules temporally. We were then able to approximately place the boundaries for S, G2, and M and thus estimate the lengths of these phases by using prior biological knowledge about where genes in four other modules peak during the cell cycle.

Three features of the resulting network model are notable. First, the computational approach correctly assigns all the regulators to stages of the cell cycle where they have been described to function in previous studies (Simon *et al* [92] and references within). Second, the algorithm identified two new regulators (Skn7 and Stb1) that are likely to be involved

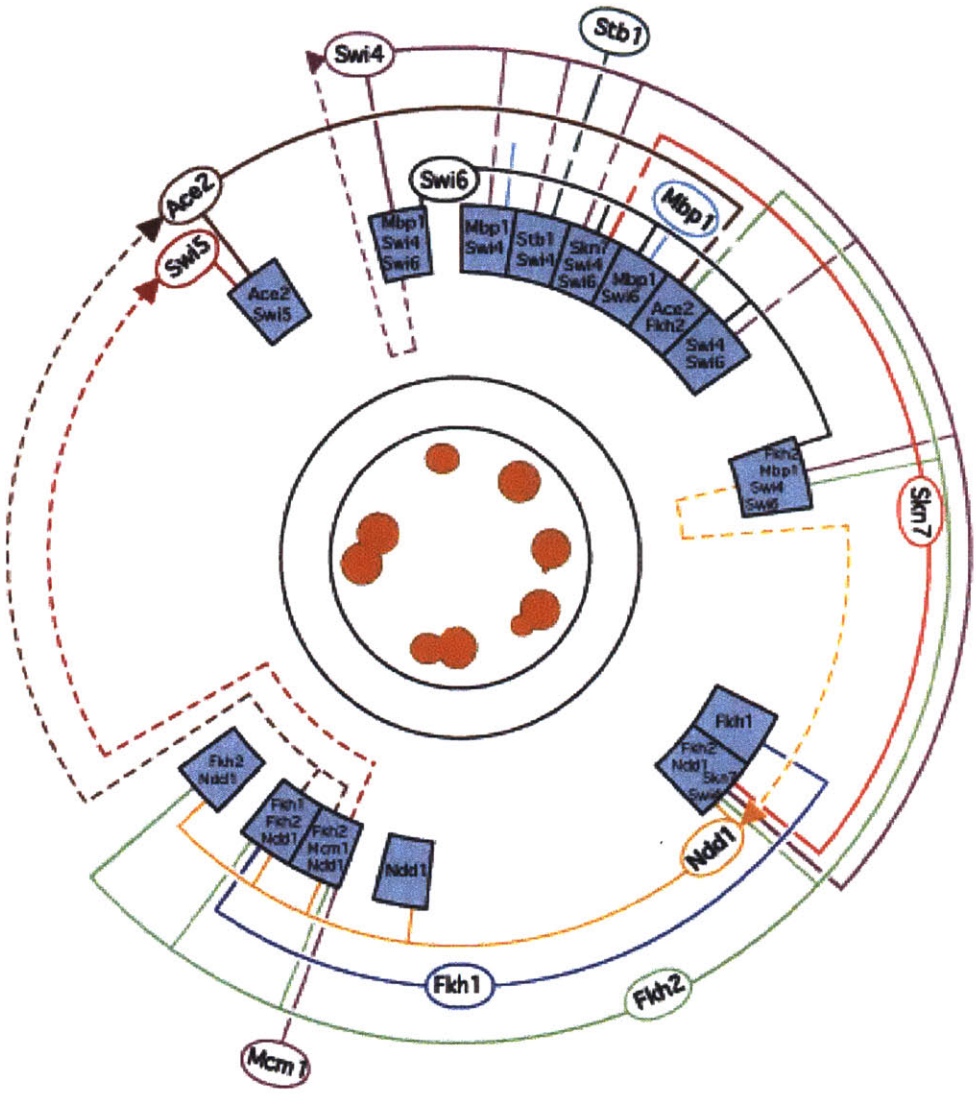


Figure 8-9: Cell cycle sub-network - In order to investigate the yeast cell cycle, we applied our sub-network discovery algorithm in combination with a continuous temporal alignment algorithm to uncover not only modules and their regulating transcription factors, but also the temporal relationships among these modules. The automatically recovered network is extremely similar to the one described in [92],which required considerable prior biological knowledge to construct. Modules are shown as ovals containing the names of regulating factors. Dashed lines pointing to factors indicate that these factors are contained in one of the modules. As can be seen, the cell cycle is controlled by factors regulating factors in a cycle.

in cell cycle control but not traditionally considered cell cycle regulators (see Bouquin *et al* [27]). Third, and most importantly, the reconstruction of the regulatory architecture is automatic and requires no prior knowledge of the regulators that control transcription during the cell cycle.

Our algorithm recovered many modules regulated by sets of transcription factors that are known to be associated or in complexes, such as Swi4/Swi6, Swi6/Mbp1, Swi5/Ace2, and Fkh1/Fkh2/Ndd1/Mcm1. Interestingly, the recovered network suggests that combinatorial factor interactions may provide control that allows for sub-dividing cell cycle phases into different biological functions. For instance, a module regulated by Mbp1/Swi4/Swi6 seems to encode genes involved with budding and can be localized at the M/G1 boundary. Another module regulated by Mbp1/Swi6 can be localized at almost the same time, but encodes a number of genes involved with DNA recombination and repair. A module in the middle of G1 can be localized that is regulated by Swi4/Swi6 and contains genes involved in cell-wall synthesis, and at the G1/S boundary a module can be localized that is regulated by Swi4/Fkh2/Ndd1 and contains many genes involved with histone synthesis.

Rapamycin network

Rapamycin is a compound that has been found to possess antimicrobial, immunosuppressive, and anticancer properties. The compounds mechanism of action has been shown to be highly conserved from yeast to humans [34]. It acts on the Tor proteins, which are critical in sensing and responding to cellular stress, and particularly in nutrient signaling [90]. In S. cerevisiae, rapamycin treatment causes the cell to enter a starvation-like state, in which it down-regulates numerous energy intensive processes, and acts as if it lacks quality nitrogen sources [54]. We selected 14 transcriptional factors that had been implicated in the above references as regulators of rapamycin response in S. cerevisiae, and performed genome-wide protein-DNA binding assays. In general, we found that the binding behavior of the 14 factors in rapamycin showed little overlap with their binding behavior in rich-media conditions obtained in previous experiments [73]. These results provide support for our view that profiling in alternate conditions is important for gaining a more detailed understanding of genetic regulation.

We ran the GRAM algorithm using genome-wide binding data for the 14 transcrip-

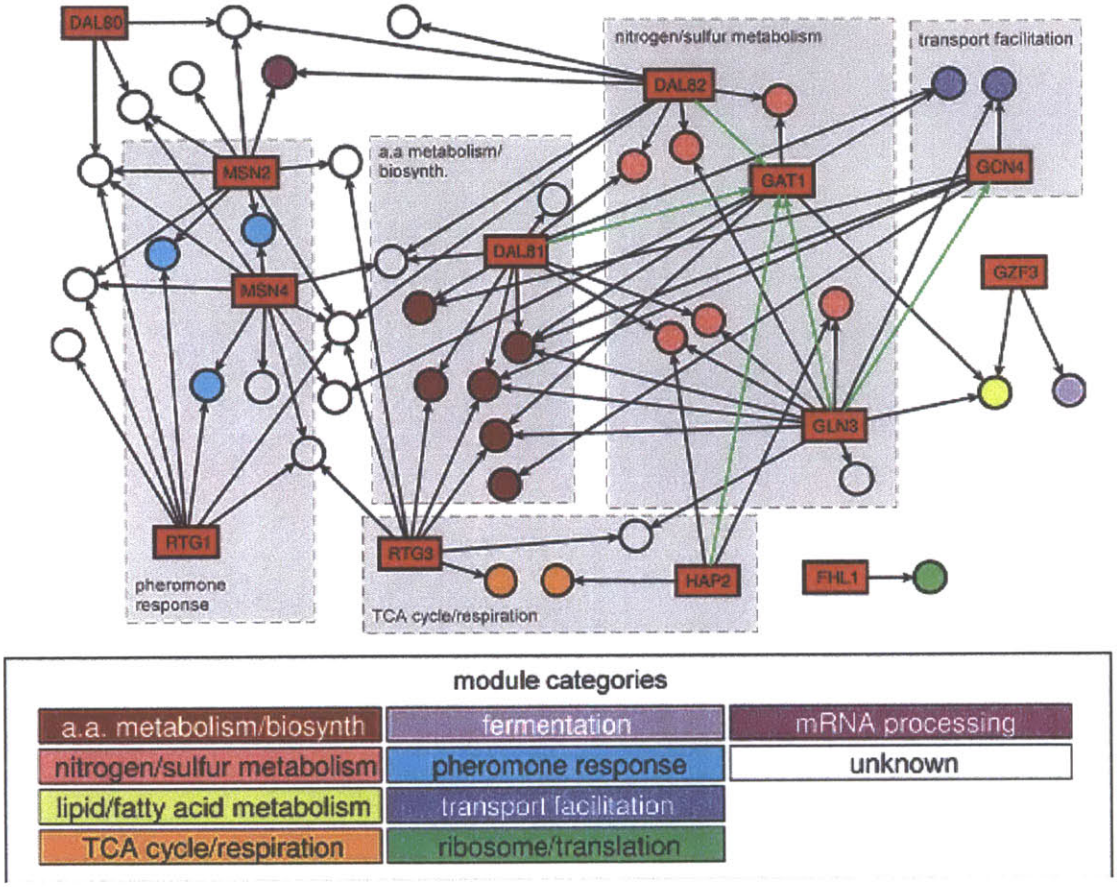


Figure 8-10: Rapamycin gene modules network - Analysis of the rapamycin regulatory network reveals several novel biological insights (see text for details). Thirty-nine modules containing 317 unique genes and regulated by 13 transcription factors were discovered. Red rectangles indicate transcription factors. Circles indicate modules. Black arrows point from transcription factors to the modules that they regulate. Green arrows between transcription factors indicate that the source transcription factor binds at least one module containing the target transcription factor. Modules are colored according to the MIPS category to which a significant number of genes belong (significance test using the hypergeometric distribution p ¡ 0.05).

tion factors in rapamycin and 22 previously published expression experiments relevant to rapamycin conditions [90, 54]. Thirty-nine modules containing 317 unique genes and regulated by 13 transcription factors were discovered (see Figure 8-10). Twenty-three of the modules were found to contain a significant number of genes (p-value < 0.05) belonging to a single MIPS category, with a total of 9 categories represented. The GRAM algorithm added 119 genes (38%) that were not bound with a strict p-value in the binding experiments, and many of the added genes could clearly be identified as belonging to the appropriate biological pathways. This result is similar to that obtained and validated for the rich-media regulatory network, suggesting that the GRAM algorithm can generally reduce the false-negative rate without significantly increasing the false-positive rate.

Analysis of the rapamycin regulatory network discovered by the GRAM algorithm reveals several biological insights. First, our results provide direct evidence for several regulatory relationships that had not previously been described in the literature. Our results indicate that Msn2/4 may be involved in repressing the pheromone response after rapamycin treatment, a role that these factors have not previously been implicated in. Msn2/4 are bound to five modules that contain a significant number of genes with down-regulated expression profiles (see [17] for the modules), and many of these modules contain genes involved in the pheromone response. An additional finding is that Dal82 and Msn2 regulate a module containing genes involved in mRNA processing. In general, the role of the TOR pathway/rapamycin in mRNA processing has not been explored, and Dal82 and Msn2 have not been previously implicated in regulating this function. Our results also indicate that Hap2 regulates two modules (along with Gln3 and/or Dal81) containing genes involved in nitrogen catabolism. While the major role of the Hap complex is thought to be in regulating genes involved in respiration, there was some previous evidence that the complex is involved in regulating ammonia utilization [36]. Our findings provide further support for this hypothesis and suggest coordinate activity with Gln3 and Dal81. It should be noted that many of the transcription factors profiled have been demonstrated to be regulated by cytoplasmic sequesterization [90, 34], so we did not expect to be able to identify activator/repressor relationships by searching for transcription factor expression profile correlations with regulated modules.

Analysis of the rapamycin network architecture reveals several instances of fairly complex interactions among modules. For example, several factors are apparently involved in

a feed-forward regulatory architecture. Dal81, Dal82, Gln3, and Hap2 all bind to modules containing Gat1, which has been previously identified as a general activator of nitrogen responsive genes [34, 90]. These modules containing Gat1 also contain many genes involved in nitrogen metabolism. Further, Gat1 itself binds a number of modules with Dal81, Dal82, and Gln3, and these modules contain many genes involved in nitrogen metabolism. This suggests a feed-forward regulation mechanism, in which Gat1 activity is amplified by the other mentioned transcription factors. Analysis of the network also reveals several instances of non-transcriptional regulatory interactions between modules. For example, Msn2 binds to a module containing Crm1, which is a nuclear export factor critical in allowing Gln3 to move from the cytoplasm to the nucleus after rapamycin treatment [90, 34]. This finding suggests that Msn2 activation upon rapamycin treatment may act to enhance or enable a step in Gln3 activation. As another example, Gcn4 binds to a module containing genes involved in amino acid biosynthesis, including Npr1, a serine/threonine protein kinase that is known to promote the function of the general permease Gap1 [90]. Gap1 itself is contained in a module regulated by Dal81 and Gln3. These findings suggest regulatory connections between these modules, in which Gap1 is transcriptionally regulated by Dal81/Gln3, Npr1 is transcriptionally regulated by Gcn4, and then Gap1 is non-transcriptionally activated by Npr1. It is possible that such regulatory relationships, which are clearly more complicated than simple activator/repressor mechanisms, may be especially important for facilitating a rapid and flexible response to environmental emergencies such as rapamycin treatment.

8.4 Discussion

Many microarray expression data sets have been analyzed using clustering methods such as k-means, or self-organizing maps [96]. While clustering can be used to classify genes and determine their function, it is less appropriate when it comes to modeling systems in the cell. In this section we have presented an algorithm that infers such models by abstracting expression regulation using modules. While a number of previous methods have used similar approaches in order to discover the different transcriptional modules that are activated in response to external conditions, our algorithm improves upon these methods in a number of ways.

As discussed above, a number of previous methods [81, 65] start with an initial set of

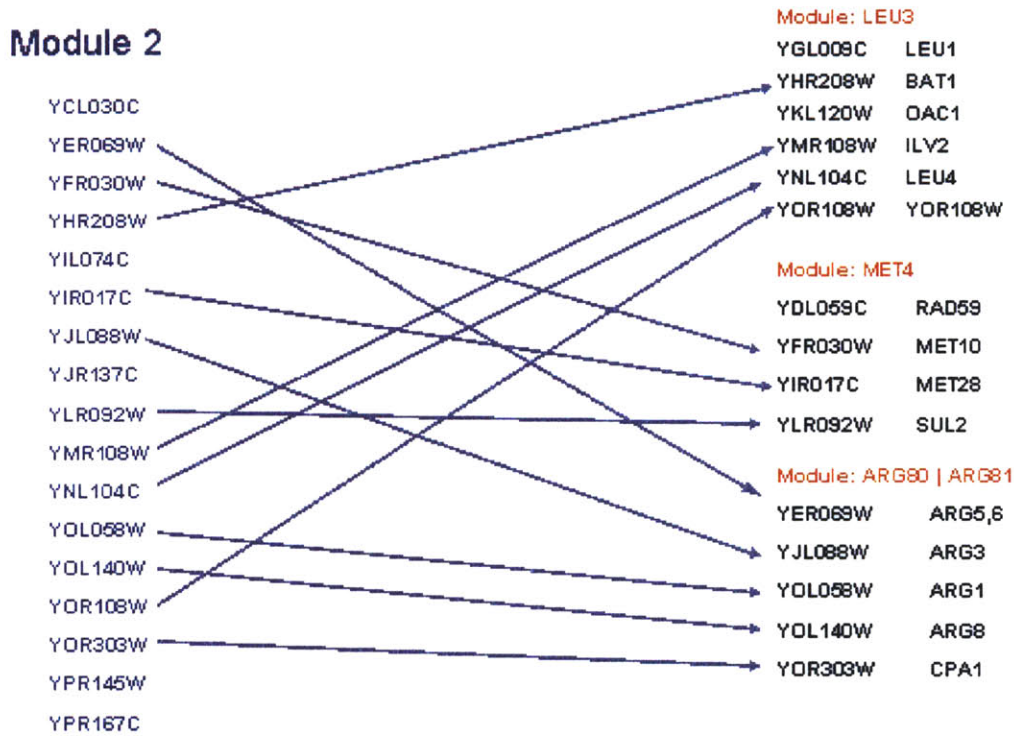


Figure 8-11: Comparison with the Ihmels *et al* [65] results. Left: A subset of the genes belonging to Module 2 of the Ihmels et al paper. Right, three of the amino acid biosynthesis modules identified in our paper. As can be seen, our method extends the results in the Ihmels paper by identifying not only genes that participate in a certain transcriptional module, but also provides evidence as to the pathway that is used to activate these genes. This allows us to answer more detailed/specific questions regarding the relationships between genes and the factors that control them.

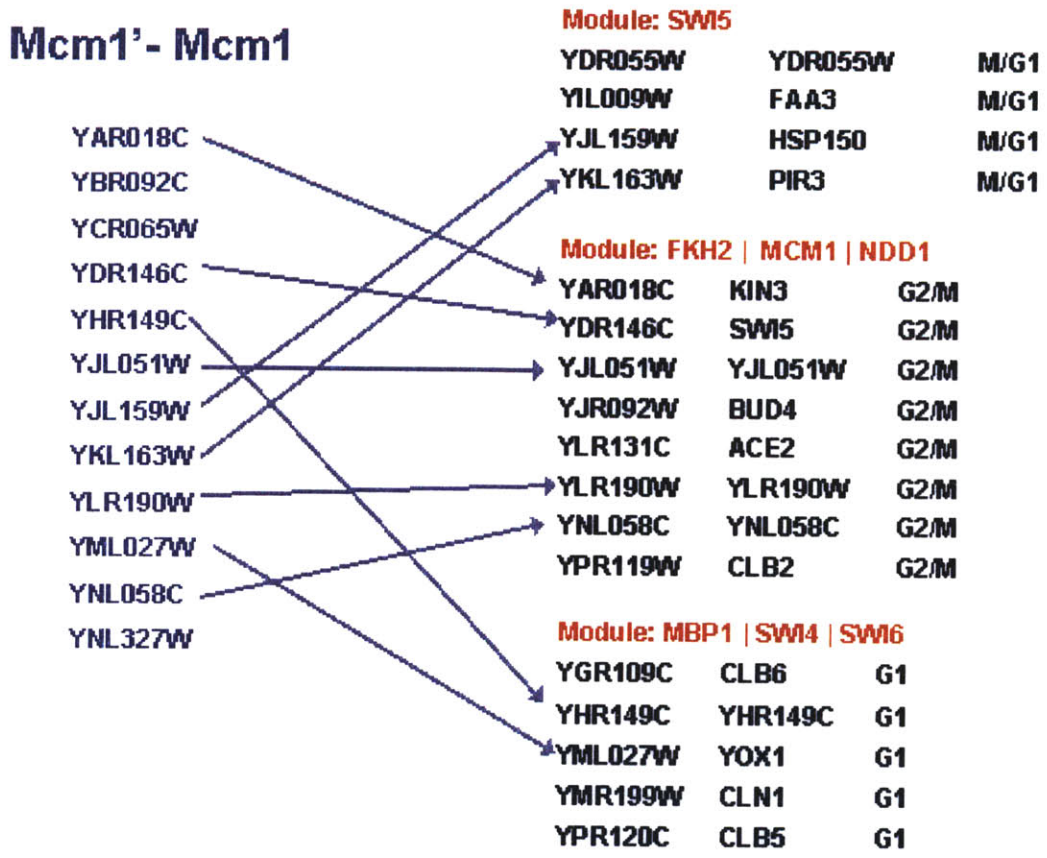


Figure 8-12: Comparison with the results of Pilpel *et al* [81]. Left: a subset of the genes that belong to the Mcm1-Mcm1 module from the Pilpel et al paper. Right: subsets of some of the cell cycle modules from our paper. As can be seen, the modules discovered by our method are a refined version of the module from the Pilpel paper. Note that our modules differ not only in the set of factors regulating the modules, but also in the different cell cycle phases to which they belong, providing a better understanding of how the cell regulates the complex expression program that is associated with the cell cycle system.

genes that are selected using a certain criteria (DNA binding motif, functional category) and use expression data to refine the initial set. While these methods represent an important first step, the method presented in this paper improves upon them in several ways, as discussed in Section 8.1. Figures 8-11 and 8-12 present a comparison between modules discovered by our algorithm and modules that were generated by previous work. As can be seen, using direct (binding) information allows us to accurately reconstruct not only the modules, but also the networks and pathways used by the cell to control gene expression programs under different conditions.

8.4.1 Limitations of the GRAM algorithm

While our module discovery approach produces many meaningful biological results, it is important to understand its inherent limitations. For instance, our approach identified only a relatively small number of transcription factors as activators or repressors. This may be due to several issues. First, our method cannot detect the many factors that are post-transcriptionally activated and thus have expression levels that are not expected to fluctuate significantly. Second, we used the entire set of expression experiments to determine the activator/repressor relationships. While using more data can produce more statistically significant results, it may be that only under certain conditions a factor serves as an activator or repressor. Finally, we required a very high correlation between modules and regulating factors. In general, by relaxing threshold parameters, the algorithm can be used in an exploratory mode to discover more relationships but with less confidence. For similar reasons, the sub-networks induced by our algorithm are necessarily incomplete. For instance, GCN4 is known to be a master regulator of the amino acid starvation response, so it seems odd at first that it shows up as a regulator of only one module in the sub-network. However, it is also known that GCN4 expression is repressed under non-starvation conditions. Hence, it is probable that little GCN4 binding is seen when yeast are grown on rich media. Note that this limitation is primarily a limitation on the data available. Indeed, as we have shown in the rapamycin sub-network, when using the right binding condition our algorithm is able to identify many more meaningful relationships when compared to those identified under rich media condition.

8.4.2 Summary

In this chapter we have presented GRAM, an algorithm for discovering groups of gene that are both similarly expressed and regulated by the same set of transcription factors. GRAM combines expression and binding data to discover these modules. We have applied GRAM to a number of biological datasets in order to determine how genes are regulated in the cell, and how various systems in the cell respond to external conditions. Our results suggest the power of using both binding data obtained under ideal growth conditions and expression data obtained under specific conditions to discriminate relationships of real biological relevance. In the future, genomic binding data obtained under a variety of conditions is likely to become available and should be of great value in further discovery of genetic regulatory networks.

Chapter 9

Conclusions and future work

In this thesis I have presented algorithms for inferring interactions between genes, identifying expression programs and discovering networks for different systems in the cell. These results are summarized in Section 9.1. While computational biology deals mainly with algorithms for various issues that arise when studying biological systems, a number of algorithms that were originally developed for computational biology problems are appropriate for other areas in computer science as well. As I discuss in Section 9.2 two of the algorithms presented in this thesis can be (and have been) applied to other problems in computer science. This is a strong indication that as we tackle exciting problems that arise from biology, we will benefits other areas of computer science. Finally, one of the nicest properties of computational biology is that there are more open problems than solved ones. In Section 9.3 I discuss a number of open problems that are left for future work.

9.1 Conclusions

Gene expression, and other high throughput biological datasets are revolutionizing molecular biology by providing a large scale overview to the activity of genes in the cell. While these techniques are promising, they require new computational tools for handling noisy datasets, pattern recognition and information fusion.

In this thesis I have presented algorithms for inferring interactions between genes, identifying expression programs and discovering networks for different systems in the cell. Since gene expression is a temporal process, in order to truly understand different systems in the cell we must study the temporal expression patterns of genes that participate in those

systems. As we have shown in different parts of this thesis, by developing algorithms that are specifically targeted to the analysis of time series expression data we can obtain a much clearer picture of the expression programs in living cells. We presented several algorithms that address various aspects of the analysis of time series expression and other high throughput biological datasets. Our algorithms overcome many problems that are unique to time series expression data, including non uniform sampling rates that differ between experiments (even those that study the same system under different conditions), difference in the timing of biological processes and lack of experimental repeats. Our algorithms can also determine temporal expression programs by determining relationships between different gene clusters. As for modeling systems, we presented algorithms that can combine static binding data and time series expression data to infer dynamic models for the cell cycle system.

While the above algorithms address computational challenges, it is important to note that they also improve our understanding of the underlying biology, as we demonstrated throughout this thesis. For example, our algorithms where able to identify many more correct targets in a knockout time series expression experiment, and suggested new roles for the two knocked out factors, Fkh1 and Fkh2, in controlling cellular activity. Our algorithms have also identified new factors that control the cell cycle system, and determined the phase in which they are involved. Several other networks can be determined by our algorithm, and as we showed using the rapamycin network, these models provide new insights as to how gene expression is regulated on the molecular level.

While this thesis presents a number of computational tools and algorithms for analyzing gene expression data, there is still a lot of work to be done in this area, as we discuss below. I believe that as expression analysis becomes a simpler and cheaper method, many of the datasets will be time series datasets, simply because the process we are interested in is temporal. It is my hope that the methods discussed in this thesis will serve the many biology researchers working in this area when they analyze, visualize and build models for the systems and process they are studying.

9.2 Relation to other problems in computer science

As a field, computational biology applies computational methods to problems in biology. Recent technology advances in biology (such as better sequencing machines and microarrays)

are generating large quantities of data. Making sense of these individual data sources, and combining them to create a better picture of the activity in the cell requires the improvement of existing, and the development of new, computational methods. While these methods are primarily aimed at solving problems arising in biology, the computational methods that are being developed can also be applied to other areas in computer science. In this thesis we have presented a number of such algorithms, that, while developed for clustering and ordering expression datasets, can also be applied to improve image compression (our optimal leaf ordering algorithm) or to improve the clustering of non uniformly sampled datasets (our continuous clustering algorithm from Chapter 5).

I believe that advances in computational biology will prove to be useful in a number of different areas in computer science. Below I list two additional research areas that are being pursued in computational biology, and that are likely to have an impact on a number of other areas in computer science.

9.2.1 Graphical models

Early on, Markov and hidden Markov models have been used to model families of proteins, and splice sites for genes [11]. More recently, various graphical models have been used to model the regulation of genes in the cell. For example, Friedman *et al* [47] used expression experiments performed on yeast cells to learn static Bayesian networks and subnetworks for the regulation of gene expression in yeast. Hartemink *et al* [56] used binding and expression data to learn such Bayesian networks. Segal *et al* used Probabilistic Relational Models (PRMs) [89] which extend the standard attribute-based Bayesian network representation to incorporate a relational structure, to identify networks of modules using gene expression data. Yeang and Jaakkola [105] used Markov random fields to model transcription regulation.

The above are only representative examples of methods that employ graphical models to model networks in the cell. Biological datasets present a number of challenges that should be dealt with when applying such methods to model systems in the cell. In most biological systems, there are a large number of genes involved, and many types of connectivity (for example, protein-DNA and protein-protein interactions). In addition, many of the observations (such as the protein levels themselves) are missing, and the ones that are observed are very noisy. This will require the development of faster inferences and learning methods that

can accommodate large fractions of missing data. Another type of graphical models that should be developed for computational biology are models that will allow us to combine time series and static data in a unified framework (see also Section 9.3.2). I believe that the methods and models developed to address these problems will prove useful in a number of other areas which deal with large quantities of noisy data.

9.2.2 Visualization and graphics

An important future area will be the visualization of large scale biological data sets. Many high-throughput data sources are becoming available, and visualizing the agreements and disagreements between these sources will be an important tool for analyzing these experiments and for suggesting new biological experiments. Some of my research has already dealt with these issues (for example, our k-ary clustering algorithm described in Chapter 7), however, new tools will be required in order to effectively visualize regulatory and other complex networks in the cell.

Biological networks are composed of a large number of interacting components. Further, there are many potential interaction modes (for example proteins can serves as activators or repressors, they can work together or alone and interact by binding DNA or directly binding other proteins). In addition, since we are using a large collection of high throughput datasets to discover these networks, each link can be derived from a number of different data sources. Developing methods which will allow us to graphically query such networks to obtain the information that was used to derive individual links and other proprieties of these networks, will be an important tool for researchers studying biological processes.

Methods that are developed for visualizing complex biological networks can be used to visualize and study other networks, such as the Internet routing network or the world wide web. It has been shown [23] that biological networks share common topological features with these other networks, and thus research into visualization issues in computational biology is likely to lead to more general solutions for visualizing complex networks.

9.3 Future work

While many issues related to the analysis of high throughput biological datasets have been addressed in this thesis, the algorithms presented here are by no means the only possible

solutions to the challenges they address. In addition to improving upon the results presented in this thesis, there are number of possible extensions that I am interested in, and I list some of them below.

9.3.1 Determining sampling rates for time series expression experiments

An important problem for time series expression experiments design is the determination of sampling rates. As shown in Table 1.1 under different experimental conditions the same system behaves faster or slower, depending on the experimental setup. Thus, it is hard to determine the correct sampling rate in advance. If the experiment is under-sampled, the results might not correctly represent the activity of the genes in the duration of the experiments, and key events can be missed. On the other hand, over-sampling is expansive and time consuming. Since many experiments are limited by budget constraints, over-sampling will result in shorter experiment duration, which might lead to missing important genes that participate in the process at a later stage.

A future direction would be to use the tools developed in this thesis for the task of determining the right sampling rates for these experiments. Due to the differences in the timing of biological processes (see Table 1.1), it is impossible to determine such rate in advance. Thus, we propose to use an online algorithm in the following way. The researcher will chose the highest possible sampling rate, and will sample at that rate. While the sampling rate will be high, the samples will not be hybridized (that is, the expression experiment will be performed on these samples) but rather frozen until we determine if we need to use them or not. In order to determine which of the samples should be hybridized, our algorithm will start by hybridizing very few (uniformly spaces) samples. Next, we use our interpolation algorithm to fit a continuous representation to each of the genes on the array. Using techniques from active learning we will determine what are the areas on the curves that we are least confident in, and the algorithm will select a new sample from that area to hybridize. This process will be repeated until we reach a certain confidence on the entire curve. When this happens we will terminate the experiment and use the set of hybridize samples (and their continuous representation) as the results of our experiment.

There is still a lot of research necessary in order to make the above method useful. We need to determine how to calculate confidence for different points on the interpolated curves, and what is the appropriate confidence level such that when we reach this level we

terminate the experiment. Still, we believe that such an approach can determine good and correct sampling rates for any experimental condition.

9.3.2 Combining static and time series expression data

While time series expression data is necessary for constructing dynamic models for systems in the cell, such experiments are more expensive and time consuming when compared to the static expression experiments. One of the most important types of expression experiments are knockout (or perturbation) experiments. These experiments allow us to determine the set of genes that are directly and indirectly affected by a a knockout of a specific gene. In addition, by determining the direction of the change (up or down regulated) we can infer the function of the knocked out gene or one of its immediate targets.

While knockout datasets are useful, they are also more expensive. In yeast there are almost 6000 genes. Performing a knockout time series expression experiment for each of these genes under a variety of experimental conditions is impractical at this stage. Further, in some cases a single knockout is not enough, and changes are only visible when a double knockout is carried out [107]. Even if we restrict ourselves to certain systems (and thus, we only need to knockout certain genes that are involved in this system), time series double knockout for all possible combinations are not practical.

Instead of performing time series knockout experiments for all these genes, we propose to perform a few time series experiments, and a large number of static knockout experiments (static experiments can be done at roughly 4% of the time and expense of time series experiments [94]). We then intend to develop an algorithm which will use the static experiments to determine the connectivity and parameters of the model, and the time series experiments to derive the dynamics of the model. We plan to use dynamic Bayesian networks for this task, and to constrain the parameters of the network using the static datasets. This will require us to develop a new probability model which will be able to constrain the dynamic structure based on static data.

9.3.3 From model organism to humans

While the algorithms presented in this paper are general, all the results presented were obtained using yeast expression and binding data. Yeast is one of the most studied model organisms, and it contains a large number of genes that have homologes in humans. Thus,

we expect that the algorithms discussed in this thesis will be useful to analyze higher organisms as well.

Still, there are number of issues that should be addressed more carefully when moving from yeast to humans. First, yeast can be grown in the lab. Thus, for yeast sample quantity is not an issue. However, when dealing with humans the amount of the material needed becomes a bottleneck for expression experiments. Another issue is the fact that humans have many more genes (up to six times more than yeast) and many more transcription factors [71, 100]. In addition, while yeast genes are controlled by an average of two or three TFs [73], it is believed that human genes are controlled by a much larger number of factors.

Thus, we need to extend our algorithms to address the above issues. More specifically, we intend to develop algorithms for analyzing short (up to 10 points) time series expression datasets to infer significant changes in sets of genes. While many profiles can be found in short datasets due to noise (especially in human samples, since there are tens of thousands of genes profiled), we will try to determine which of these profiles are significant and which are the result of noise. In addition, we intend to improve our networks algorithm so that they can handle much larger sets of TFs, and efficiently search these large spaces to determine modules for the regulation of gene expression in humans.

Appendix A

Appendix

In this appendix we provide further details about the algorithm and results for identifying differentially expressed genes which were discussed in Section 4. In Section A.1 we present a symmetric version for this algorithm. In Section A.2 we present the details of the method we have used to identify the expression experiments in which the non cycling genes identified by our algorithm are significantly co-expressed.

A.1 The symmetric version of our algorithm

So far we have assumed a fixed referenced curve. That is, when we compute the area between the reference and test curves we do not consider the fact that the reference curve might be a noisy realization of the true underlying curve. Under the null hypothesis we assume that both datasets where generated from the same underlying profile, and thus a symmetric version of our algorithm seems more appropriate for this hypothesis. As we show in this section, the algorithm presented in the methods section can also be described as a symmetric test on both the reference and test curve (with appropriately scaled p-values), and thus our algorithm is suitable for the null hypothesis as well.

Instead of relying on the reference curve, we assume that both C_1 and C_2 are realizations of the same underlying curve C. We reformulate the comparison in terms of joint probabilities over the two curves (making the comparison symmetric) but rely on the distance between the curves rather than on the points directly. As in the methods section, Let $e^2 = D(C_1, C_2)$. For the null hypothesis we solve the following maximization problem:

$$max_{C,C',C''} P(Y_{C'}, Y_{C''}|Y_C, \sigma^2) s.t. D(C', C'') = e^2$$

where C' and C'' are new (arbitrary) versions of the reference and test curves and C denotes the common underlying true curve. Due to the Gaussian noise we assume for individual measurements, the maximization over C yields $Y_C = (Y_{C'} + Y_{C''})/2$ and therefore, ignoring constant terms we have:

$$\log P(Y_{C'}, Y_{C''}|Y_C, \sigma^2) = -(1/2\sigma^2)(||Y_{C'} - Y_C||^2 + ||Y_{C''} - Y_C||^2) = -(1/4\sigma^2)||Y_{C'} - Y_{C''}||^2$$

If we now set $Y_\delta = Y_{C'} - Y_{C''} = SF' - SF'' = S\delta$, then we end up solving the same optimization problem as before:

$$min_\delta (S\delta)^T (S\delta) \quad s.t. \quad \delta^T A \delta = 1$$

The only difference between the result obtained by this method (symmetric), and the result discussed in the methods section (asymmetric) is that the value of the likelihood ratio test using the symmetric test is half the value obtained using the asymmetric method. Thus, for every p-value cutoff used by the asymmetric method, there is a corresponding p-value for the symmetric method which yields the same results (i.e. the same set of genes are determined to be significantly changing). Since our p-value is tuned using synthetic data, changing from the asymmetric to the symmetric method would not have changed the results presented in this paper. Since the asymmetric method is somewhat easier to explain, we have focused on it in the methods part of our paper.

A.2 Non cycling genes

As mentioned in Section 4.4.2 our algorithm identified 22 of the non cycling genes as differentially expressed (p-value < 0.001). In order to determine the role Fkh1/2 play in controlling this set, and to test whether this set is biologically significant, we have used large collection of gene expression experiments (see supplementary website [16]). As was done by Hughes *et al* [63], we have looked for experiments in which these genes was significantly correlated,

in the following way. Denote the set of genes selected by our algorithm by S, and let G represent the entire set of genes profiled. For experiment i, let G_u^i be the set of genes in G with a log fold change higher than 1 (or a 2-fold change), and let S_u^i represent the same for the genes in S. Let G_d^i be the set of genes in G with a log fold change less than -1, and let S_d^i represent the same for the set of genes in S. Using G, S, G_u^i and S_u^i, we can perform a hyper-geometric test to compute the significance (or p-value) for the up-regulation of the genes in S in experiment i. The same could be done for down regulation by replacing G_u^i and S_u^i with G_d^i and S_d^i. Using these p-values we include experiment i in the selected set if:

- The p-value for either up-regulation (S_u^i) or down-regulation (S_d^i) is below 0.001 and
- At least 25% of the genes in S are up or down regulated (depending on the direction used for the p-value).

Based on these criteria, 20 expression experiments were selected (see Table A.1). As can be seen, these experiments come from six different datasets, indicating that our result are not an artifact of a specific hybridization method. Most of these experiments are related to stress response. These results suggest a new function for Fkh1 and Fkh2 which is related to controlling yeast response to stress.

In order to test the significance of these findings, we performed the following randomization test. We have selected at random sets of 22 genes, and for each such set looked at how many expression experiments satisfy the criteria presented above (p-value < 0.001 and at least 25% of the genes in the same direction). We have repeated this process 1000 times. For all of the random sets, we did not find *even one* experiment that was significantly correlated (according to the above criteria) with this set. We conclude that the genes identified by our algorithm represent a biologically significant set of genes that are involved in yeast response to stress.

Significant experiments for non cycling genes

Change	Reference	Experiment type	Experiment details
up-regulated	Gasch [49]	red/ox	105min 1mM menadione / 105min wt
up-regulated	Gasch [49]	red/ox	180min 2.5mM DTT / 180min DTT pool
up-regulated	Gasch [49]	red/ox	120min 2.5mM DTT / 120min DTT pool
up-regulated	Gasch [49]	red/ox	40 min 1.5mM diamide / 40 min wt
up-regulated	Gasch [49]	red/ox	50 min 1.5mM diamide / 50 min wt
down-regulated	Gasch [49]	red/ox	15min 2.5mM DTT/ 15min DTT pool
up-regulated	Gasch [49]	starvation	6hr YNB-AA / 6hr wt
up-regulated	Gasch [49]	starvation	10h YPD 30C / 10h wt
up-regulated	Gasch [49]	starvation	1d YPD 25C / 1d wt
up-regulated	Gasch [49]	starvation	2% EtOH / EtOH Carbon pool
down-regulated	Natarajan [76]	starvation	δgcn4 100mM 3AT / δgcn4 100mM 3AT δgcn4
up-regulated	Jelinsky [68]	α factor	G1 arrest - 0.3uM α factor 120 min α factor/wt
up-regulated	Roberts [86]	α factor, pheromone	90 min α factor, 50nM / 90min wt
up-regulated	Roberts [86]	α factor, pheromone	120 min α factor, 50n / 120min wt
down-regulated	Lyons [75]	zinc	replete vs. deficient zinc / 3mM 61nM
down-regulated	Lyons [75]	zinc	replete vs. deficient zinc / 3mM 76nm
down-regulated	Lyons [75]	zinc	δzap1 excess vs. deficient zinc/ δzap1 3mM δ zap1 61nM
down-regulated	Lyons [75]	zinc	δzap1 excess vs. deficient zinc, δzap1 3mM δzap1 76nM
up-regulated	Causton [30]	others	Alkali 80'/0'
down-regulated	Jelinsky [68]	others	30 min 0.1% MMS / 30 min wt

Table A.1: The 20 expression experiments in which the set of non cycling genes identified by our algorithm were significantly correlated. See text for discussion.

Bibliography

[1] *Affymetrix.* URL: http://www.affymetrix.com/index.affx.

[2] *Saccharomyces cerevisiae - MIPS categories.* URL: http://www.mips.biochem.mpg.de.

[3] J. Aach and G. M. Church. Aligning gene expression time series with time warping algorithms. *Bioinformatics*, 17:495–508, 2001.

[4] M.D. Adams, S.E. Celniker, R.A. Holt, and et al. The genome sequence of drosophila melanogaster. *Science*, 287:2185–95, 2000.

[5] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Molecular Biology of the Cell.* Garland Pub, 2002.

[6] U. Alon, N. Barkai, D.A. Notterman, K. Gish, S. Ybarra, D. Mack, and A.J. Levine. Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. *PNAS*, 96:6745–6750, 1999.

[7] O. Alter, P.O. Brown, and D. Botstein. Generalized singular value decomposition for comparative analysis of genome-scale expression data sets of two different organisms. *PNAS*, 100(6):3351 6, 2003.

[8] M.N. Arbeitman, E.E. Furlong, F.J. Imam, E. Johnson, B.H. Null, B.S. Baker, M.A. Krasnow, M.P. Scott, R.W. Davis, and K.P. White. Gene expression during the life cycle of drosophila melanogaster. *Science*, 298:2270–75, 2002.

[9] M. Badoiu and K. Clarkson. Smaller core-sets for balls. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2000.

[10] M. Badoiu and K. Clarkson. Smaller core-sets for balls. In *14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'03)*, 2003.

[11] P. Baldi. *Bioinformatics: the Machine Learning Approach.* MIT Press, 2001.

[12] P. Baldi and G.W. Hatfield. *DNA Microarrays and Gene Expression.* Cambridge, 2002.

[13] Z. Bar-Joseph and D. Cohen-Or. Hierarchical context-based pixel ordering. *Computer Graphics Forum (Eurographics 03)*, 2003.

[14] Z. Bar-Joseph, E. Demaine, D. Gifford, A. Hamel, N. Srebro, and T. Jaakkola. k-ary clustering with optimal leaf ordering for gene expression data. *Bioinformatics*, 19:1070–78, 2003.

[15] Z. Bar-Joseph, R. El-Yaniv, D. Lischinski, , and M. Werman. Texture mixing and texture movie synthesis using statistical learning. *IEEE Transactions on Visualization and Computer Graphics*, 7(2):120–135, 2001.

[16] Z. Bar-Joseph, G. Gerber, and *et al.* URL: http://www.psrg.lcs.mit.edu/DiffExp/DiffExp.html.

[17] Z. Bar-Joseph, G. Gerber, and *et al.* URL: http://www.psrg.lcs.mit.edu/~zivbj/modules/modules.html.

[18] Z. Bar-Joseph, G. Gerber, T.S. Jaakkola, D.K. Gifford, and I. Simon. A new approach to analyzing gene expression time series data. In *RECOMB*, 2002.

[19] Z. Bar-Joseph, G. Gerber, T.S. Jaakkola, D.K. Gifford, and I. Simon. Comparing the continuous representation of time series expression profiles to identify differentially expressed genes. *Proc. Natl. Acad. Sci. USA (PNAS)*, to appear, 2003.

[20] Z. Bar-Joseph, G. Gerber, T.S. Jaakkola, D.K. Gifford, and I. Simon. Continuous representations of time series gene expression data. *Journal of Computational Biology*, 3-4:341–356, 2003.

[21] Z. Bar-Joseph, G.K. Gerber, T.I. Lee, N.J. Rinaldi, J. Yoo, F. Robert, D.B. Gordon, E. Fraenkel, T.S. Jaakkola, R.A. Young, and D.K. Gifford. Computational discovery of gene modules and regulatory networks. *submitted*, 2003.

[22] Z. Bar-Joseph, D. Gifford, and T. Jaakkola. Fast optimal leaf ordering for hierarchical clustering. *Bioinformatics (Special Issue, Proceedings of ISMB)*, 17:s22 s29, 2001.

[23] A.L. Barabasi. *Linked: The New Science of Networks.* Perseus Publishing, 2002.

[24] R. Bartels, J. Beatty, and B. Barsky. *Splines for Use in Computer Graphics and Geometric Modeling.* Morgan Kaufman, 1987.

[25] A. Ben-Dor, R. Karp, B. Schwikowski, and Z. Yakhini. Universal dna tag systems: A combinatorial design scheme. In *RECOMB*, 2000.

[26] A. Ben-Dor, Shamir R., and Yakhini Z. Clustering gene expression patterns. *Journal of Computational Biology*, 6:281–297, 1999.

[27] N. Bouquin, A.L. Johnson, B.A. Morgan, and L.H. Johnston. Association of the cell cycle transcription factor mbp1 with the skn7 response regulator in budding yeast. *Mol Biol Cell*, 10(10):3389–400, 1999.

[28] B. Brumback and J. Rice. Smoothing spline models for the analysis of nested and crossed samples of curves. *Am. Statist. Assoc.*, 93:961–976, 1998.

[29] R. E. Burkard, Deineko V. G., and G. J. Woeginger. The travelling salesman and the pq-tree. *Mathematics of Operations Research*, 24:262–272, 1999.

[30] H.C. Causton, B. Ren, S.S. Koh, C.T. Harbison, E. Kanin, E.G. Jennings, T. Lee, H.L. True, E.S. Lander, and R.A. Young. Remodeling of yeast genome expression in response to environmental changes. *Mol. Biol. Cell*, 12:323–337, 2001.

[31] R.J. Cho, M.J. Campbell, E.A. Winzeler, L. Steinmetz, A. Conway, L. Wodicka, T.G. Wolfsberg, A.E. Gabrielian, D. Landsman, D.J. Lockhart, and R.W. Davis. A genome-wide transcriptional analysis of the mitotic cell cycle. *Mol. Cell.*, 2(1):65–73, 1998.

[32] S. Chu, J. DeRisi, and et al. The transcriptional program of sporulation in budding yeast. *Science*, 282:699–705, 1998.

[33] F.S. Collins, E.D. Green, Guttmacher A.E., and M.S. Guyer. A vision for the future of genomics research. *Nature*, 422:835–847, 2003.

[34] J.L. Crespo, T. Powers, B. Fowler, and M.N. Hall. The tor-controlled transcription activators gln3, rtg1, and rtg3 are regulated in response to intracellular levels of glutamine. *PNAS*, 99(10):6784–9, 2002.

[35] R. Dafner, D. Cohen-Or, and Y. Matias. Context based space filling curves. *Computer Graphics Forum*, 19:209–218, 2000.

[36] V.D. Dang, C. Bohn, M. Bolotin-Fukuhara, and B. Daignan-Fornier. The ccaat box-binding factor stimulates ammonium assimilation in saccharomyces cerevisiae, defining a new cross-pathway regulation between nitrogen and carbon metabolisms. *J Bacteriol*, 178(7):1842–9, 1996.

[37] R. Degerman. Ordered binary trees constructed through an application of kendall's tau. *Psychometrica*, 47:523–527, 1982.

[38] L. Deng, M. Aksmanovic, D. X. Sun, and C. F. J. X. Wu. Recognition using hidden markov models with polynomial regression functions as nonstationary states. *IEEE Transactions on Speech and Audio Processing*, 2:507–520, 1994.

[39] P. D'haeseleer, X. Wen, S. Fuhrman, and R. Somogyi. Linear modeling of mrna expression levels during cns development and injury. In *PSB99*, 1999.

[40] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer, New-York, NY, 1999.

[41] S. Dudiot and et al. Statistical methods for identifying differentially expressed genes in replicated cdna microarray experiments. *Statistica sinica*, in press.

[42] D.J. Duggan, M. Bittner, Y. Chen, P. Meltzer, and J.M. Trent. Expression profiling using cdna microarrays. *Nature Gen.*, 21(1 Suppl):10–14, 1999.

[43] M.B. Eisen, P.T. Spellman, P.O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *PNAS*, 95:14863–14868, 1998.

[44] D. Eppstein. Fast hierarchical clustering and other applications of dynamic closest pairs. In *Proc. of the 9th ACM-SIAM Symp. on Discrete Algorithms*, pages 619–628, 1998.

[45] R. Eubank. *Nonparametric regression and spline smoothing.* Marcel Dekker, 1999.

[46] R. A. Fisher. *Statistical methods for research workers.* Oliver and Boyd, 1970.

[47] N. Friedman, M. Linial, I. Nachman, and D. Pe'er. Using bayesian network to analyze expression data. *Journal of Computational Biology*, 7:601–620, 2000.

[48] N. Gale, W. C. Halperin, and C.M. Costanzo. Unclassed matrix shading and optimal ordering in hierarchical cluster analysis. *Journal of Classification*, 1:75–92, 1984.

[49] A.P. Gasch, P.T. Spellman, C.M. Kao, O. Carmel-Harel, M.B. Eisen, G. Storz, D. Botstein, and P.O. Brown. Genomic expression programs in the response of yeast cells to environmental changes. *Mol. Biol. Cell*, 11(12):4241–4257, 2000.

[50] K.G. Gerber. Do the time wrap: Continuous alignment of gene expression time series data. Master's thesis, MIT LCS, 2002.

[51] A. Goffeau, B.G. Barrell, and et al. Life with 6000 genes. *Science*, 274:563–67, 1996.

[52] T.R. Golub, D.K. Slonim, P. Tamayo, C. Huard, M. Gassenbeek, J. Mesirov, H. Coller, M. Loh, J.R. Downing, and et al. Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *Science*, 286:531–537, 1999.

[53] G. Gruvaeus and H. Wainer. Two additions to hierarchical cluster analysis. *British Journal of Mathematical and Statistical Psychology*, 25:200–206, 1972.

[54] J. S. Hardwick, F. J. Kuruvilla, J. K. Tong, A. F. Shamji, and Schreiber S. L. Rapamycin-modulated transcription defines the subset of nutrient-sensitive signaling pathways directly controlled by the tor proteins. *Proc. Natl. Acad. Sci. USA*, 96(26):14866–70, 1999.

[55] A. J. Hartemink, D. K. Gifford, T. S. Jaakkola, and R.A. Young. Using graphical models and genomic expression data to statistically validate models of genetic regulatory networks. In *PSB01*, 2001.

[56] A. J. Hartemink, D. K. Gifford, T. S. Jaakkola, and R.A. Young. Combining location and expression data for principled discovery of genetic regulatory network models. In *PSB02*, 2002.

[57] D. Hilbert. Mathematical problems. *Bull. Am. Math. Soc.*, 437-479:8, 1902.

[58] P.C. Hollenhorst, M.E. Bose, M.R. Mielke, U. Muller, and C.A. Fox. Fork-head genes in transcriptional silencing, cell morphology and the cell cycle. overlapping and distinct functions for fkh1 and fkh2 in saccharomyces cerevisiae. *Genetics*, 154:1533–1548, 2000.

[59] N. S. Holter, A. Maritan, M. Cieplak, N.V. Fedoroff, and J.R. Banavar. Dynamic modeling of gene expression data. *PNAS*, 98:1693–1698, 2001.

[60] N.S. Holter, M. Mitra, A. Maritan, M. Cieplak, J.R. Banavar, and N.V. FedoroffNeal. Fundamental patterns underlying gene expression profiles: Simplicity from complexity. *PNAS*, 97:8409–8414, 2000.

[61] L. Hood and D. Galas. The digital code of dna. *Nature*, 7:601–The digital code of DNA620, 2000.

[62] Q. Huang, D. Liu, P. Majewski, L.C. Schulte, J.M. Korn, R.A. Young, E.S. Lander, and N. Hacohen. The plasticity of dendritic cell responses to pathogens and their components. *Science*, 294:870–75, 2001.

[63] T.R. Hughes, M.J. Marton, A.R. Jones, C.J. Roberts, R. Stoughton, C.D. Armour, H.A. Bennett, E. Coffey, H. Dai, and et al. Functional discovery via a compendium of expression profiles. *Cell*, 102:109–126, 2000.

[64] T. Ideker, O. Ozier, B. Schwikowski, and A.F. Siegel. Discovering regulatory and signalling circuits in molecular interaction networks. *Bioinformatics*, Suppl 1:S233–40, 2002.

[65] J. Ihmels, G. Friedlander, S. Bergmann, O. Sarig, Y. Ziv, and N. Barkai. Revealing modular organization in the yeast transcriptional network. *Nature Genetics*, 31(4):370–377, 2002.

[66] N.B. Ivanova, J.T. Dimos, C. Schaniel, J.A. Hackney, K.A. Moore, and I.R. Lemischka. A stem cell molecular signature. *Science*, 298:601–604, 2002.

[67] G. James and T. Hastie. Functional linear discriminant analysis for irregularly sampled curves. *Journal of the Royal Statistical Society*, to appear, 2001.

[68] S. Jelinsky, P. Estep, G. Church, and L. D. Samson. Regulatory networks revealed by transcriptional profiling of damaged saccharomyces cerevisiae cells: Rpn4 links base excision repair with proteasomes. *Mol. and Cell. Bio.*, 20(21):8157–67, 2000.

[69] S.K. Kim, J. Lund, M. Kiraly, K. Duke, M. Jiang, J.M. Stuart, A. Eizinger, B.N. Wylie, and G.S. Davidson. A gene expression map for c. elegans. *Science*, 293:2087–92, 2001.

[70] M. Koranda, A. Schleiffer, L. Endler, and G.A. Ammerer. Forkhead-like transcription factors recruit ndd1 to the chromatin of g2/m-specific promoters. *Nature*, 406(6791):94–98, 2000.

[71] E.S. Lander, L.M. Linton, B. Birren, and et al. Initial sequencing and analysis of the human genome. *Nature*, 409:860–921, 2001.

[72] J. et al Lee. Yap1 and skn7 control two specialized oxidative stress response regulons in yeast. *J Biol Chem*, 274(23):16040–16046, 1999.

[73] T.I. Lee, N.J. Rinaldi, F. Robert, D.T. Odom, Z. Bar-Joseph, G.K. Gerber, N.M. Hannett, C.R. Harbison, C.M. Thompson, Simon I., and et al. Transcriptional regulatory networks in *saccharomyces cerevisiae*. *Science*, 798:799–804, 2002.

[74] A. Lempel and J. Ziv. Compression of two-dimensional data. *IEEE Transactions on Information Theory*, 32:2–8, 1986.

[75] T.J. Lyons, A.P. Gasch, L.A. Gaither, D. Botstein, P.O. Brown, and D. Eide. Genome-wide characterization of the zap 1 p zinc-responsive regulon in yeast. *PNAS*, 97:7957–7962, 2000.

[76] K. Natarajan, M.R. Meyer, B.M. Jackson, D. Slade, C. Roberts, Hinnebusch A.G., and M.J. Marton. Transcriptional profiling shows that gcn4p is a master regulator of gene expression during amino acid starvation in yeast. *Mol. and Cell. Bio.*, 21:4347–4368, 2001.

[77] K. Natarajan, M.R. Meyer, B.M. Jackson, D. Slade, C. Roberts, A.G. Hinnebusch, and M.J. Marton. Transcriptional profiling shows that gcn4p is a master regulator of gene expression during amino acid starvation in yeast. *Mol. Cell Biol.*, 21(13):4347–68, 2001.

[78] G.J. Nau, J.F.L. Richmond, A. Schlesinger, E.G. Jennings, E.S. Lander, , and R.A. Young. Human macrophage activation programs induced by bacterial pathogens. *PNAS*, 99:1503–1508, 2002.

[79] P.E. Neiman and et al. Analysis of gene expression during myc oncogene-induced lymphomagenesis in the bursa of fabricius. *PNAS*, 98:6378–6383, 2001.

[80] S. Panda, M.P. Antoch, B.H. Miller, A.I. Su, A.B. Schook, M. Straume, P.G. Schultz, S.A. Kay, J.S. Takahashi, and J.B. Hogenesch. Coordinated transcription of key pathways in the mouse by the circadian clock. *Cell*, 109(3):307–20, 2002.

[81] Y. Pilpel, P. Sudarsanam, and G.M. Church. Identifying regulatory networks by combinatorial analysis of promoter elements. *Nature Genetics*, 29:153–159, 2001.

[82] T. Pramilla, S. Miles, D GuhaThakurta, D. Jemiolo, and L.L. Breeden. Conserved homeodomain proteins interact with mads box protein mcm1 to restrict ecb-dependent transcription to the m/g1 phase of the cell cycle. *Genes Dev.*, 16(32):3034–3045, 2002.

[83] Sharan R. and Shamir R. Click: A clustering algorithm for gene expression analysis. In *ISMB00*, 2000.

[84] Sharan R. and Shamir R. Algorithmic approaches to clustering gene expression data. *Current Topics in Computational Biology*, To appear.

[85] B. Ren, F. Robert, J.J. Wyrick, O. Aparicio, E.G. Jennings, I. Simon, J. Zeitlinger, J. Schreiber, N. Hannett, E. Kanin, T.L. Volkert, C.J. Wilson, S.P. Bell, and R.A Young. Genome-wide location and function of dna binding proteins. *Science*, 290:2306–9, 2000.

[86] C.J. Roberts, B. Nelson, M.J. Marton, R. Stoughton, M.R. Meyer, H.A. Bennett, Y.D. He, H. Dai, W.L. Walker, T.R. Hughes, Tyers, and *et al.* Signaling and circuitry of multiple mapk pathways revealed by a matrix of global gene expression profiles. *Science*, 287:873–880, 2000.

[87] D. Rogers and J. Adams. *Mathematical Elements for Computer Graphics.* McGraw-Hill, 1990.

[88] E. Segal and D. Koller. Probabilistic hierarchical clustering for biological data. In *Recomb02*, 2002.

[89] E. Segal, M. Shapira, A. Regev, D. Pe'er, D. Botstein, D. Koller, and N. Friedman. Module networks: identifying regulatory modules and their condition-specific regulators from gene expression data. *Nature Genetics*, 34(2):166–76, 2003.

[90] A.F. Shamji, F.G. Kuruvilla, and S.L. Schreiber. Partitioning the transcriptional program induced by rapamycin among the effectors of the tor proteins. *Curr Biol*, 10(24):1574–81, 2000.

[91] D. D. Shoemaker and P. S. Linsley. Recent developments in dna microarrays. *Curr Opin Microbiol.*, 5(3):334–337, 2002.

[92] I. Simon, J. Barnett, N. Hannett, C.T. Harbison, N.J. Rinaldi, T.L. Volkert, J.J. Wyrick, J. Zeitlinger, D.K. Gifford, T.S. Jaakkola, and R.A. Young. Serial regulation of transcriptional regulators in the yeast cell cycle. *Cell*, 106:697–708, 2001.

[93] S.T. Smale. Core promoters: active contributors to combinatorial gene regulation. *Genes Dev.*, 15:2503–2508, 2001.

[94] P. T. Spellman, G. Sherlock, M.Q. Zhang, V.R. Iyer, K. Anders, M.B. Eisen, P.O. Brown, D. Botstein, and B. Futcher. Comprehensive identification of cell cycle-regulated genes of the yeast *saccharomyces cerevisia* by microarray hybridization. *Mol. Biol. of the Cell*, 9:3273–3297, 1998.

[95] K.F. Storch, O. Lipan, I. Leykin, N. Viswanathan, F.C. Davis, W.H. Wong, and C.J. Weitz. Extensive and divergent circadian gene expression in liver and heart. *Nature*, 418:78–83, 2002.

[96] P. Tamayo, D. Slonim, J. Mesirov, Q. Zhu, S. Kitareewan, E. Dmitrovsky, E.S. Lander, and T.R. Golub. Interpreting patterns of gene expression with self organizing maps: Methods and applications to hematopoietic differentiation. *PNAS*, 96:2907–2912, 1999.

[97] A. H. Y. et al Tong. A combined experimental and computational strategy to define protein interaction networks for peptide recognition modules. *Science*, 295:321–324, 2001.

[98] O. Troyanskaya, M. Cantor, and et al. Missing value estimation methods for dna microarrays. *Bioinformatics*, 17:520–525, 2001.

[99] O.G. Troyanskaya, M.E. Garber, P.O. Brown, D. Botstein, and R.B. Altman. Non-parametric methods for identifying differentially expressed genes in microarray data. *Bioinformatics*, 18(11):1454–1461, 2002.

[100] J. C. Venter, M. D. Adams, Myers E. W., and et al. The sequence of the human genome. *Science*, 291:1304–51, 2001.

[101] P. Viola. *Alignment by Maximization of Mutual Information.* PhD thesis, MIT AI Lab, 1995.

[102] J. D. Watson and F. H. C. Crick. Molecular structure of nucleic acids: A structure for deoxyribose nucleic acid. *Nature*, 171:737, 1953.

[103] A. Watt and M. Watt. *Advanced Animation and Rendering Techniques.* Addison-Wesley, 1992.

[104] X.L. Xu, J.M. Olson, and L.P. Zhao. A regression-based method to identify differentially expressed genes in time course studies and its application to inducible huntington's disease. *Hum. Mol. Genet.*, 11:1977–1985, 2002.

[105] C.H. Yeang and T. Jaakkola. Physical network models and multi-source data integration. In *RECOMB*, 2003.

[106] L. P. Zhao, R. Prentice, and L. Breeden. Statistical modeling of large microarray data sets to identify stimulus-response profiles. *PNAS*, 98:5631–5636, 2001.

[107] G. Zhu, Spellman P. T., T. Volpe, P.O. Brown, D. Botstein, T.N. Davis, and B. Futcher. Two yeast forkhead genes regulate cell cycle and pseudohyphal growth. *Nature*, 406:90–94, 2000.

[108] A. Zien, J. Fluck, R. Zimmer, and T. Lengauer. Microarrays: How many do you need? In *RECOMB*, 2002.