Clustering Genes and Inferring Gene Regulatory Networks using Multiple Information Sources

A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Master's Degree

by

Kumar Abhishek

to the
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur
May 2006

Certificate

This is to certify that the work contained in the thesis entitled "Clustering Genes and Inferring Gene Regulatory Networks using Multiple Information Sources", by *Kumar Abhishek*, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

May 25, 2006

(Prof. Harish Karnick)
Professor
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur

Abstract

Gene clustering **is grouping genes together in clusters on the basis of some similarity measure. This problem has been extensively studied by the bioinformatics community. Most of the techniques employed for clustering genes use gene expression data as the only source of information. We propose a novel hybrid approach towards gene clustering which utilizes useful biological information available in the form of a gene ontology, *GO*, along with gene expression data to perform the clustering task. The proposed approach has been tested on Yeast cell cycle data and performed better than existing methods like k-means, random clustering, and algorithm without GO.**

Expression of a gene is usually regulated by the expression of some other genes in the genome. So it is a collaborative effort of genes which are coupled to each other in finally performing a needed function and not isolated genes contributing their parts independently. *Inferring gene regulatory network* from gene expression data has been an active area of research in the bioinformatics community. Inferring gene networks have been a hard problem to solve and there is no computational method which reliably solves this problem. We propose a *GO* based framework for inferring gene network in a hierarchical manner. The proposed framework helps in parallelizing the inference task by splitting it into small manageable groups. The *GO* based gene groups were highly informative; most of the gene pairs forming direct links

during Yeast cell cycle phase G1 were co-grouped. We also propose a novel network inference algorithm which was employed in this framework and outperformed existing methods like the one proposed by Kim et. al. with a significant margin.

Acknowledgements

I would like to wholeheartedly thank Prof. Harish Karnick for his continuous support and encouraging guidance. His valuable suggestions went a long way in the smooth completion of this work. I am sincerely thankful to him for listening patiently to my weird ideas. The line of thinking he used to ask me to pursue were quite helpful in refining the half-baked ideas. He never used to set a deadline which gave me immense freedom and flexibility to enjoy my work. Even on a few weekly meetings when I used to express my inability to work well in the past week, he used to encourage me to work harder in the coming week. I cannot imagine writing a report of this volume without his help. He helped me a lot in improving my crude write up. I consider myself extremely lucky to work under his guidance. It was a truly a memorable experience, I have learnt a lot of good things from his nature and I would love to practise them in my life.

Contents

Chapter 1

Introduction

In this work we have focused on two research problems related to analysis of gene expression data. The first problem concerns *Gene Clustering* which involves forming groups of "similar" genes from gene expression data. Two genes can be defined to be "similar" in terms of correlated expression profiles, or similar functions, or a combination of both. The second problem is *Inferring Gene Regulatory Networks* which involves mining gene expression data to reveal regulation relationships among genes.

1.1 Clustering gene expression data

A gene is said to be expressed when the protein it codes for is synthesized. This expression varies with time in response to signals that reflect the state of the organism. Also, a cell may need different numbers of different proteins so every gene may be expressed at a different level. In a nutshell, expression of a gene varies with time and is different for different genes. A DNA chip is a device that is capable of measuring the concentration of thousands of genes simultaneously and is referred to as a DNA microarray. A typical experiment provides expression profiles of several thousands of genes with a few samples($\sim$100). This leads to the formation of a gene expression data matrix of size of the order of a few millions(i.e. $100 * 10000$) . This vast amount of data is available to be mined in order to extract the rich biological information embedded within it. Clustering is one of the data mining techniques applied to the expression matrix data. Analysis of this huge amount of

data in a monolithic chunk would be a difficult task. We need to reduce the dimensionality of the system. Clustering is aimed at the task of grouping "similar" genes together which in turn helps in reducing the dimensionality of the system. The groups which are formed through clustering should be "meaningful" enough in order to facilitate further analysis. Some of the useful information which can be derived from identification of co-expressed genes are:

- It has been hypothesized that co-expressed genes serve similar or related functions. So clustering can help in grouping genes doing similar functions.
- Predicting functions of unknown genes. Genes whose functions are not known are likely to have functions similar to the members of the cluster to which they belong.
- With high correlation genes which are co-expressed are also co-regulated.

The problem of clustering can be framed as:

- Input: Gene expression data matrix D of order $n * m$ where n is the number of genes and m is the number of samples.
- Method: Clustering Algorithm which processes the input and groups "similar" genes together. Similarity measure depends on the algorithm used; this measure is used in a way such that genes belonging to the same group are more "similar" than genes belonging to foreign or different groups.
- Output: Set of clusters $C = \{C_i, 1 <= i <= k\}$, where k is the number of clusters and C_i's are the clusters i.e., groups of genes.

Many algorithms have been developed to perform the clustering task. Methods such as hierarchical clustering, k-means clustering, self organizing maps have been used to cluster gene expression profiles. Also a few algorithms such as CLICK and Minimum Spanning Tree based clustering have been developed which utilize graph theoretic techniques to form clusters. We review each technique in section 2.1 in a more detailed manner.
Our aim is to develop a technique which could form "meaningful" clusters from gene expression data. Clusters should be meaningful in the sense that members of a particular cluster

should have similar functions and genes having unknown functions should be strongly co-expressed with other members of the cluster. Strong correlation in expression enhances the confidence with which prediction of functions is made for unknown genes.

1.2 Our contribution

We propose a novel hybrid approach towards gene clustering. It utilizes useful biological information available in the form of a gene ontology, GO, (discussed in section 2.2) along with gene expression data to perform the clustering task. The proposed approach has been tested on Yeast cell cycle data and been found to perform better than k-means clustering, random clustering, and a variant of the proposed algorithm i.e., one which ignores biological information. Also, the proposed evaluation function for evaluating clusters can be employed in other clustering algorithms.

1.3 Inferring gene regulatory networks

In order to understand the overall functioning of a cell, we need to study the behavior of genes as a group and not as isolated entities. This is because expression of a gene is usually regulated by the expression of some other genes in the genome. So it is a collaborative effort of genes which are coupled to each other in finally performing a needed function and not isolated genes contributing their parts independently. A gene codes for RNA molecules which translate into proteins. These RNA and proteins regulate the expression of other genes. This interaction between genes, RNA molecules, and proteins produces a regulatory network called a gene regulatory network. As mentioned in [1] some of the applications of gene regulatory networks are:

- Finding drug targets.
- Classification of normal versus diseased tissues.
- Classification of diseases in terms of cellular networks.
- Study of cellular control.

- Modeling of biological processes.
- Mapping of related networks across phylogeny.

A gene network can be represented by a directed graph in which nodes are genes and edges represent regulation(activation or inhibition) between genes. The problem of inferring gene regulatory networks can be formulated as:

- Input: Gene expression data matrix D of order $n * m$ where n is the number of genes and m is the number of samples.
- Method: Inference algorithm which uses D to infer regulation relationships between genes.
- Output: Directed Graph $G = \{V, E\}$ where V is the set of n gene nodes and E is the set of edges between nodes belonging to V. Each edge belonging to E also possesses a "type" property which can take values 'Activation' and 'Inhibition' depending on the regulation type between connecting gene nodes.

Inferring gene regulatory network from gene expression data has been an active area of research in the bio-informatics community. Inferring gene networks have been a hard problem to solve and there is no computational method which reliably solves this problem. A variety of approaches involving learning methodologies and statistical models have been applied to solve this problem. Each technique solves the problem only partially and suffers to varying degrees of the inability to handle computational complexity, error margins, modeling capabilities etc. Some of the approaches used to model gene regulatory networks are:

- Bayesian networks
- Boolean networks
- Probabilistic Boolean Networks
- Module Networks

- Differential Equations

We will discuss these approaches in a more detailed manner in section 4.1.
Our aim is to come up with a framework which can incorporate multiple inference algorithms and can be used to solve the inference task efficiently by using multiple processors and information sources.

1.4 Our Contribution

This work makes two contributions related to inferring gene regulatory networks from expression data:

- It proposes a novel GO based framework for inferring gene networks in a hierarchical manner. The proposed framework is flexible and can incorporate a variety of inference algorithms. Also it helps in parallelizing the inference task by splitting it into small manageable groups. We show the effectiveness of GO based gene groups produced by this framework. We present results related to Bayesian inference over GO based groups using this framework on Yeast cell cycle data and show it to be better than when applied to random groups.

- It also proposes a novel inference algorithm which can infer gene regulation relationships from expression data. The inference algorithm employed in the GO based framework can be used to efficiently infer gene networks. We present the results of applying this algorithm to the Yeast cell cycle data. We compare the network obtained with the real biological network present in Yeast cell.

1.5 Organization of the report

This chapter introduced gene clustering and inferring genetic regulatory networks along with our contributions. Chapter 2 deals with a detailed description of clustering algorithms and the Gene Ontology(GO). Chapter 3 presents our proposed clustering algorithm and we do a comparative analysis of results obtained on Yeast cell cycle data. Chapter 4 discusses

techniques for inferring gene networks and the difficulties faced by these techniques. It also highlights some of the issues which we are targeting through our approach. Chapter 5 presents our proposed GO based framework for inferring gene networks in a hierarchical fashion. Chapter 6 presents a novel inference algorithm and discussed the results obtained for the Yeast cell cycle data. Chapter 7 contains conclusions and pointers to directions in which the work can be extended.

Chapter 2

Clustering Gene Expression Data

2.1 Algorithms for clustering gene expression data

In this section we discuss widely used clustering algorithms that have been used to cluster gene expression data.

2.1.1 k-means clustering

The k-means clustering algorithm uses the metric properties of the vector space(k stands for the number of clusters desired). It first divides the data space into k parts and calculates the means of each subspace. Then individual points are moved around so that each point belongs to a cluster whose means is nearest to that point. This movement of points changes the cluster subspaces, the algorithm then re-calculates the means and the procedure is repeated iteratively until convergence or after maximum number of iterations. This algorithm is an EM-type algorithm that eventually attains a local optimum. K-means requires an a priori estimate of the number of clusters and it suffers from the problem of converging to local optima. More details can be found in[16],[17].

2.1.2 Hierarchical Clustering

Hierarchical clustering is widely used for clustering micro-array data. It views each data point as a node and at each iterative step merges two "closest" nodes to form a new node. A tree is constructed in a bottom up fashion after $n-1$ steps where n is the number of data points. For merging two nodes different linkage methods can be used which decide the "closest" pair of nodes to merge. Single linkage calculates shortest distance of pair-wise points from the two nodes. Complete linkage calculates largest distance of pair-wise points from the two nodes. Average linkage calculates average distance of pair-wise points from the two nodes. The problem with hierarchical clustering is it decides the nodes to be merged locally on the basis of some form of linkages without taking a global objective into account. Also, once nodes are merged they cannot be separated in later steps. More details can be found in[15].

2.1.3 Minimum Spanning Tree Based Gene Clustering

Minimum Spanning tree of a graph is a tree which has all the vertices of the graph as nodes such that the total edge weight of the tree is minimum. This approach utilizes graph theory to find gene clusters. It views data points as nodes and defines a distance measure between nodes(like Euclidean distance or Mahalanobis distance). So the edge weight between nodes is the distance between the nodes. A minimum spanning tree is constructed, now in order to get k clusters the tree has to be k-partitioned by removing $k-1$ edges ([19]). There can be various criteria for removing $k-1$ edges like pick the longest $k-1$ edges or partition with a global objective of minimizing the total distance between the center of each cluster and its data points. Unlike algorithms like k-means or Self Organizing Maps which depend on the geometry of cluster boundaries this method is robust enough to be unaffected by geometries or regularities of data points. This method is very sensitive to noise; outliers create problems in partitioning.

2.1.4 CLICK

CLICK (Cluster Identification via Connective Kernels) is another graph theoretic approach towards gene clustering([20]). It searches for "tight clusters" or kernels. "Tight clusters" or

kernels are clusters which are highly coupled and cannot be further divided (as explained later in this section). The tight patterns or kernels are extracted from the data and it leaves out scattered genes which are incapable of forming a kernel. Initially a fully connected graph is generated; then CLICK recursively divides the graph into two using minimum weight cut until a kernel condition is met. The kernel condition tests if a cluster is a "tight cluster", hence, cannot be further divided. This approach is robust to errors and does not require an a priori knowledge of the number of clusters. But it suffers from the local search problem; while searching for kernels the search may not be exhaustive.

2.1.5 Self Organizing Maps

A Self Organizing Map(SOM) has a set of nodes with a topology(like two-dimensional grid) and a distance measure defined between nodes([18]). Now if the gene expression has d samples then nodes are linked to the input data points in the d dimensional space by weighted edges. Initial weights are assigned randomly and then iteratively updated. Each iteration involves selection of a data point P and movement of all the nodes in the direction of P with the moving distance being inversely proportional to the distance between P and the node point under consideration. Neighboring points in the initial node topology tend to be mapped to nearby points in d-dimensional space. The process is continued for a number of iterations which is fed as an input to the algorithm. SOM allows partial structure to be imposed on the cluster, are known to be robust to noise, and allows for easy visualization and interpretation. SOMs suffer from the following disadvantages: (a) Requires an a priori knowledge of the number of clusters, (b) no guarantee that clusters would fit in a 2D topology, (c) Sensible to initial parameter setting, and (d) has possibility of getting trapped in locally optimal solutions.

2.2 Gene Ontology

Ontology can be defined to be "*a set of controlled vocabulary that classifies concepts and defines the relationship between them.*"
Gene Ontology(GO) [4] provides *a controlled vocabulary to describe biological knowledge for gene and gene products and the relationships between them.*

We use the definitions and examples provided in [4] which is a repository for all information in GO. The three components of GO are *molecular function*, *biological process*, and *cellular component*. A gene product can have one or more molecular functions, participate in one or more biological processes, and can be a part of one or more cellular components. For example, the gene product *cytochrome c* can be described by the molecular function term *oxidoreductase activity*, the biological process terms *oxidative phosphorylation* and *induction of cell death*, and the cellular component terms *mitochondrial matrix* and *mitochondrial inner membrane*.

Molecular function describes activities at the molecular level; the activities can be performed by individual gene products(narrower functional terms) or by a group of gene products(broader functional terms). For example *binding* is a broad functional term while *adenylate cyclase activity* is a narrower functional term.

A **biological process** is a series of events accomplished by one or more ordered assemblies of molecular functions. For example *DNA replication* and *response to stimulus*.

A **cellular component** is a component of a cell such as nucleus, ribosomes, and cytoskeleton.

GO terms are organized in a Directed Acyclic Graph(DAG), such that child terms are more specialized than parent terms. For example the biological process term *hexose biosynthesis* has two parents, *hexose metabolism* and *monosaccharide biosynthesis*. This is because biosynthesis is a subtype of metabolism, and a hexose is a type of monosaccharide. Each GO term is annotated with a list of gene products. These annotations follow a "true path rule" i.e., "*if the child term describes the gene product, then all its parent terms must also apply to that gene product*". So any gene involved in *hexose biosynthesis* is annotated to this term, it is automatically annotated to both *hexose metabolism* and *monosaccharide biosynthesis*. Also, the edges of the DAG representing relationship between connecting GO terms has a type property which can be "is−a" or "part−of" relationship. For example, nucleus is "part−of" cell and nuclear membrane "is−a" membrane.

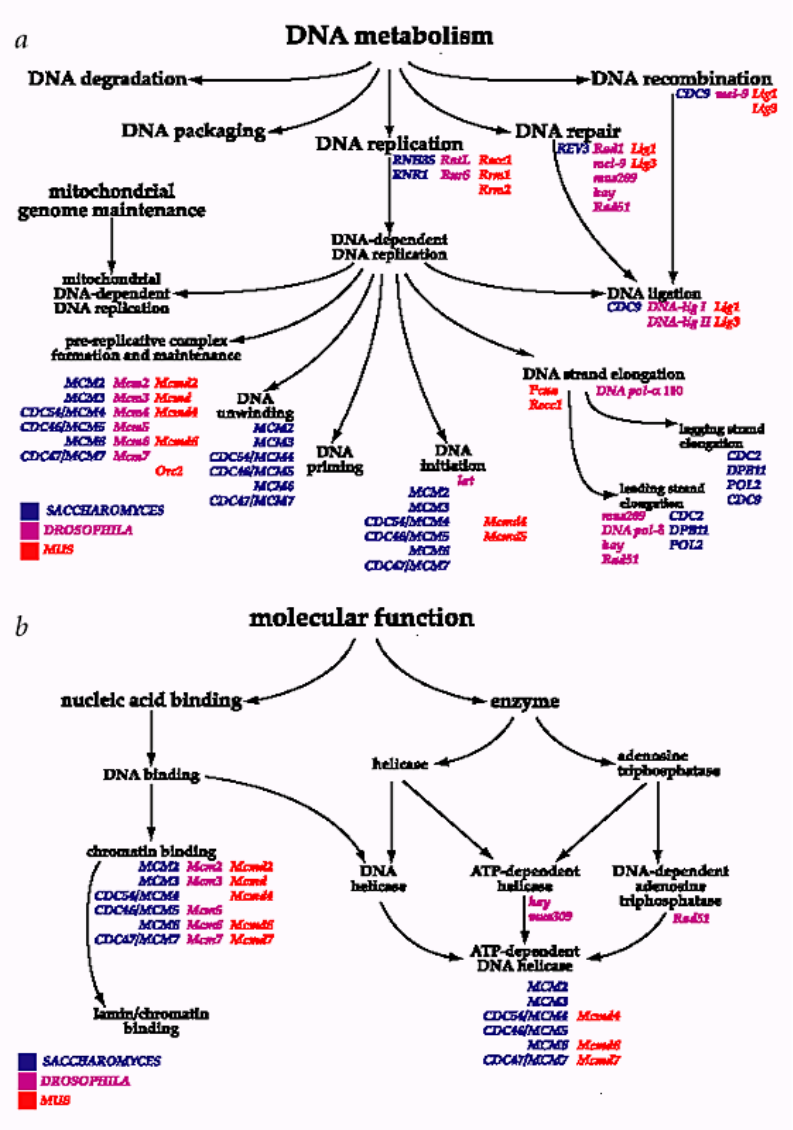


Figure 2.1: The two examples depict a part of function and process ontologies; it also shows how genes are associated with GO nodes. Figure taken from http://www.biomedcentral.com

2.3 Usage of GO information for clustering

Different clustering algorithms give different outputs on the same data set. There should be a measure to assess the quality of the produced clusters and perform a comparative analysis of the algorithms which would enable one to select the best algorithm for a particular data set. Also, it is known that expression based clustering does not always result in clusters which are biologically similar. So there arises a need of some cluster assessment which incorporates biological information that can help in selecting a clustering algorithm best suited to produce "meaningful" clusters. Also, it makes eminent sense to use the biological information during the clustering phase itself to arrive at biologically similar clusters.

Recently valuable biological information present in GO has been incorporated in clustering techniques. Most of the work in this direction focuses on using the Gene Ontology for cluster validation. Speer et al [2] use GO to develop a cluster validity index. The proposed cluster validity index is used to compare the quality of clusters obtained from different clustering algorithms. Until now most of the cluster validity indices were based on properties related to mathematical properties of clusters like tightness of individual clusters, good separation between clusters, or a combination of the two. The indices used in this work can be used to identify functionally similar clusters. It has been shown that biological cluster indices are able to distinguish biologically more homogeneous clusters from less homogeneous ones. Also, expression based clustering was unable to find biologically cohesive clusters which enhances the demand for indices which can identify biologically meaningful clusters.

Bolshakova et al [3] also describe a similar approach using GO for cluster validity assessment; an information content based similarity measure is devised for calculating similarity between genes. The cluster validity index incorporates this similarity measure to assess the quality of clusters obtained. The cluster validity index is able to identify optimal number of clusters in a subset of Yeast cell genes which is consistent with the expected biological structure . Some of the approaches use GO annotation information of the members belonging to a cluster for validating the cluster. Clusters are assigned a functional class on the basis of various criteria like *highest annotations* i.e., annotation which is shared by maximum number of genes in a cluster.

In this work we adopt a hybrid approach towards gene clustering which is both knowledge and data driven. The knowledge source is GO and the data source is the gene expression data. We use biological information from GO both during the clustering phase to arrive at clusters as well as during the cluster validation phase.

Chapter 3

A Knowledge and Data Based Hybrid Approach to Gene Clustering

In the proposed hybrid approach, we aim to incorporate available biological information for producing better gene clusters. We do not make any a priori assumptions on the number of clusters, so our task to find the optimum number of clusters present in the input data set as well as to ensure that each cluster obtained is cohesive both expression wise(determined from input expression data) and function wise(determined from the GO process tree). We compare the proposed algorithm with standard algorithms and also investigate the usefulness of biological information in producing high quality clusters.

In the proposed approach we first calculate a gene distance matrix which takes contribution from two sources: (a) GO DAG is used to define GO distance between genes, and (b) Expression Data is used to define expression distance between genes. A fully connected graph is constructed with genes as nodes and edge weights being the net distance between genes. This graph is fed to Minimum Spanning Tree algorithm; k clusters can be found by k-partitioning of the MST formed. The algorithm varies number of desired clusters and ultimately finds the set of optimal clusters (measured by an objective function defined later) corresponding to optimal number of clusters.

3.1 Transformation of GO DAG into GO directed tree

We use the GO process ontology, for determining the closeness between GO nodes. GO is represented in the form of a Directed Acyclic Graph (DAG). Each GO node is annotated with a list (possibly empty) of genes, so some nodes can be left un-annotated. Each GO node can have multiple parents and multiple children. We define the "*level*" of a GO node as the number of nodes from the node to the root. The "*Level*" of a GO node is used later to define distance between GO nodes(discussed in the next section). Since a particular node can have multiple parents so directly using the DAG for defining the level of GO nodes would not be possible as there can be multiple paths from the root to the node ruling out the existence of a unique "*level*". So we perform a transformation on the DAG structure to get a directed tree structure which allows us to define the "*level*" of a node uniquely. The procedure for converting DAG to a directed tree is as follows:

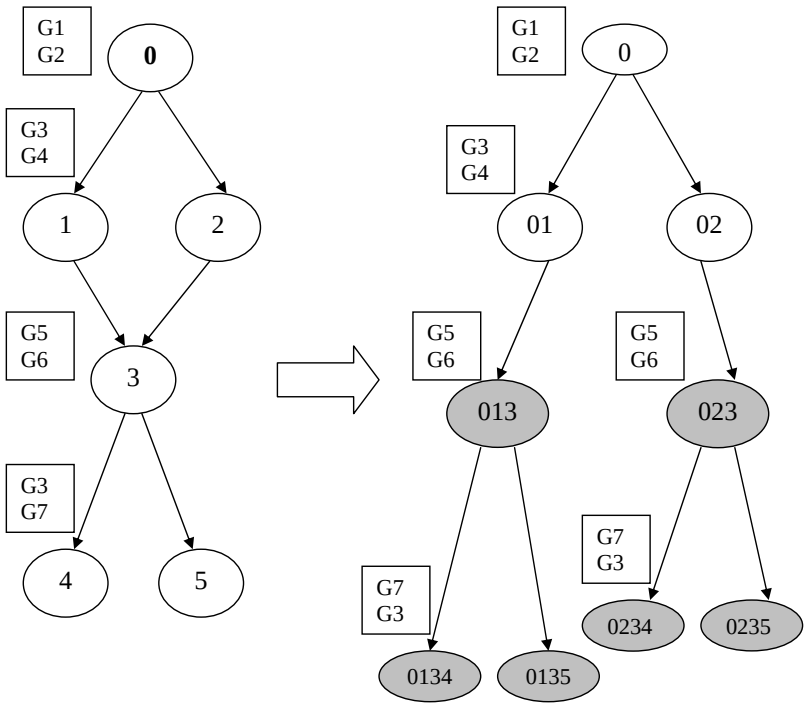
procedure tranformDAG
for each node x ε DAG do {

- If node x has $k > 1$ parents {
  - Create k nodes and make each of the newly created nodes a single child of a distinct parent of x.
  - Replicate annotations of x across the k newly created nodes.
  - Make each child of x a child of each newly created node.
  - Remove node x.

  }

} while(Change in DAG)

We illustrate the procedure by showing a sample transformation in Figure 3.1. This directed tree would be used to define distance measure between GO nodes.



The left structure is GO DAG, the right structure is directed tree obtained after transformation. The ovals represent nodes (shaded ones are newly created nodes); the labels in the rectangular boxes adjacent to the ovals are the gene-annotations of the nodes.

Figure 3.1: Transformation of GO DAG

3.2 Distance measure between GO nodes

As mentioned in the previous section the level of a node is the depth of the node in the directed tree obtained after transforming the GO DAG. We refer to the tree by Γ

Definitions:

- *Weight function:* We define a weight function $f : \{1, 2, ..., N\} \longrightarrow R$ where N is the maximum level of any node in Γ, R is the set of real numbers, f is a decreasing function i.e. $i < j \Rightarrow f(i) > f(j)$. So, we have a weight associated with each level of the directed tree.

- *Least Common Ancestor:* Least Common Ancestor of two nodes of the directed tree is defined to be the common ancestor of the two nodes which is at the highest level or farthest from the root.

- The distance between two nodes i and j is defined to be the weight of the level of the "*Least Common Ancestor*" (LCA) of the two nodes i and j. As one goes farther away from the root the nodes become increasingly specific. So children of a node are more similar to each other when the node is farther away from the root. Therefore, two nodes which have a common ancestor farther away from the root are more similar to each other than those which have common ancestor closer to the root. The proof that it is in fact a distance metric can be found in [5] .

3.3 Calculation of GO distance matrix from the directed GO tree

The tree Γ allows us to calculate a distance between any two gene pairs. We first prepare a n*m table T where n is the total number of genes and m is the number of nodes in the GO DAG. A row represents a gene and a column represents the attributes of the GO node. We treat each GO node as an attribute. As a single gene may be annotated to multiple nodes we check for the list of annotations of a single gene and put a 1 in every attribute column with which the particular gene is annotated. So we get a binary table, we use this table

	0	1	2	3	4	5
G1	1	0	0	0	0	0
G2	1	0	0	0	0	0
G3	0	1	0	0	1	0
G4	0	1	0	0	0	0
G5	0	0	0	1	0	0
G6	0	0	0	1	0	0
G7	0	0	0	0	1	0

The binary table constructed from the GO DAG given in Figure 1. G1,…G7 are the genes and 0,1,…5 are the attributes or the DAG nodes.

	G1	G2	G3	G4	G5	G6	G7
G1	0	0	4	4	4	4	4
G2	0	0	4	4	4	4	4
G3	4	4	0	3	3	3	3
G4	4	4	3	0	3	3	3
G5	4	4	3	3	0	0	2
G6	4	4	3	3	0	0	2
G7	4	4	3	3	2	2	0

Suppose the weight function f from [1:4] corresponding to the four levels of directed tree obtained in Figure 1 be f(i)=5-i where i is the level. The gene distance matrix obtained using the binary table and the distance measure defined to be weight of level of LCA is shown in the above table.

Figure 3.2: Calculation of GO distance

for calculating gene distance. However, genes which are not annotated to any node are not considered in calculating gene distance. The gene distance between two genes can be thought of as average We define diff(i, j) to be number of differing entries in corresponding columns of rows i and j, this quantity helps in calculating the average distance between genes corresponding to row i and j. The gene distance between genes g_i and g_j is defined to be:

$$GOdist(g_i, g_j) = \frac{1}{diff(i,j)} * \sum_{1<=\alpha<=m} \sum_{1<=\beta<=m} (T_{i\alpha} - T_{j\beta})^2 * d(\alpha, \beta)$$

where $d(\alpha, \beta)$ is the GO distance between GO nodes α and β(as defined in the previous section). A sample demonstration of gene distance calculation can be found in Figure 3.2. We get a gene distance matrix G of the order $n * n$ where n is the number of genes.

3.4 Calculation of expression data distance matrix

We are provided with gene expression profiles; we use the d observations associated with each gene to constitute a d dimensional vector to represent the gene. The expression distance measure between two genes is the Euclidean distance between the two d-dimensional observation vectors of the two genes. We calculate the expression distance matrix E of order $n * n$ where n is the number of genes.

3.5 Minimum Spanning Tree based Gene Clustering

We use a graph theoretic approach towards gene clustering. We construct a fully connected graph with genes as nodes and edge weight between node i and node j equal to the net distance (sum of GO distance and expression distance) between gene i and gene j. We find the minimum spanning tree of this fully connected graph. The clustering task now boils down to partitioning this minimum spanning tree into k subtrees where k is the number of desired clusters. We apply the following procedure to obtain the minimum spanning tree:

procedure formMST

- Calculate GO distance matrix G.

- Calculate Expression distance matrix E.
- Scale both G and E to the same range and add them to get net distance D.
- Form a fully connected graph with genes as nodes and use distance between those genes as obtained from D as the edge weight.
- Apply Prim's algorithm([22]) to the above graph to get the MST.

3.6 Iterative Clustering Algorithm

Input to the algorithm is the MST obtained in the previous section and output is a set of clusters which are optimal in homogeneity as well as in number. Before we present this part of the algorithm we define some terms needed in the algorithm.
Definitions:

- Let $C = \{g_i, 1 <= i <= k\}$ be a gene cluster with members g_i.Average GO distance GOavg(C) corresponding to cluster C:

$$GOavg(C) = \sum_{1<=i,j<=k} \frac{GOdist(g_i, g_j)}{^kC_2}$$

  $GOdist(g_i, g_j)$ is the GO distance measure between g_i and g_j based on the GO distance matrix.

- Scatter of a cluster $Scatter(C)$:

$$Scatter(C) = \sum_{1<=i<=k} (x_i - \mu)(x_i - \mu)^T$$

  where x_i is the expression vector of gene g_i and μ is the average expression vector for the cluster C. $(x_i - \mu)^T$ is the transpose of the vector $(x_i - \mu)$.

- The objective function for partitioning the MST for a given number of clusters k F(k):

$$F(k) = \sum_{1<=i<=k} (GOavg(C_i) + Scatter(C_i))/k$$

- $Score_i$ is the minimum value of the objective function $F(i)$ obtained for a given number of clusters i and $Cluster_i$ is the corresponding optimal cluster set.

We use a two level approach to optimize the quality of obtained clusters. The first level concerns itself with optimizing the structure of clusters keeping the number of clusters fixed. The second level builds on top of the first level in that it concerns itself with selecting the optimum number of clusters, the clusters corresponding to that number is the final output. We partition the MST to get the clusters. The algorithm attempts to select clusters which minimize the objective function $F(k)$. The iterative clustering algorithm is as follows:

procedure performClustering

- Fix maximum number of clusters i.e., MAXCLUSTERS
- Initialize k to 1.
- while $k < MAXCLUSTERS$ do {

  Perform a random k-partitioning of the MST by removing k-1 edges to get k clusters.
  $F(k)_{old} = \infty$, $F(k)_{new} = F(k)$; $F(k)_{old}$ is the value of F(k) in the previous iteration (described below) and $F(k)_{new}$ is the value of F(k) after current iteration.
  Set a change threshold ϵ.
  while($|F(k)_{new} - F(k)_{old}| > \epsilon$) {

  $F(k)_{old} = F(k)_{new}$.
  For each pair of adjacent clusters {

  Merge the two clusters.
  Select the edge which globally optimizes the 2-partitioning of the merged cluster measured by objective function F(k).
  $F(k)_{new} = F(k)$

  }

}
$Score_k = F(k)$
Save the optimal cluster as $Cluster_k$
Increment k.

}

- Search the list of scores to find the minimum element $Score_{min}$ and output the cluster set $Cluster_{min}$ corresponding to it.

3.7 Experimentation and Results

We used the data set of Yeast cell cycle data in which activity was measured at 18 time points. We used two subset of genes each consisting of 500 genes for testing the algorithm. Cluster validation was done using a figure of merit score(FOM). We implemented our algorithm and k-means and then did a comparative analysis of clustering results on the chosen data set. We compared our algorithm with k-means, random, and the algorithm without using GO distances.

3.7.1 Evaluation Methodology

As described in [7] , we use Figure of Merit(FOM) scores to assess the quality of the clusters obtained from different algorithms and compare the algorithms. It's basically a leave one out approach, clustering is performed using all but one of the experimental conditions in the data set(17 in our case). The left out condition is used assess the predictive power of the algorithms. It has been found that this measure of cluster quality is consistent with external standards of cluster quality using biological knowledge. Now we will try to define FOM as described in [7] . Suppose e is the left out condition of the m experimental conditions present in the data set, let there be k clusters $C_1, C_2, ...C_k$, and suppose the $E_{g,e}$ be the expression level of g under condition e. Let $\mu_{C_i}(e)$ be the average expression level in condition e for genes of cluster C_i.

$$FOM(e,k) = \sqrt{\frac{1}{n} * \sum_{i=1tok} \sum_{x \varepsilon C_i} (E_{x,e} - \mu_{C_i}(e))^2}$$

where n is the number of genes.

$$FOM(k) = \sum_{e=1tom} FOM(e,k)$$

We use this score to assess the quality of clusters obtained from our algorithm and compare it with the k-means and the random algorithms to evaluate the performance of the proposed algorithm. We also remove the GO distances from the algorithm to investigate how GO distances affect the performance of the algorithm. As can be inferred from the definition of FOM, lower is the FOM score higher is the cluster quality because lower FOM means lower scatter from the actual value at the test condition.

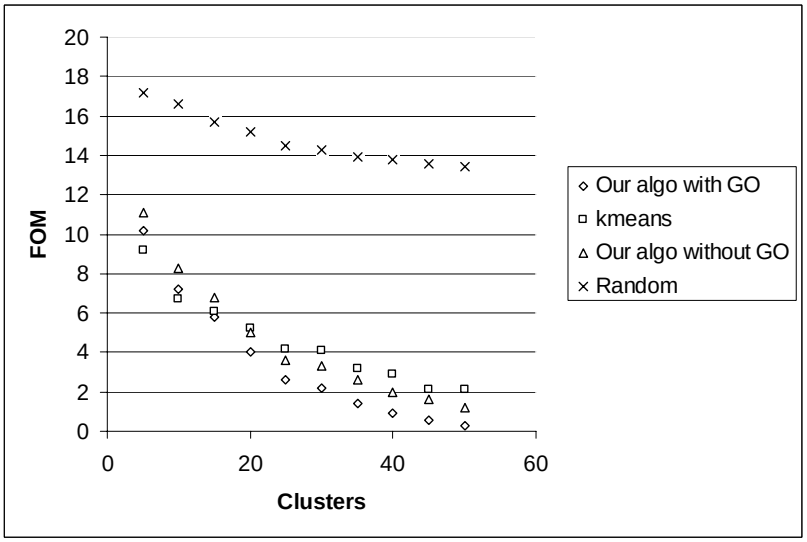
3.7.2 Results

The performance graph for the four algorithms on the two data sets is shown in Figure 3.3. The following can be observed from the upper plot of Figure 3.3 i.e., Data Set 1:

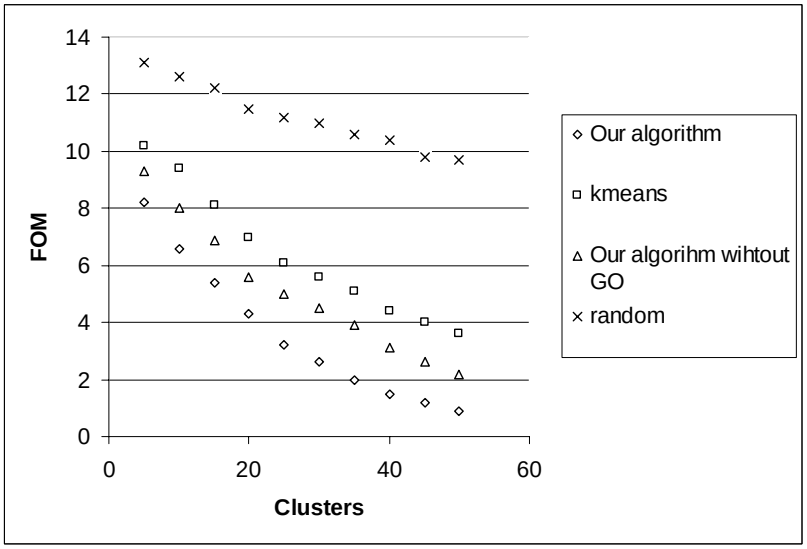
- The proposed algorithm outperforms k-means and the random algorithm for greater than 10 clusters.
- The proposed algorithm without GO distances outperforms k-means after 20 clusters.
- The proposed algorithm without GO distances outperforms random algorithm right from the beginning.
- The proposed algorithm outperforms the one without GO distances right from the beginning.

Performance plots on a second data set (lower plot of Figure 3.3) reveals the following:

- The proposed algorithm with and without GO distances outperforms k-means and random algorithm right from the beginning.



DATA SET 1



DATA SET 2

Figure 3.3: Evaluation of the proposed clustering algorithm

- The proposed algorithm outperforms the one without GO distances right from the beginning.

It can be observed that the proposed algorithm outperforms k-means and the random algorithm comprehensively. Interesting to investigate was the behavior of this algorithm without GO distances. It can be seen the proposed algorithm has lower FOM than the algorithm without GO distances. It shows that GO distances definitely improve the cluster quality produced by the algorithm.

3.8 Conclusions

We compared our algorithm with k-means and the random algorithm and found that the proposed algorithm performs better than both. We also compared the proposed algorithm with its variant i.e., algorithm without GO distances. We found that introduction of GO distance contributes positively to the quality of obtained clusters. It will help to parallelize the proposed algorithm to bring down computational requirements at a particular workstation and perform cluster analysis for even larger data sets reasonably fast. Also, alternate evaluation methods can be used to validate the clusters obtained.

Chapter 4

Inferring gene regulatory networks

Gene regulatory networks have been well understood for several biological processes. Modeling gene networks is of great help for biologists who can use the model to simulate experiments and guide biological experimentation along more productive pathways. A number of algorithms have been proposed by researchers to solve the problem of inferring gene regulatory networks from gene expression data. Before we discuss these algorithms in more detail we look at some of the inherent difficulties in this problem:

- *Gene Expression Data Errors:* Expression data is accompanied by errors which hampers the performance of inference algorithms. Some of the possible errors as mentioned in [1] are:
  - *Biochemical Noise:* Small fluctuation in concentration of molecules which have small concentrations(Biochemical Noise) may cause significant variation in gene expression in cells.
  - *Bio-technical Errors:* Errors introduced during m-RNA preparation, Polymerase chain reaction (PCR), hybridization.
  - *Device Errors:* Devices like filters and glass- slides may introduce errors.
  - *Algorithmic Errors:* Image analysis algorithms used for calibrating signal intensity are affected by errors mentioned above and introduce new errors in the data.

- *Huge problem space:* Inferring gene networks involving thousands of genes requires the exploration of a huge problem space. The exponential problem space makes this problem very hard to solve computationally using only gene expression profiles as input.

- *Shortage of Data:* The number of samples available is typically a few tens, while the number of variables is of the order of a few thousands. All the regulation relationships have to be learnt from this small amount of available data.

On the basis of the above difficulties inherent in the problem of inferring gene regulatory networks, one would like an inference algorithm to have the following properties:

- *Robustness:* The algorithm should be robust enough to deal with gene expression data errors.

- *Low Data Requirement:* The algorithm should be able to infer regulation relationships among a large number of genes from the small amount of data samples available.

- *Scalable:* The algorithm should scale to large sized networks.

- *Low computational complexity:* In order to work well for large networks the algorithm should be computationally efficient so that it can infer all the relationships within a reasonable time limit.

- *Biologically Inspired:* The algorithm should be able to use existing biological information, where available, along with the expression data. This is likely to reduce the problem space size and data requirement, and may also help in improving the quality of the output.

4.1 Algorithms for inferring gene regulatory networks

In this section we discuss a few techniques employed for inferring gene regulatory networks. We also point out the advantages and disadvantages of each technique(the list of techniques mentioned is not exhaustive).

- *Differential Equations:*A system of linear differential equations is used to infer gene regulation relationships [8]. Since biological gene networks are expected to be sparse, some of the regulation matrix entries are set to zero(determined by information criterion) and inference is carried out using only the non-zero entries.
  Advantages:
  - Represents gene interaction explicitly in a numerical form.
  - Allows loops in the inferred network.
  - Suitable for modeling continuous behavior.

  Disadvantages:
  - Algorithm breaks down for large networks.
  - Difficult to model discrete state transitions.

- *Boolean Networks:* This algorithm assumes two states for gene expression; a gene is either ON(expressed) or OFF(inhibited). Expression of a gene is modeled as a boolean function of other boolean gene variables.
  Advantages: Explores the full state space.
  Disadvantages:
  - Mostly genes have multilevel expression so this method is not suitable in such systems.
  - The system is rigidly deterministic.

- *Probabilistic Boolean Networks:* To overcome determinism of Boolean networks, a probabilistic element was introduced in Boolean networks [9]. Each gene node's expression is now modeled by a vector of boolean functions and each member function is assigned a probability with which it would be used for predicting the particular gene's expression value.
  Advantages:
  - More robust so tolerant to errors.

  - Flexible in terms of number of functions used to predict a gene's expression as well as number of gene variables on which these functions operate.
  - Can incorporate biological knowledge easily.

  Disadvantages: Unable to incorporate multilevel gene expression which is quite common.

- *Bayesian Networks:* As mentioned in [10], "*Bayesian networks are interpretable and flexible models for representing probabilistic relationships between multiple interacting agents*". Qualitatively, the structure of a Bayesian network represents interaction between genes quantitatively described by a conditional probability distribution.
  Advantages:

  - Probabilistic nature allows for handling noise.
  - Allows confidence in the inferred network to be evaluated objectively.

  Disadvantages:

  - Can only infer a Directed Acyclic Graph. Removal of loops means that there can be no feedback which is known to be quite common in biological systems.

- *Module Networks:*[11] describes a recent approach towards modeling gene regulatory networks as module networks. A module is a group of coregulated genes; learning a module network involves identification of modules and connections between them from gene expression data. Structure of a module network can be thought of as a collection of modules with connections between them. The learning algorithm is basically an expectation maximization algorithm(EM), in which first step keeps the structure fixed and learns the parameters; while the second step varies the structure keeping the parameters fixed. The above process is repeated until convergence, i.e., module network score has reached a maxima.
  Advantages:

  - Can identify conditions under which regulation occurs.

  - Generates testable hypothesis regarding role of regulators and the conditions under which regulation takes place.

  Disadvantages:

  - Number of modules need to be fixed a priori.
  - Unable to detect "focused" relationships involving a regulator and its target; all the relationships are learnt at modular level.

- *Markov Models*: Utilizing the stochastic nature of gene regulation; the network is modeled using a Markovian process. States in the Markov model can be biological entities like genes, proteins, or RNAs and the state transition is Markovian i.e., probability of making a transition at any given time depends only on the state of the system at that time.
  Advantages:

  - Both discrete and continuous valued gene expression can be modeled.
  - Captures stochastic nature of gene regulation and has the usual advantage of probabilistic models i.e., robustness.

  Disadvantages:

  - Huge search space to explore, computationally expensive for large systems.
  - Biological measurements are not always known to the detail required by the model for probability calculations.

4.2 Proposed algorithm

In this work we propose an approach which aims to address the three desired requirements of inference algorithms:

- *Low Data Requirement:* In our technique we divide the input gene set into smaller subsets(see chapter 5). Inferring relationships over smaller gene sets requires fewer data samples.

- *Computationally Efficient:* As mentioned above we divide the gene set into smaller subsets; the inference task can be carried out independently over the subsets. This can be done in parallel thereby speeding up the algorithm(section 5.8).
- *Biologically Inspired*: We use biological knowledge available in the form of GO for partitioning the problem into smaller sub-problems. The primary goal of using biological knowledge is to reduce the complexity of the problem space i.e., the size of the connection (regulation link) space.

Chapter 5

GO Based Framework for Hierarchical Inference

In this chapter we propose a framework for hierarchical inference of gene regulatory networks from gene expression data. The proposed framework is shown to be flexible enough to encompass a variety of inference algorithms. We also discuss the advantages of using a hierarchical approach towards network inference. Results obtained from applying Bayesian inference within this framework over Yeast cell cycle data is also discussed at the end of the chapter.

5.1 Filtering data

A typical input gene set consists of few thousands of genes. It has been observed that a significant proportion of the genes exhibit minor changes in expression values during the experiment. Such genes which are not actively changing expression cannot be utilized for the task of inferring gene regulation relationships purely on the basis of expression data. As inclusion of these genes increases the search space for possible relationships, we filter out these genes. The procedure for filtering out genes is as follows:

procedure filterGenes

- Set a variance threshold ϵ
- for each gene g in the input set do
  {
  - Suppose the input data consists of expression measurement at n time points; let x_i be the expression value measurement at i^{th} time point. Then define mean μ as: $\mu = \Sigma_{1<=i<=n} \frac{x_i}{n}$
  - Variance of expression σ is defined as: $\sigma = \Sigma_{1<=i<=n}(x_i - \mu)^2$
  - if($\sigma < \epsilon$) filter out gene g

  }

At the end of this procedure we obtain genes whose expression changes actively during the experiment; such genes have higher chances of being involved in the regulatory process.

5.2 Filling missing data

It has been found that gene expression data has significant number of 'missing values'. Inference algorithms work better with complete data; also some of the algorithms might not work with expression data having gaps. So it is desirable to fill the missing values with reasonable values. Oba et. al.[12] describe an approach for filling missing values in gene expression data. It is basically an approach based on Bayesian Principal Component Analysis(BPCA). This method outperforms existing methods like the k-nearest neighbors approach or singular value decomposition. We use this method for filling missing values in the expression data set.

5.3 Detecting Periodicity

Expression of genes associated with periodic biological processes such as cell cycle regulation is known to be rhythmic. DNA micro-arrays allow for identification of periodic genes;

the data consisting of expression values of thousands of genes at a number of time points is fed to an algorithm which can detect periodicity. Detection of periodic genes helps in reducing the input gene set to a set containing periodic genes making it computationally simpler for the inference algorithm. We use Lomb-Scargle periodograms to identify periodic genes; this method has a number of advantages over other approaches. It can deal with uneven time sampling, it can handle missing values directly, and is capable of detecting multiple frequencies. The details of the algorithm and its application to detecting periodicity can be found in [13].

5.4 Grouping genes based on GO information

As discussed in previous chapters, GO is organized in the form of a Directed Acyclic Graph(DAG). This graph has been formed on the basis of biological experiments, observations, and inferences. Several genes are annotated to each GO node or GO term of the DAG. This valuable biological knowledge can be utilized in an effective way for aiding the task of inferring gene regulatory networks. The GO DAG can be used to form groups which can be used as a starting point for the inference task. This group formation helps in significantly reducing the search space for gene regulation relationships. Two genes annotated to a particular GO node or forming a parent-child pair are quite similar to each other in the biological functions they perform or the biological processes in which they participate. It is hypothesized that the gene pairs forming direct links in the gene regulatory networks are very similar to each other function-wise or process-wise, hence, most of such gene pairs must be annotated to nodes quite close to each other in the GO DAG. We propose the following hypothesis for preparing groups:

Hypothesis:
Most of the gene pairs forming direct links in the gene regulatory networks belong to:
(a)Same GO node or GO term, or
(b)One is annotated to the parent GO node and the other is annotated to child GO node.

procedure formGroup

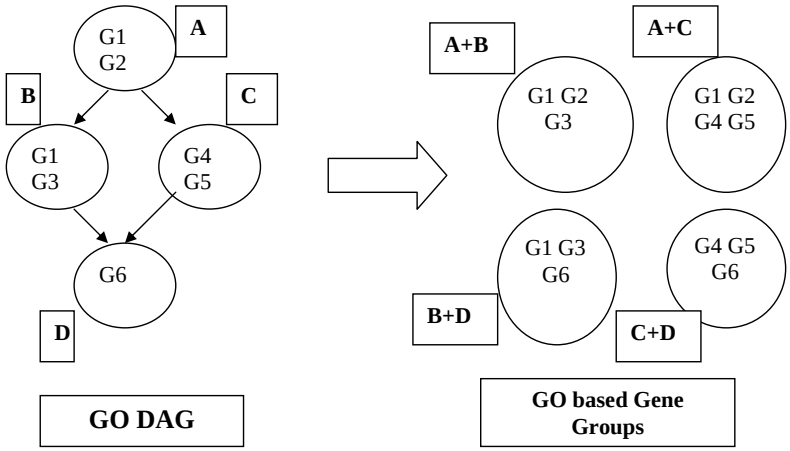
- Identify all possible parent child node pairs in the GO DAG. Suppose the list of pairs formed is $P = \{(p_i, c_i), 1 <= i <= n\}$, where n is the number of pairs formed, p_i is the parent GO node, and c_i is the child GO node.
- for all members (p_i, c_i) of the list P do
  {
  - Find the list of genes annotated to p_i, say it is A_{pi}
  - Find the list of genes annotated to c_i, say it is A_{ci}
  - Form a group $G_i = A_{pi} \bigcup A_{ci}$

  }

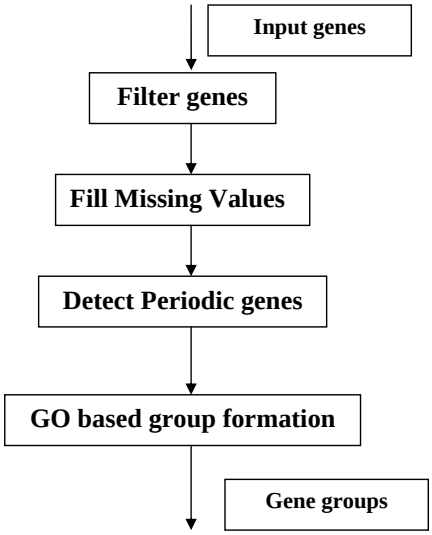
At the end of this procedure we have n groups of genes ($\{G_i, 1 <= i <= n\}$). An example of group formation based on a simple GO DAG is shown in Figure 5.1. Most of the gene pairs forming direct links in the gene regulatory network belong to one or more of these gene groups. These groups which are significantly smaller in size than the input gene set are somewhat easier to work with for inferring relationships.

5.5 Inferring regulatory networks for small groups

A flow chart for the preparation of GO based gene groups which are ready to be used for the inference task is shown in Figure 5.1. There are two classes of inference algorithms for which GO based gene groups can turn out to be useful :
(a) **Partial regulator set algorithms:** These are inference algorithms which can infer direct links to a gene using a group of genes containing a partial list of regulators of that gene.
(b) **Complete regulator set algorithms:** These are inference algorithms which can infer direct links to a gene only using a group of genes containing all the regulators of that gene.



Transformation of GO DAG to GO based gene groups.



Flowchart for obtaining GO based gene groups ready for use with inference algorithm.

Figure 5.1: GO based group formation and flow chart for preprocessing

5.5.1 Partial regulator set algorithms

Partial regulator set algorithms are simpler to deal with, we can directly apply the inference algorithm to the individual GO based gene groups formed at the end of the procedure described in the flow chart. As each of these groups may possibly contain a partial list of regulators for some of the member genes, the direct links can be inferred using expression data for genes belonging to the group. This can be summarized as:

procedure inferPartial

Input: GO based gene groups and their associated expression data
for each of the GO based gene group {
    Input: Read expression data of the member genes
    Method: Apply network inference algorithm to the input expression data
    Output: Regulatory network consisting of genes belonging to this group
}
Output: List of small gene networks corresponding to each of the GO based gene group.

5.5.2 Complete regulator set algorithms

Complete regulator set algorithms are harder to deal with; these algorithms require a transformation of the GO based gene groups formed till this point. We need to form groups which contain the complete set of regulators for a gene. A particular gene may belong to multiple GO based gene groups; each of these groups may possibly contain a partial list of regulators for that gene. Union of all the groups to which this gene belongs forms a set which contains the complete set of regulators for that gene. We try to form regulator groups for each of the genes and then apply the inference algorithm to these regulator groups in order to infer direct links to that particular gene whose regulator group is under consideration. The procedure to be followed is as follows:
procedure inferComplete

- Input: Gene expression data and GO based gene groups
- for each gene g do {

- Find the list of GO based gene groups to which g belongs, say they are $\{G_i, 1 <= i <= n\}$ where G_i's are the gene groups to which g belongs.
- The regulator group for g is defined to be $R = \{\bigcup_{1<=i<=n} G_i\}$
  - Input: Read expression data of genes belonging to regulator group of g i.e. R.
  - Method: Apply the inference algorithm to input expression data.
  - Output: Direct links to gene g

  }
- Output: List of regulatory networks; each network has direct links to a particular gene

5.6 Combining small networks to form overall network

In the previous section, small networks were inferred by applying inference algorithms to GO based gene groups. In this section we present the procedure to combine these small networks in a hierarchical fashion to arrive at the overall network. As seen in the previous section two different approaches were applied to deal with the two classes of inference algorithms i.e., complete regulator set algorithms and partial regulator set algorithms. Similarly, the task of combining the small networks involves different approaches in the two cases. We now discuss the different approaches adopted by the framework while dealing with the two classes of inference algorithms:

5.6.1 Partial regulator set algorithms

We observed in the previous section that inferring networks over GO based gene groups was simpler in the case of partial regulator set algorithms. But the combination task in this case presents a bigger challenge as there might be algorithms which require the inferred network to be of a particular structure, this constraint might be violated when subnetworks

are combined. We would like to divide this class of algorithms into two subclasses:

(a)Constrained Algorithms: Inference algorithms which have constraints on the structure of the inferred gene regulatory networks, like many inference algorithms output a network which is a DAG. The problem in merging two networks which were inferred using this class of algorithms is that union of the two networks may not satisfy the "structure constraint". Structure constraint is a restriction which the inference algorithm imposes on the structure of inferred network like inferred network should be a DAG. For example, if the algorithm requires inferred network to be a DAG, then individually the two subnetworks are a DAG but merging the two networks might not preserve the DAGness of the inferred network. We therefore, combine large network with small ones by dismantling the connections from the smaller one. Initial input now is union of larger network and the dismantled smaller network; this union preserves the structure constraint. This input is now fed to the inference algorithm to infer the merged network (Figures 5.2 and 5.3). The procedure for combining the networks is as follows:

procedure inferConstrained
Input: Small networks corresponding to GO based gene groups obtained as explained in previous section.
Method:

- $n = numberofnetworks$
- $PrevNetworks$ =List of input networks.

for (iteration=1 to α){

- $CurrentNetworks$ =null;
- Sort the list of networks belonging to $PrevNetworks$ in decreasing order of size $D = \{G_i, 1 <= i <= n$, s.t. for all i, j $i > j \Rightarrow size(G_i) \geq size(G_j)\}$
- Sort the list of networks belonging to $PrevNetworks$ in increasing order by size of the network, say we get an sorted list of network as $A = \{G_i, 1 <= i <= n$, s.t. for all i, j $i > j \Rightarrow size(G_i) \leq size(G_j)\}$

- for i=1 to n/2 {
  - Pick i^{th} element network from D, say it is N_1
  - Pick i^{th} element network from A, say it is N_2
  - Remove all the connections from N_2; so we get a group of perfectly isolated genes, say it is IG.
  - Feed $N_1 \bigcup IG$ as the initial network to the inference algorithm. It is possible that no new links are formed, in that case final network is simply $N_1 \bigcup IG$.
  - Add the output network from the above step to $CurrentNetworks$.

  }
- n=n/2;
  $PrevNetworks = CurrentNetworks$

}

Output: Fuse the $CurrentNetworks$ to get the overall network. The procedure for fusing is as follows:
Let us define $RegulatorSet_i$ to be the set of genes which are regulators for gene g_i.
for each network ε $CurrentNetworks$ {
for each gene g_i in the network {
Add the regulator genes of g_i to $RegulatorSet_i$, no duplicate entry is allowed.
}
}
Gene regulatory network is constructed as follows: For every gene node g_i directed edges are made to incident upon it from all the gene nodes in the set $RegulatorSet_i$.

(b)Unconstrained Algorithms: Inference algorithms which have no constraints on the structure of the inferred gene regulatory networks. Merging two networks inferred using this class of algorithms is relatively straightforward (Figures 5.2 and 5.4).
We make use of the assumption(section 5.4) that regulator-regulated pairs are part of the same GO based gene groups to define a parameter α which signifies the number of iterations required for learning links which were missed in the GO based gene groups i.e., cross group

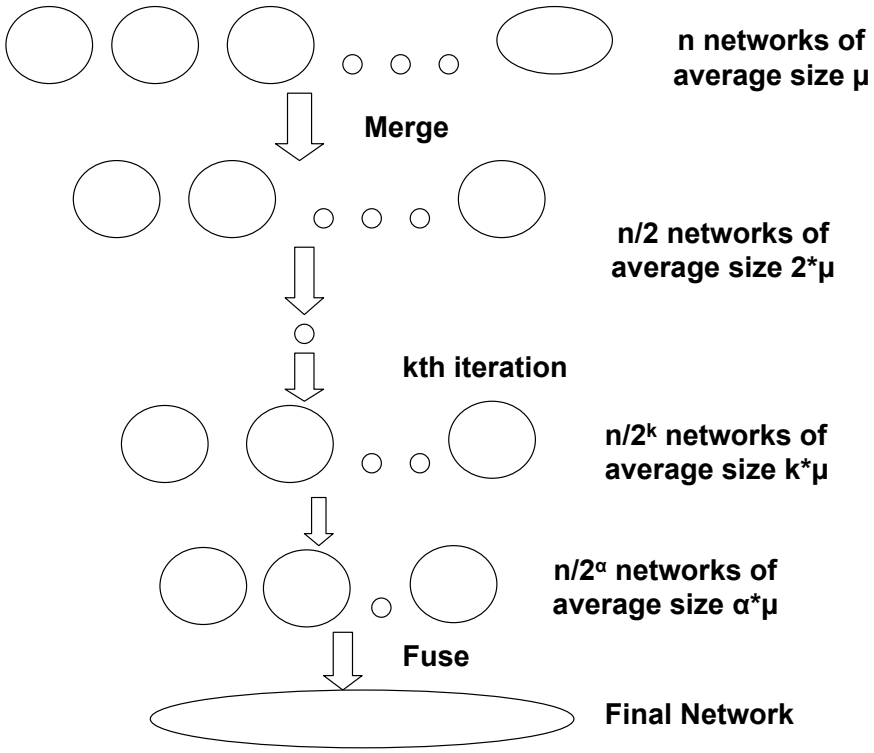


Figure 5.2: Hierarchical inference of gene networks starting from GO based gene networks for partial regulator set algorithms.

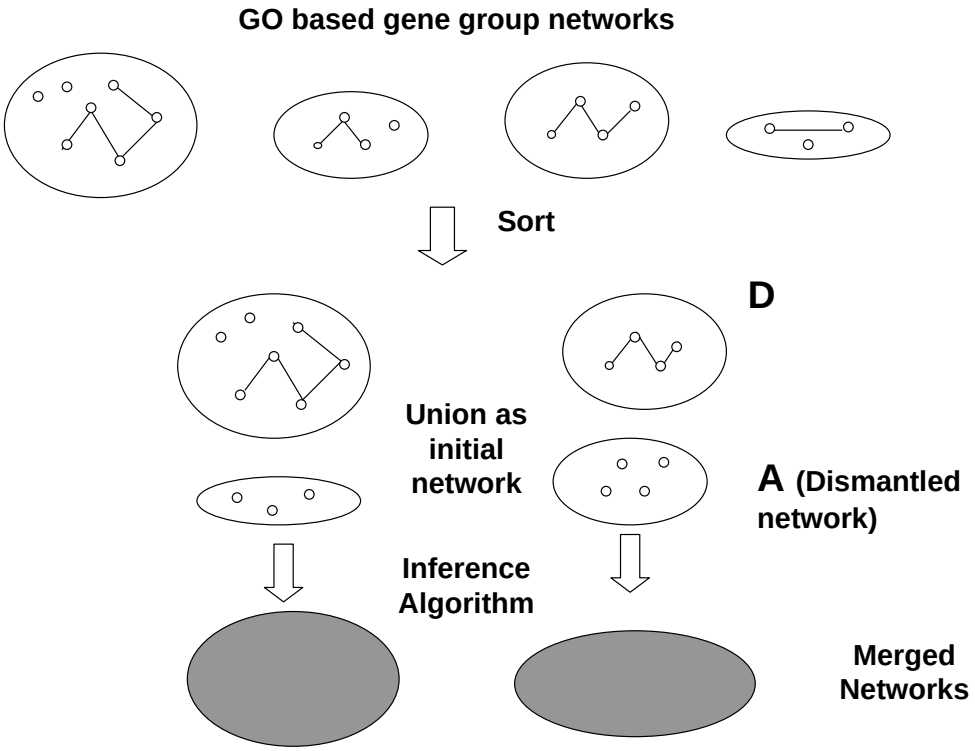


Figure 5.3: Single iteration of "merge" operation for constrained algorithms.

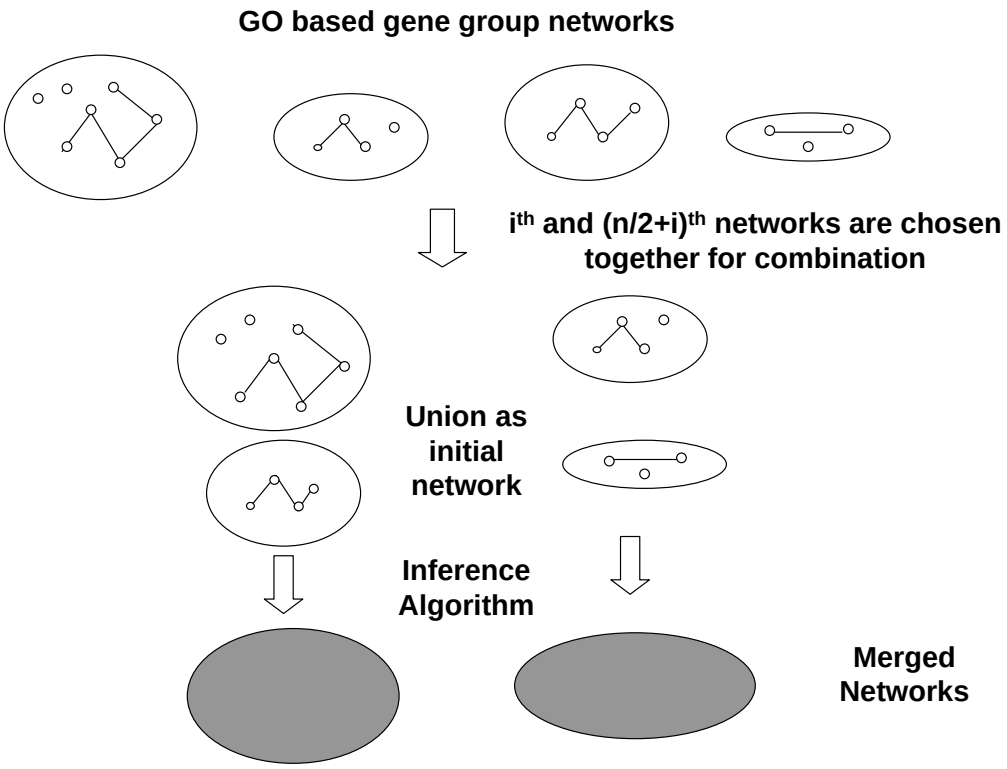


Figure 5.4: Single iteration of "merge" operation for unconstrained algorithms.

links. In order to infer the full gene network, the inference algorithm needs to be applied to each possible gene pair at least once so that all possible direct links are investigated by the algorithm. α tries to capture the number of iterations of merging groups required to reach at the condition where every gene pair have been a part of some group. Typically $\alpha \ll logn$ where n is the number of GO based gene groups. α is a parameter which have to be chosen experimentally for a particular set of GO based gene groups. As α increases the number of distinct gene pairs which are co-grouped increases. However, after a few iterations the number of gene pairs which are not co-grouped at any stage falls sharply. Also, the number of cross links is small so α need not be comparable to $logn$.

procedure inferUnconstrained

Input: Small networks corresponding to GO based gene groups obtained as explained in previous section.
Method:

- $n = numberofnetworks$
- $PrevNetworks$ =List of input networks.

for (iteration=1 to α){

- $CurrentNetworks$ =null;
- for i=1 to n/2 {
  - Pick i^{th} element from $PrevNetworks$, say it is N_1.
  - Pick $(n/2+i)^{th}$ element from $PrevNetworks$, say it is N_2.
  - Feed $N_1 \bigcup N_2$ as the initial network to the inference algorithm. It is possible that no new links are formed, in that case final network is simply $N_1 \bigcup N_2$.
  - Add the output network from the above step to $CurrentNetworks$.

}

- n=n/2;
  $PrevNetworks = CurrentNetworks$

}

Output: Fuse the $CurrentNetworks$ to get the overall network. The procedure for fusing is the same as in the case of inferring constrained networks (described in the previous section).

5.6.2 Complete regulator set algorithms

During the phase of inferring networks on small regulator groups we inferred direct links to each of the gene in the input gene set. This is exactly the same as inferring the full gene network, so nothing extra needs to be done for performing the task of combination in this case.

5.7 Size of Search Space Comparison

In this section we will derive the size of search space of the proposed hierarchical approach and compare it with the size of search space of the algorithm applied over the full gene set. We would like to define a few terms which would aid the process of derivation and comparison.

Terms and Definitions:

μ is the average size of GO based gene group formed in section 5.4.

β is the average membership of a gene i.e., average number of groups a gene belongs to.

n is the number of GO based gene groups

$numgenes$ is the total number of genes annotated in the GO DAG

$NormalSearchSpace$ is the size of search space when working on the full set of genes i.e., $numgenes$. It can be easily seen that:

$NormalSearchSpace = 2 * numgenes * (numgenes - 1)$ i.e., $NormalSearchSpace = \Theta(\frac{n^2 * \mu^2}{\beta^2})$ as $numgenes = \frac{n*\mu}{\beta}$.

5.7.1 Partial regulator set algorithms

Case A: Constrained Algorithms
Constrained Algorithms require sequential execution of **inferPartial** and **inferConstrained**.
The procedure **inferPartial** has total size of search space as the product of number of GO based gene groups and search space size of individual GO based gene groups.
Size of search space for GO based gene group is $2 * \mu * (\mu - 1)$.
Size of search space for inferPartial is $2 * n * \mu * (\mu - 1)$
During each iteration two networks are merged to form a single network, so number of networks halves after every iteration and average size of groups doubles after every iteration.
k^{th} iteration of **inferConstrained** has search space size of $\frac{n}{2^k}$ times the size of search space for merging two networks.
Size of each of network in the k^{th} iteration is $2^{k-1} * \mu$.
Size of search space for merging two networks keeping one network fixed and dismantling the other network is $2 * 2^{k-1} * \mu * (2^{k-1} * \mu - 1) + 2 * (2^{k-1} * \mu)^2$. The first term in the sum i.e., $2 * 2^{k-1} * \mu * (2^{k-1} * \mu - 1)$ is the size of search space for inferring network in the dismantled network while the second term $2 * (2^{k-1} * \mu)^2$ is the size of search space for inferring connections linking the two networks. So the total size of search space for inferConstrained is:
$\sum_{1<=k=\alpha} 2 * \frac{n}{2^k} * (2 * 2^{k-1} * \mu * (2^{k-1} * \mu - 1) + 2 * (2^{k-1} * \mu)^2)$ i.e.,$\Theta(\Sigma_{1<=k<=\alpha} n * \mu^2 * 2^{k-1})$.
Hence the order of size of search space is $\Theta(2^\alpha * n * \mu^2)$.
It is observed that size of search space has reduced by a factor of $\frac{n}{2^{\alpha} * \beta^2}$.
Case B: Unconstrained Algorithms
Unconstrained Algorithms require sequential execution of **inferPartial** and **inferUnconstrained**. Size of search space for inferPartial is $2 * n * \mu^(\mu - 1)$
Size of search space complexity for inferUnconstrained at the k^{th} iteration is:
$\sum_{1<=k=\alpha} 2 * \frac{n}{2^k} * (2^{k-1} * \mu)^2$.
Hence the order of size of search space is $\Theta(2^\alpha * n * \mu^2)$.
In this case also the size of search space has reduced by a factor of $\frac{n}{2^{\alpha} * \beta^2}$.

5.7.2 Complete regulator set algorithms

inferComplete involves conversion of n GO based gene groups to *numgenes* regulator groups. So the average size of each regulator group turns out to be $\beta * \mu$. Size of search space complexity for individual regulator group is $\beta * \mu$. Total size of search space is $\Theta(numgenes * \beta * \mu)$ i.e., $\Theta(n * \mu^2)$.
In this case the size of search space has reduced by a factor of $\frac{n}{\beta^2}$.

5.8 Time Complexity Comparison

In this section we will derive the time complexity of the proposed hierarchical approach and compare it with the complexity of algorithm applied over full gene set. We carry forward the terms and definitions of the previous section and add these terms to the list:
$T(k)$ is the time complexity of the inference algorithm on a input data set consisting of k genes. So the time complexity for application on full gene set consisting of $\frac{n*\mu}{\beta}$ genes is $T(\frac{n*\mu}{\beta})$
numproc is the number of processors available.

5.8.1 Partial regulator set algorithms

Case A: Constrained Algorithms

Constrained Algorithms require sequential execution of **inferPartial** and **inferConstrained**.
inferPartial has time complexity $\frac{n}{numproc} * T(\mu)$.
k^{th} iteration of **inferConstrained** has time complexity as $\frac{n}{2^k * numproc} * T(2^{k-1} * \mu)$.
Total time complexity is $\Sigma_{1<=k<=\alpha} \frac{n}{2^k * numproc} * T(2^{k-1} * \mu)$.
If T(k) is polynomial in k then T(k)=$\Theta(k^\lambda)$. So the time complexity would be $\Theta(\frac{n}{numproc} * 2^{\lambda*\alpha} * \mu^\lambda)$.
Normal time complexity on application of full gene set would have been $\Theta(\frac{n^\lambda * \mu^\lambda}{\beta^\lambda})$. As $\alpha \ll logn$ so $2^{\lambda*\alpha} \ll n^\lambda$. Therefore the speedup obtained is $\frac{n^{\lambda-1} * numproc}{2^{\lambda*\alpha} * \beta^\lambda}$.

Case B: Unconstrained Algorithms

As the asymptotic time complexity of **inferUnconstrained** is same as **inferUnconstrained** so the analysis presented above holds in this case also. We obtain a speedup of $\frac{n^{\lambda-1}*numproc}{2^{\lambda*\alpha}*\beta^{\lambda}}$

5.8.2 Complete regulator set algorithms

inferComplete involves conversion of n GO based gene groups to *numgenes* regulator groups. So the average size of each regulator group turns out to be $\beta*\mu$. Time complexity would be $\frac{numgenes}{numproc} * T(\beta*\mu)$
If T(k) is polynomial in k then T(k)=$\Theta(k^{\lambda})$. So the time complexity would be $\frac{n*\mu}{numproc} * \beta^{\lambda} * \mu^{\lambda}$.
Normal time complexity on application of full gene set would have been $\Theta(\frac{n^{\lambda}*\mu^{\lambda}}{\beta^{\lambda}})$. As $\alpha \ll logn$ so $2^{\lambda*\alpha} \ll n^{\lambda}$. Therefore the speedup obtained is $\frac{n^{\lambda-1}}{\beta^{2*\lambda}*\mu} * numproc$. So in order to obtain a *speedup* > 1 *numproc* has to be greater than $\frac{\beta^{2*\lambda}*\mu}{n^{\lambda-1}}$. As $\beta \ll n$ so $numproc > \frac{\mu}{n}$ would always give good speedup.

5.9 Experimentation and Results

We use the Yeast cell cycle data set for our experiments. It has expression levels for 6178 genes at 80 time points. However, there are a lot of missing points in the input data set. We follow the following steps for pre-processing the input data set:

- Filter out genes whose variance lies in the bottom 20 percent.
- Fill the missing values.
- Detect periodicity; use the top 800 genes for further analysis.

As a result of preprocessing we have transformed the incomplete data set of 6178 genes to a complete data set of 800 periodic genes.

5.9.1 GO based grouping results

We group 6178 genes of the Yeast cell using the method described earlier in the chapter. We retain genes which belong to the set of 800 periodic genes obtained above and discard the remaining genes. Consider the Yeast cell gene network during cell cycle phase G1(Figure 5.5). We present the results of GO based grouping with respect to this network. There are 5 groups of co-regulated genes labeled as $G1$, $G2$,...$G5$ in the network diagram(Figure 5.5). These are the findings obtained by analysis of GO based gene groups:

- Group $G1$:
  - The three co-regulated genes are part of a GO based gene group.
  - 6 gene pairs are formed by 6 direct links between G1 and G2; each gene pair is part of some GO based gene group(FAR1 is not a part of any gene group).
- Group $G2$:
  4 gene pairs formed by 4 direct links between G2 and G3 belong to one GO based gene group.
- Group $G3$:
  As G3 is involved only in regulating other genes; involvement of genes belonging to G3 is implicitly discussed in results of gene groups G2 and G4 which are regulated by G3.
- Group $G4$:
  4 gene pairs are formed by 4 direct links between G3 and G4; each gene pair is part of some GO based gene group.
- Group $G5$:
  - The three co-regulated genes are part of a GO based gene group.
  - 6 gene pairs are formed by 6 direct links between G4 and G5; each gene pair is part of some GO based gene group.
  - 3 gene pairs are formed by 3 direct links between CDC20 and G5; each gene pair is part of some GO based gene group.

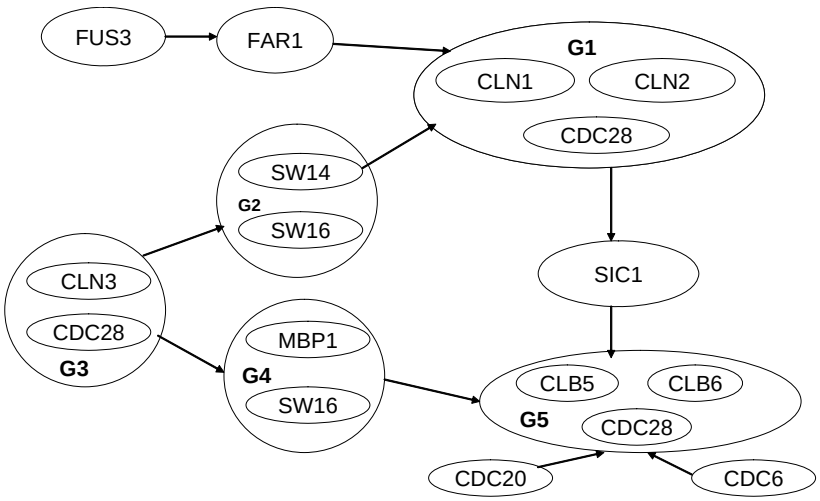


Figure 5.5: Gene Regulatory Network of Yeast cell during cell cycle phase G1.

– 3 gene pairs are formed by 3 direct links between CDC6 and G5; each gene pair is part of some GO based gene group.

We observe that all the gene pairs forming a direct link belong to some GO based gene group. This means that we can find all the regulators for a particular gene by taking the union of all the groups containing that gene (We missed out 2 genes FAR1 and FUS3 after the preprocessing stage, we manually included them in order to study the effectiveness of this grouping). Also, some GO based gene groups are more informative in the sense that they contain all the regulators for a particular gene.

5.9.2 Bayesian inference on GO based groups

The network shown in Figure 5.5 has 14 genes and 35 links. Kim et. al.([21]) have used this network to test their inference algorithm. We conducted two experiments which involves applying Bayesian inference over small gene groups and compare the results obtained with the biological network(Figure 5.5):

- *GO based gene groups, random groups, and Kim et. al.[21]*: We create a random group of the same size corresponding to each GO based gene group. We apply Bayesian inference over individual GO based gene groups and the corresponding random group. The results obtained are summarized in the following table:

Method	Number of links found	Percentage of links found
GO group	5	14.3
Random group	1	2.9
Kim et. al.[21]	3	9

  We observe that GO based gene groups certainly contain information which facilitates the inference task. It outperforms random grouping comprehensively. Also, Bayesian inference on GO based group lead to better performance than the method suggested by Kim et. al.([21]).

- *Regulator groups, random groups, and Kim et. al.[21]*: We construct a regulator group for each of the 800 periodic genes. Also, a same sized random group is created

corresponding to each regulator group. The results obtained are summarized in the following table:

Method	Number of links found	Percentage of links found
Regulator group	8	22.8
Random group	2	5.7
Kim et. al.[21]	3	9

We observe that regulator groups certainly contain information which facilitates the inference task. It outperforms random grouping comprehensively. Also, regulator groups lead to better performance than GO based gene groups. Also, regulator groups lead to better performance than the method suggested by Kim et. al.([21]).

Chapter 6

Network Inference Algorithm for Groups Formed Using GO

In the previous chapter we proposed a hierarchical framework for inferring gene regulatory networks. In this chapter we present an algorithm for gene network inference using the framework presented in the previous chapter. We also discuss the experiment carried out on the Yeast cell cycle data using this algorithm. The gene network inferred using this algorithm is compared with the biological gene network actually governing the Yeast cell cycle. It has been assumed that GO groups have been already formed by utilizing the following set of procedures described earlier: (a) ***Filtering genes***, (b) ***Filling missing values***, (c) ***Detecting periodicity***, (d) ***GO group formation using GO tree***.

6.1 Finding Regulators of a gene

We adopt an approach in which we identify all the regulators of a gene and try to model expression of the gene in terms of its identified regulators. A gene may belong to multiple GO groups that were formed using the GO tree's parent child relationships. Suppose a particular gene g belongs to a set of groups G_i, $1 <= i <= k$, all the regulators for the gene g belong to a union of these groups i.e. $\bigcup_{1<=i<=k} G_i$. For each gene g we form a regulator group R which is a union of the groups to which gene g belongs. Let g_i $(1 <= i <= n)$ be the genes belonging to the input set, then we form n regulator groups R_i where R_i is the

regulator group for gene g_i.

$$R_i = \bigcup_{1<=j<=k} G_{ij}$$

where G_{ij}, $1 <= j <= k$ are the k GO groups to which gene g_i belongs. Expression of a gene g_i is modeled in terms of genes belonging to its regulator group R_i.

6.2 Modeling Gene Expression

It is assumed that expression of a gene is affected only by its regulators. A linear model is used to model the expression of a gene in terms of its regulators. Suppose $R_i = \{g_j, 1 <= j <= n, j \neq i\}$ is the regulator group for gene g_i. Also, suppose there are M samples or time points of expression data available for each gene. The expression of a gene g_i is modeled by a function f_i, which has the form:

$$f_i(t) = \sum_{1<=j<=n, j\neq i} w_{ij} x_j(t-1), 1 <= t <= M-1$$

where $f_i(t)$ is the calculated expression value for gene g_i at time t, $x_j(t-1)$ is the expression value for gene g_j at time $t-1$ and w_{ij} is the measure of influence that gene g_j has on gene g_i. w_{ij} is positive for positive regulators g_j which tend to enhance the expression of the regulated gene g_i. Similarly, w_{ij} is negative for negative regulators g_j which tend to inhibit the expression of the regulated gene g_i. The magnitude of w_{ij} signifies the degree of influence that gene g_j has on gene g_i, higher the magnitude greater the influence. Our task is to learn the weights w_{ij} from regulators g_j to a gene g_i for all genes belonging to the input set.

6.3 Algorithm

In this section we present an algorithm for learning the weights of connections from regulators of a gene to that gene. Suppose $R_i = \{g_j, 1 <= j <= n, j \neq i\}$ is the regulator group for gene g_i and there are M samples or time points of expression data. Let $x_i(t)$ be the observed expression of gene g_i at time t. There are three possible cases related to number

of sample points and number of regulators for the gene: (a) $n < M-1$ (b) $n = M-1$ (c) $n > M-1$. We now discuss the approach pertaining to each of the three cases.

6.3.1 Case 1: $n < M-1$

As $n < M-1$ we have a situation in which we have more equations(M-1) than number of variables(n) as gene expression function f_i requires only n equations to solve for the n weights w_{ij}. A sliding window approach is adopted in which a data window of size n is extracted from the sample set of size M and is used to calculate a weight solution. Let us define w_{ij}^k to be the value of w_{ij} in the k^{th} subset of the sample set. The following procedure is followed to find $M-n$ weight solutions:

procedure weightSolve

- Initialize k to 1.

while($k <= M-n$){

- $$X = \begin{pmatrix} x_i(k) & x_i(k+1) & . & . & . & x_i(k+n-1) \end{pmatrix}$$
- $$\mathsf{D}= \begin{pmatrix} x_1(k-1) & x_1(k) & . & . & . & x_1(k+n-2) \\ x_2(k-1) & x_2(k) & . & . & . & x_2(k+n-2) \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ x_n(k-1) & x_n(k) & . & . & . & x_n(k+n-2) \end{pmatrix}$$
- Let $W_{ik} = [w_{i1}^k w_{i2}^k ... w_{in}^k]$ be the k^{th} solution, then

$$W_{ik} = XD^{-1}$$

- Increment k by 1.

}

By the end of above procedure we have $M - n$ weight solutions $W_{ik} = \{w_{ij}^k, 1 <= j <= n, j \neq i, 1 <= k <= M - n\}$. Now we try to evaluate the "goodness" of each of the $M - n$ solution by measuring it's performance over the entire sample set. Let f_i^k be the k^{th} function modeling gene expression for g_i using weights W_{ik}. Define E_k to be the error of function f_i^k over the entire sample set as :

$$E_k = \sum_{1<=t<=M-1} (f_i^k(t) - x_i(t))^2$$

where $f_i^k(t)$ is the predicted value of expression for gene g_i at time t and $x_i(t)$ is the observed value of expression for gene g_i at time t. We define a net weight over the entire sample set to be the weighted sum of the $M - n$ solutions with weight being assigned to each of the solutions inversely proportional to the error caused by that solution. The net weight solution W_i for gene g_i can be formulated as:

$$W_i = \frac{1}{\frac{1}{E_1} + \frac{1}{E_2} + ... + \frac{1}{E_{M-n}}} * (\frac{W_{i1}}{E_1} + \frac{W_{i2}}{E_2} + ... + \frac{W_{i(M-n)}}{E_{M-n}})$$

The weight solution $W_i = \{w_{ij}, 1 <= j <= n, j \neq i\}$ obtained is now used as a starting point for a gradient descent algorithm which converges to a local optimum. The procedure to be followed is described below:

procedure gradientDescent

$W_{current} = W_i$

$do\{$

$W_{prev} = W_{current}$

$for(1 <= j <= n, j \neq i)\{$

$\Delta w_{ij} = 0$

$for(1 <= t < M)\{$

$\Delta w_{ij} = \Delta w_{ij} + \eta * (f_i(t) - x_i(t)) * x_j(t)$, where $f_i(t)$ is the predicted expression value for gene g_i using W_{prev} and η is the learning rate.

}

}

$\Delta W = \{\Delta w_{i1}, \Delta w_{i2}, ..., \Delta w_{in}\}$

$W_{current} = W_{current} + \Delta W$

}

while($\mid W_{current} - W_{prev} \mid > \epsilon$), ϵ is the error limit for convergence.

6.3.2 Case 2: $n = M - 1$

In this case the number of variables is equal to the number of equations so we expect a unique solution if one exists.

$$X = \begin{pmatrix} x_i(1) & x_i(2) & . & . & . & x_i(M-1) \end{pmatrix}$$

$$D = \begin{pmatrix} x_1(0) & x_1(1) & . & . & . & x_1(M-2) \\ x_2(0) & x_2(1) & . & . & . & x_2(M-2) \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ x_2(0) & x_2(1) & . & . & . & x_n(M-2) \end{pmatrix}$$

Let $W_i = \{w_{i1}, w_{i2}, ..., w_{in}\}$ be the weight solution, then it can be solved for as:

$$W_i = XD^{-1}$$

Now, the W_i obtained can be used to invoke the procedure **gradientDescent** described in the previous case. As a result we obtain a weight solution which is a local optimum.

6.3.3 Case 3: $n > M - 1$

This is the most difficult of the three possible cases, we have more variables than equations. We assume that the number of regulators would not exceed the number of sample points

available which is quite a reasonable assumption looking at real biological networks. In this case, we try to iteratively filter out genes which have poor weights till the point when number of possible regulators goes below the number of sample points. After this point, we invoke the two procedures **weightSolve** and **gradientDescent** (described in case 1) sequentially. The procedure is described below:

procedure filterRegulators

- Initialize with random weights.
- Invoke procedure **gradientDescent** with the above weights to get refined weights.
- Eliminate poor weights from the weights obtained in the above step. Also, genes corresponding to those weights are removed from the list of potential regulators.
- if(number of regulators $<$ number of sample points)
  Invoke **weightSolve** and **gradientDescent** sequentially.
- else repeat the above steps

At the end of above analysis, a weight vector is obtained elements of which represent strength of regulation of the gene by its regulators. The above analysis finds strength of regulators for a particular gene g_i; the analysis is replicated across all the genes in the input set to find regulators and their regulation strengths for a gene.

6.4 Experimentation and Results

We use the Yeast cell cycle data set for our experiments. It has expression level for 6178 genes at 80 time points. However, there are a lot of missing points in the input data set. We follow the following steps for pre-processing the input data set:

- Filter out genes whose variance lies in the bottom 20 percent.

- Fill the missing values.
- Detect periodicity; use the top 800 genes for further analysis.

As a result of preprocessing we have transformed the incomplete data set of 6178 genes to a complete data set of 800 periodic genes. We group 6178 genes of the Yeast cell using the method described in the previous chapter. We retain genes which belong to the set of 800 periodic genes obtained above and discard the remaining genes. Now we form regulator groups for each of these 800 periodic genes as described in the previous chapter. We apply the proposed algorithm to each of these regulator groups in order to find potential regulators of every gene. After the weights have been learnt we choose the top k weights i.e., k regulators whose weights have a greater magnitude than the rest of the regulators. The table below compares the results with the biological network(Figure 5.5):

Method	Number of links found	Percentage of links found
Top 5	27	77.1
Top 7	31	88.5
Top 10	34	97.1
Kim et. al.[21]	3	9

We find that as 'k' increases the number of links identified goes up, at $k = 10$ almost all the links are identified. This algorithm outperforms Kim et. al.[21] by a significant margin showing the effectiveness of the algorithm.

Chapter 7

Conclusions and Future Work

7.0.1 Conclusions

We have proposed a novel hybrid approach to gene clustering using multiple information sources, i.e., Gene Ontology and Gene Expression Data. The proposed clustering algorithm was compared with k-means clustering, random clustering, and clustering without using GO. The algorithm performed significantly better than the three methods mentioned showing the effectiveness of the proposed approach. Also, better performance when using GO when compared to without it, strengthened the belief that GO can be used to increase the quality of clustering algorithms. The proposed algorithm is flexible enough to incorporate parallelism.

We have presented a GO based framework for hierarchical inference of gene regulatory networks. The framework supports a variety of inference algorithms. It divides the gene set into small "meaningful" groups based on GO information. GO based gene groups could capture most of the direct links present in the Yeast cell cycle phase G1. GO based gene grouping leads to improved performance of Bayesian inference (22 percent) compared to random grouping (4 percent). Also, it lead to better performance than the approach proposed by Kim et. al.[21] (9 percent) on the G1 phase regulatory network. Parallelism is inherent in the framework which brings down the time complexity of the inference algorithm. Also, we have shown that the size of search space goes down when using this framework compared to the case when using inference algorithm over the full

gene set. Right now, when we need to preserve the structure of the inferred network we dismantle the smaller network. This is a waste of the useful computation done in inferring the smaller network. There is a need to devise ways to save the wastage of useful computations.

We have presented a novel network inference algorithm which can infer gene regulatory networks from gene expression data. We modeled the expression of a gene to be a weighted sum of expression levels of it's regulators. We employed this algorithm in the GO based framework and tested it on the Yeast cell cycle data. We could infer more links (78 percent) than than proposed by Kim et. al.[21] (9 percent). in the cell cycle phase G1. This approach gave almost all the links when the number of top weights was increased to 10. The advantage of this approach lies not only in it's ability to detect links but also the speed at which links are inferred. As the framework is parallel and GO based grouping reduces the size of search space the algorithm can efficiently infer networks over large gene sets (in our case we used 800 genes).

7.0.2 Future Work

Some efforts can be put into parallelizing the algorithm which can help in efficient clustering of large gene sets. Also, it would be worthwhile to devise alternate distance measures and compare the performance with existing methods. Finding a good way for combining small networks would be an interesting extension to this framework. The proposed approach for inferring gene regulatory networks relies on GO heavily and cannot infer networks for genes which are not annotated at all in the GO DAG. It would be interesting to incorporate un-annotated nodes in the approach as that will help in annotating unknown genes. For grouping un-annotated genes, one can possibly use co-expression to find groups in which the un-annotated gene could fit in.

Bibliography

[1] Ashvin Agrawal, "Simulating and Inferring Biological Regulatory Networks", MTech Thesis Report, Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, 2004.

[2] Nora Speer, Christian Spieth, Andreas Zell, "Biological Cluster Validity Indices Based on the Gene Ontology", LNCS, pp. 429-439, 2005.

[3] Nadia Bolshakova, Francisco Azuajeb, Padraig Cunningham, "A knowledge-driven approach to cluster validity assessment", Bioinformatics, 21(10):2546-7, 2005.

[4] http://www.geneontology.org.

[5] Sung Guen Lee, Jung Uk Hur, Yang Seok Kim, "A Graph Theoretic modeling on GO space for biological interpretation of gene clusters", Bioinformatics, Vol 20, No 3, Pages 381-388, 2004.

[6] Ying Xu, Victor Olman, Dong Xu, "Clustering Gene Expression Data using a graph-theoretic approach: an application of minimum spanning tree", Bioinformatics, Vol 18, No 4, Pages 536-545, 2002.

[7] K.Y. Yeung, D. R. Haynor, W. L. Ruzzo, "Validating clustering for gene expression data", Bioinformatics, Vol 17, No 4, Pages 309-318, 2001.

[8] Michiel J.L. De Hoon, Seiya Imoto, Kazuo Kobayashi, Naotake Ogasawara, Satoru Miyano, "Inferring Gene Regulatory Networks from Time-Ordered Gene Expression Data of Bacillus Subtilis using Differential Equations", Pacific Symposium on Biocomputing, 17-28, 2003.

[9] IIya Schmulevich, Edward R. Dougherty, Seungchan Kim, Wei Zhang, "Probabilistic Boolean Networks: a rule-based uncertainity model for gene regulatory networks", Bioinformatics, Vol 18, No 2, 2002.

[10] Dirk Husmeier, "Sensitivity and specificity of inferring genetic regulatory interactions from microarray experiments with dynamic Bayesian networks", Bioinformatics, Vol 19, No 17, 2003.

[11] Eran Segal, Dana Peer, Aviv Regev, Daphne Koller, Nir Friedman, "Learning Module Networks", Journal of Machine Learning Research, pp 557-588, 2005.

[12] Shigeyuki Oba, Masa-aki Sato, Ichiro Takemasa, Morito Monden, Ken-ichi Matsubara, Shin Ishii, "A Bayesian missing value estimation method for gene expression profile data", Bioinformatics, Vol. 19 no. 16 2003.

[13] Earl F. Glinn, Arcady R. Mushegian, "Searching for Periodic Gene Expression Patterns Using Lomb-Scargle Periodograms", Bioinformatics, 22(3):310-316, 2006.

[14] T. Kohonen, Self-Organizing Maps, Springer, Berlin, 1997.

[15] Michael B. Eisen, Paul T. Spellman, Patrick O. Brown, David Botstein, "Cluster analysis and display of genome-wide expression patterns", Proc. Natl. Acad. Sci. USA 95, 14863-14867.

[16] J.A. Hartigan, "Clustering Algorithms", John Wiley and Sons, New York, 1975.

[17] S. Tavazoie, D. Hughes, M.J. Campbell, R.J. Cho, G.M. Church, "Systematic determination of genetic network architecture", Nature Genetics 22, 281-285, 1999.

[18] P. Tamayo, D. Slonim, J. Mesirov, Q. Zhu, S. Kitareewan, E. Dmitrovsky, E. Lander, T. Golub, "Interpreting patterns of gene expression with self-organizing maps: Methods and application to hematopoietic differentiation", Proc. Natl. Acad. Sci. USA 96, 2907-2912, 1999.

[19] Ying Xu, Victor Olman, Dong Xu, "Clustering gene expression data using a graph-theoretic approach: an application of minimum spanning trees", Bioinformatics, Vol 18, No 4, Pages 536-545, 2002.

[20] R. Shamir, R. Sharan, "CLICK: A clustering algorithm for gene expression analysis", In Proceedings of the 8^{th} International Conference on Intelligent Systems for Molecular Biology (ISMB '00), AAAI Press.

[21] Sim Yong Kim, Seiya Imoto, Satoru Miyano, "Dynamic bayesian network and nonparametric regression model for inferring gene networks", Genome Informatics, 13:371-372, 2002.

[22] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, "Introduction to Algorithms", Section 23.2: The algorithms of Kruskal and Prim, pp. 567574, Second Edition, MIT Press and McGraw-Hill, 2001.