168736

Integrating gene expression, protein interaction and protein domain data to improve gene expression clustering

by

Alper Tolga Kocataş

A Thesis Submitted to the
Graduate School of Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Electrical & Computer Engineering

168736

Koç University

February, 2005

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Alper Tolga Kocataş

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

[illegible]

Assist. Prof.Attila Gürsoy

[illegible]

Assist. Prof. Engin Erzin

[illegible]

Assist. Prof. Özlem Keskin

Date: ______________________

ABSTRACT

Cell activity is carried out by the interaction of various proteins. Complex interactions among proteins constitute molecular pathways, which are the mechanisms by which the living cells perform biological processes. Understanding pathways is crucial in revealing mechanisms of cellular activity, thus understanding the reasons behind genetic disorders. Domains, which are independent subunits of proteins, play an important role in protein interactions. The first method presented in this thesis uses association rule mining on protein interaction data to extract domain-domain interaction rules. The method was applied on a database of protein interactions, which resulted in rules, some of which are supported by biological knowledge.

Microarray expression data is another data source to study protein interactions. Most microarray data analysis methods are based on clustering genes that show similar expression patterns. However, clustering results often need to be refined, which can be done either by using biological expertise or by integrating other biological data. The second proposed method integrates domain-domain interaction rules with microarray data. The method is based on a previously developed probabilistic model which unifies protein interaction and microarray data. Results show that integrating domain-domain interaction rules produces gene clusters of higher coherence.

Finally, a parallelization of the second proposed method and its implementation, together with performance results are presented.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

BP	Belief Propagation
EM	Expectation Maximization
LHS	Left Hand Side
RHS	Right Hand Side
YSD	Yeast Stress Data
UPIM	Unified protein interaction model developed by Segal et al. [27].
UDIM	Our method, which integrates domain-domain interaction rules into UPIM.

Chapter 1

INTRODUCTION

Cell activity is carried out by the interactions of various proteins. Complex interactions among the proteins constitute molecular pathways, which are the mechanisms by which the living cells carry out biological processes. Understanding pathways is crucial in revealing mechanisms of cellular activity, thus understanding the reasons behind genetic disorders.

Normally, molecular pathways are identified using laboratory experiments, which are costly and prone to errors. Identifying an interaction between two proteins is an experimentally hard task. Identifying molecular pathways, which are carried out by a series of interactions among various proteins is even harder.

With the invention of the microarray technique, it became possible to screen the activity levels of genes in a cell under certain conditions. Outputs of microarray experiments are datasets that specify the activity levels of genes in the cells under different conditions. Recently, various methods have been developed to analyze outputs of microarray experiments. Most methods are based on the idea of clustering genes that show similar expression patterns. Clustering results often yield valuable information and they help us discover genes that are expressed together. They are useful in identifying the genes, which are responsible for a certain condition in the cell (for example, a genetic disorder).

On the other hand, results from clustering are often hard to interpret. This is mainly because the assumption that genes, which show similar expression profiles belong to the same molecular pathway is a relaxed one. Biological expertise is needed to interpret the clustering results. Alternatively, it is possible to refine the results computationally by incorporating other biological facts. For example, known protein interactions and protein functions can be used to improve the clustering results. This thesis focuses on integrating protein-protein interactions, domain-domain interaction rules with microarray expression profiles to refine the clustering results. Before introducing our approach, the following sections introduce background information about protein-protein interactions, domains of proteins and microarray experiments.

1.1 Background Information

1.1.1 Protein Interactions

Structural and chemical bindings between proteins are called *protein interactions*. As a result of an interaction between two proteins, one of the interacting partners may become active, phosphorylated, degraded, etc. (Figure 1.1). Interactions occur by binding of two proteins to each other via special sites, called *interfaces*.

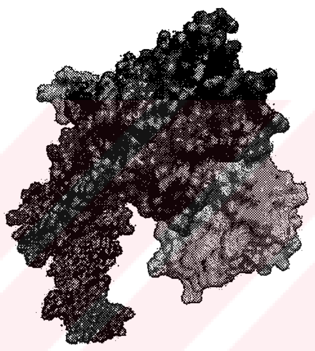


Figure 1.1: Proteins interact with structural binding [60].

The binding of two protein molecules is one of the basic mechanism of carrying signals through biological pathways in cells. Biological pathways can be defined as a chain of events in the cell that occur in a specific order and responsible for a biological process. Understanding the mechanisms behind protein interactions is crucial in deciphering the underlying complex biological pathways of cells.

Identifying protein interactions among the proteins of an organism can be done in various ways. There exists a number of laboratory experiments used for this task. *Yeast two-hybrid* method is the major high throughput technique, by which the interactions between proteins are identified. In this elegant technique, a special yeast protein complex that starts the protein synthesis of a reporter gene is divided into two parts: activating domain and the

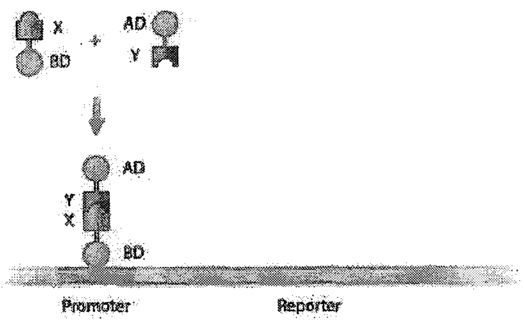


Figure 1.2: Yeast two-hybrid system [59].

binding domain. This special protein complex can not start the protein synthesis unless the two domains come together. Let protein X and protein Y be two proteins that are suspected to interact (Figure 1.2). Protein X is fused to the binding domain and protein Y to the activation domain. If proteins X and Y interact, then binding domain and activation domain of the protein complex come together and synthesis of the reporter gene is started. If they do not interact, reporter gene can not be synthesized. Finally, by observing the protein product of the reporter gene, interaction between proteins X and Y can be verified.

1.1.2 Domains of Proteins

As a result of the structural and biochemical evidence, traditional belief that proteins are simply polypeptides is not adequate to describe some proteins. It was indeed shown that different regions along a single polypeptide chain can act as independent units. When excised from the polypeptide chain, these regions can still fold correctly and can often exhibit their biological characteristics. These independent regions along the polypeptide chains are called the *domains* of proteins.

Domains are regarded as the basic units of proteins and are associated to protein function, folding, and evolution [16]. From the structural point of view, a domain is any part of a polypeptide which, when folded, creates its own hydrophobic core. From the functional genomics point of view, domains are regions, which are conserved in evolution during recombination events.

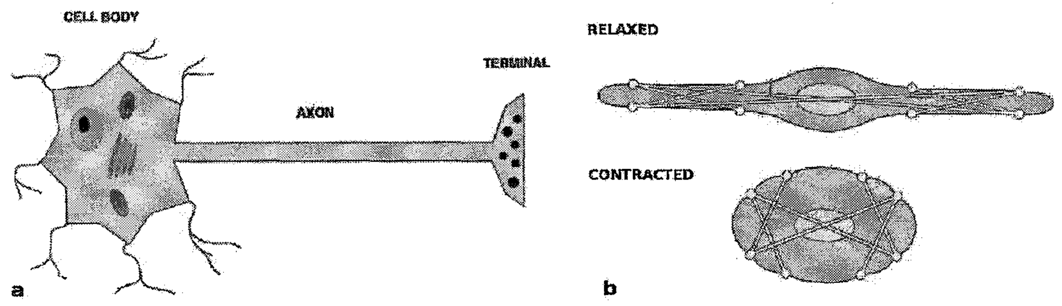


Figure 1.3: **a.** Nerve cell [57], **b.** Muscle cell [58].

It is important to decompose protein interactions into interactions between domains of proteins, because interactions between proteins often occur between domains.

1.1.3 Gene Expression

Genes are specific regions on DNA molecules. All the protein molecules in an organism are encoded by DNA and different genes are synthesized to different proteins. Encoding of the genes to proteins is carried out by a process known as *protein synthesis* (Figure 1.4). Proteins are synthesized in the cell in three main steps. First DNA is *transcribed* into mRNAs in the nucleus, and then mRNAs are *translated* into proteins in special structures called the ribosomes.

Any cell of an organism carries all of the genes that are specific to that organism[1]. However, in highly developed organisms such as animals, not all the cells look like the same and have the same function. This is because of different proteins synthesized in different cells and different biological pathways taking place. For example, muscle cells are specialized in contracting-relaxing and their shape is round. On the other hand, functions of nerve cells are different and they have a long axon and a head at the end of the axon (Figure 1.3).

Difference between these two types of cells is because of different *gene expression mechanisms*. This means that some genes, which give the nerve cell its characteristics, are kept *off* in the muscle cell, and they are kept *on* in the nerve cell. This regulation of the genes is called the gene expression mechanism.

[1]Except some special cells, which do not carry any genetic material.

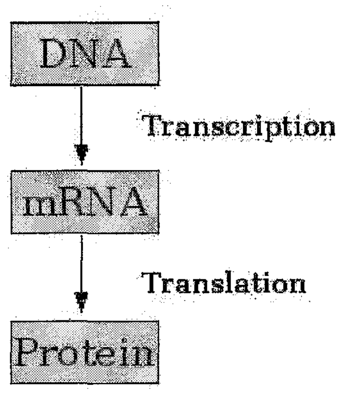


Figure 1.4: Steps of protein synthesis.

1.1.4 DNA Microarrays

Microarrays are used to monitor the gene expression profile of a cell at a given state of the cell by measuring the amount of mRNAs present in the cells. If a gene is expressed, then its mRNA should be present in the cell. Furthermore, concentration levels of the mRNAs of corresponding genes gives us an idea about how much the genes are expressed.

A DNA microarray consists of an orderly arrangement of DNA fragments representing the genes of an organism [55]. Basic idea behind microarrays is to put a part of the sequences of the genes of an organism on a special chip, then to *hybridize*[2] these sequences with pre-synthesized cDNA[3] sequences. When comparing two cells, mRNAs from the cells are converted to cDNAs, which are then labelled with green-fluorescing dye and red-fluorescing dye separately. Then, cDNAs from both cells are hybridized on the same chip. After scanning of the array with laser, the color of the cell, whose gene is expressed more, will be dominant at a given spot. A green spot means that the gene was expressed more in the first cell and a red spot means that the gene was expressed more in the second cell. A yellow spot, which is the result of same amount of red and green labeled cDNA means that the corresponding gene is expressed in both cells. Finally, a black spot is the result of no hybridization, which means that gene was not expressed in both cells. A more detailed

[2]**Hybridization:** The process by which cDNAs bind to the sequences on the microarray chip.

[3]**cDNA:** Single strand DNA molecule; complementary sequence of mRNA.

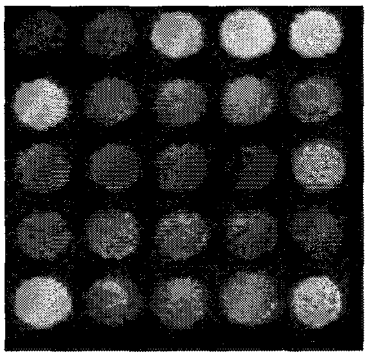


Figure 1.5: DNA Microarray image after hybridization and scanning [61].

Table 1.1: Microarray data format.

	Experiment 1	*Experiment 2*	...	*Experiment m*
Gene 1	0.1	-0.4	...	-0.6
Gene 2	0.6	0.7	...	0.2
.	...	...	...	...
Gene n	-0.9	-0.8	...	0.3

explanation of the procedure can be found in Appendix B.

Finally, light intensities of the spots, which represent the expression levels of genes are converted to numbers. Usually, data from multiple microarray experiments (as shown in Figure 1.5) are gathered into a single table. This table contains numbers representing the expression levels of the genes. Assume that m types of cells with n genes were examined. Then data for these experiments are stored as a matrix where the cell a_{ij} stores the expression level of ith gene in the jth sample. Table 1.1 shows an illustration of such data. Further normalizations and adjustments can be done over the raw data extracted from the chip. For example, expression values can be log-normalized because the light intensities can vary much.

1.2 Thesis Contribution

Integrating information from various sources into a common framework to infer new knowledge is a common paradigm in bioinformatics. This is because there is a high amount of biological data, but the connection between the data is rather loose. In order to extract valuable information from the data, often one needs to unite various sources of data. Data mining and machine learning methods proved successful to extract information from huge data sets however, heuristic methods are also used.

In this thesis, two methods focused on identifying protein-protein interactions and member proteins of molecular pathways from various kinds of biological data are explained. In Chapter 3, a method is presented, which uses a data mining technique called association rule mining to extract knowledge from protein-protein interaction data and domain decompositions of proteins. The ultimate aim of this method is to predict new protein-protein interactions by using the association rules derived from protein interaction data and domain data. The method outputs a set of rules that indicate the likelihood of interactions between proteins by looking at their domain decompositions. This method is first used as a primary method to predict interactions between proteins. Then, results from this study is used as input to the second method developed.

Identifying molecular pathways using gene expression data is another challenging task. In Chapter 4, a method (UDIM) is presented, which uses gene expression data, protein interaction data and domain decompositions of proteins to discover member proteins of biological pathways. Domain-domain interaction rules from the association rule mining method is integrated into an existing method by Segal et al. [27] (UPIM), which uses gene expression and protein interaction data. The motivation behind this integration is the assumption that when two proteins interact, interaction often takes place between domains of proteins and proteins often show similar gene expression profiles. This method is based on probabilistic models. Gene expression data is modeled by a probabilistic model, and then this model is integrated into a second probabilistic model that models the protein interaction data, which results in a unified model. Our contribution was to integrate the results of the association rule mining method into this framework.

Finally, a parallel implementation of UDIM is presented in Chapter 5. Because the model uses EM algorithm to learn model parameters, running until convergence takes considerable time in order of hours. Heavy computation is at the inference part of the algorithm, where belief propagation algorithm is executed to run approximate inference over the graphical model. A graph partitioning tool is used to partition the protein interaction network so that the load is balanced and the communication is minimized among the processors.

Chapter 2

RELATED WORK

Inferring knowledge from biological data is one of the challenges of bioinformatics. Some of the studies in the field focused on predicting protein-protein interactions. Others include studies on predicting biological pathways by identifying a network of protein interactions. There are also studies on extracting domain-domain interactions using protein-protein interactions. The methods used to solve problems are diverse. Data mining and machine learning methods proved successful to extract information from huge data sets however, heuristic methods were also used. This section gives a brief survey of these studies.

The task of protein-protein interaction prediction is studied very often. Oyama et al. [29] used association rule mining on protein-protein interaction data. They used the yeast protein-protein interaction data from the MIPS [14] and YPD [15] databases, along with different features of proteins like domains, motifs and localization data from other resources. Then, they used association rule mining to extract knowledge such as "if a protein has a set of features α and another protein has a set of features β, interaction between these proteins can occur according to some dependability parameters". Resulting association rules often agreed with the existing biological evidence.

Bock et al. [30] studied predicting protein-protein interactions directly from the primary structure of proteins and associated data. They used a diverse database of known protein-protein interactions (DIP [11]) to train a support vector machine (SVM) learning system for recognizing and predicting protein-protein interactions based solely on primary structure and associated physiochemical properties. Interaction data was divided into two, as training and test data. For each amino acid sequence of a protein complex, they assembled feature vectors from encoded representations of tabulated amino acid properties such as charge, hydrophobicity and surface tension. Their argument was: since sequentially proximal protein secondary structure elements are often co-located in three dimensional structures, sequential profile of these additional features (charge, surface tension) must also co-locate upon folding. Resulting system was successful up to 80% of inductive accuracy when tested against the test data.

Gomez et al. [33] also studied predicting protein-protein interactions from sequences of proteins. In their work, they describe an attraction-repulsion model, in which the interaction between a pair of proteins is represented as the sum of attractive and repulsive forces associated with small, domain- or motif-sized features along the length of each protein. Their model assumes that the evolutionary conserved features within each protein are responsible for interactions. This assumption of evolutionary redundancy implies that features that have predictive value for a portion of a large network will also have predictive value in the unknown part of the same network. Results of the study showed that the attraction-repulsion model with features extracted from Pfam performed slightly better than the SVM approach.

Ng et al. [32] also worked on inferring protein-protein interactions in *silico*. Their argument was that since protein-protein interactions involved physical interactions between protein domains, domain-domain interaction information can be useful for validating, annotating and even predicting protein-protein interactions. They present a method to computationally derive putative domain-domain interactions from multiple data sources, including protein interaction data, protein complexes and Rosetta Stone[1] sequences. They applied their resulting domain-domain interactions to validate and predict protein-protein interactions.

Jansen et al. [34] worked on integrating protein interaction information from various data sources to evaluate protein-protein interactions. They proposed a Bayesian approach for integrating interaction information that allows for the probabilistic combination of multiple data sets and demonstrated its application to yeast. Basic idea of this method was to assess each source of evidence for interactions by comparing it against samples of known positives and negatives, yielding a statistical reliability. Then, extrapolating genome-wide, they predicted the chance of possible interactions for every protein pair by combining each independent evidence source according to its reliability. They verified their predictions by comparing them against experimental data and TAP (tandem affinity purification) tagging experiments.

[1]Common sequences in interacting proteins.

On the other hand, some studies focused on information extraction from text sources. Ono et al. [31] present a method to extract protein-protein interactions from scientific literature. This method, which employs a protein name dictionary, surface clues on word patterns and simple part-of-speech rules, achieved high precision rates for yeast and *E. coli.* According to this method, first protein names are identified, which is done by using a dictionary of manually constructed protein names. The process of name identification is based on pattern matching between dictionary entries and words in sentences. Then, sentences which are reported to report information about two proteins are parsed with simple part-of-speech rules. Resulting information is evaluated by simple recall and precision measures.

Husmeier et at. [35] worked on applying Bayesian networks to infer protein interactions from gene expression data. Their aim was to measure the sensitivity of inferring genetic regulatory networks from microarray experiments with Bayesian networks. First, they simulated gene expression data from a realistic biological network. Then, they inferred interaction networks from this data in a reverse engineering approach, using Bayesian networks and Bayesian learning with Markov chain Monte Carlo. Results demonstrated how the network inference performance varies with the training set size, degree of inadequacy of prior assumptions, experimental sampling strategy and inclusion of further sequence-based information.

Nakaya et al. [36] worked on extracting correlated gene clusters by comparing multiple graphs derived from various genomic data sources. They first encoded the relationships among the genes according to each data type as a graphical structure, whose nodes correspond to genes. Their purpose was to find out gene groupings, which are correlated in all data sets. Gene sets, which show similarity in each graph would be biologically related with a high probability. Problem of finding correlated gene clusters was then formalized as a subgraph isomorphism problem, which required special graph algorithms. Using their method, they identified correlated gene clusters in *E. coli.* They also analyzed *S. cerevisiae* protein interactions and gene expression profiles with the same method, which yielded 16 putative interactions.

Vert et al. [37] worked on extracting features from microarray data, based on the knowledge of a graph which links genes known to participate in interactions of metabolic pathways. Motivation behind the study was that the biologically relevant features are likely to exhibit coherence with respect to a graph topology. Their algorithm involved encoding the graph and the set of gene expression profiles into kernel functions, and performing a generalized form of canonical correlation analysis. After extracting the features, they used them as an input to a state-of-the-art classifier and observed improvements with *S. cerevisiae* genes. They made the classifier use the vector of features instead of a vector of gene expression profiles to predict the functional class of a gene.

Yamanishi et al. [38] studied the extraction of correlated gene clusters from multiple genomic data such as gene expression, protein-protein interaction data and phylogenetic profiles. As a first step, they investigated the amount of correlation between these data sets. They extended the concept of correlation for non-vectorial data, which was possible by the use of generalized kernel canonical correlation analysis (KCCA). They proposed a method to extract groups of genes, which are responsible for detected correlations. In another study [43], they addressed the problem of inferring protein network of an organism from multiple data sources. Based on a variant of KCCA, they incorporated heterogeneous genomic data into a supervised learning framework to predict the protein interaction network of *S. cerevisiae.* They used four types of genomic data: gene expression profiles from microarray experiments, protein-protein interactions identified by yeast two-hybrid experiments, protein localizations in the cell and phylogenetic profiles of proteins. They made experiments with both supervised and unsupervised approaches. They showed that the supervised approach, in which a part of the information from the actual protein interaction network is used, yielded the best results.

Letovsky et al. [39] worked on predicting protein function from protein-protein interaction data using probabilistic methods. Their method assigned functions to proteins based on probabilistic analysis of graph neighborhoods in a protein-protein interaction network. They exploited the fact that neighboring nodes in the graph are more likely to share functions than the nodes which are not neighbors. They combined the protein interaction data with Gene Ontology (GO) [26] database to assign proteins different GO terms. They used a binomial model of local neighbor function labeling probability, combined with a Markov random field - belief propagation algorithm to assign functions to proteins. As a result, the method verified the known GO term assignments and it also assigned functions to 10% of the yeast proteins that lack GO term annotation.

Deng et al. [40] worked on combining protein function data and protein-protein interaction data to predict the function for unknown proteins. They also used a Markov random field model for the prediction of yeast protein functions based on multiple protein-protein interaction databases. Then they compared their prediction results with GO database to measure the efficiency of their method. They used a leave-one-out validation scheme for testing, which resulted in a 52% precision.

Nye et al. [41] studied statistical analysis of domains in interacting protein pairs. Their purpose was first, to combine protein-protein interaction data with individual domain decompositions of proteins in order to learn about the structure of interacting protein pairs; and second, to assess the evidence for physical contacts between domains in a statistical way. They assigned statistical values to pairs of domain super-families to measure the strength of evidence that domains from these super-families play role in interactions. Super-families of domains were created by grouping homologous domains into individual families. Using this method, they predicted domain-domain contacts in 705 protein interactions.

Analyzing gene expression data is a challenging task. Often, clustering methods, which group the genes that show similar expression profiles have proved successful in some extent.

Eisen et al. [44] used hierarchical clustering to group the co-expressed genes into same clusters. Their results proved that in *S. cerevisiae*, clustering gene expression data groups together genes of known similar function. Resulting argument was that the patterns seen in genome-wide expression experiments can be interpreted as indications of the status of cellular processes. Similar studies on clustering gene expression data followed [45, 46, 47] and different clustering methods have been widely applied to gene expression data. Self organizing maps [48] and Bayesian infinite Gaussian mixture models [49] are two examples. Going further, some studies focused on co-clustering different gene expression data sets together to fill the information gaps [50]. Supervised methods that use information resources other than the gene expression data were developed to aid the clustering process. Robinson et al. [51] and Adryan et al. [52] used the GO database [26] to refine clustering results. Adryan et al. [52] worked on co-clustering gene expression data and protein-protein interaction data. They adapted custom distance functions derived from both data sets and used hierarchical clustering as the clustering method. Segal et al. [27] also used the protein-protein interaction data to refine the results from gene expression data clustering. They developed a unified probabilistic model, which integrates gene expression data with protein interaction data. They trained this model using EM algorithm.

For the parallelization of belief propagation algorithm, Hong et al. [54] worked on a parallel approach on evidence combination on qualitative Markov trees. They transformed the Markov tree to a binary tree, and then they clustered the binary tree into different clusters so that the work load is balanced among the processors. As a result, they observed a speed-up of 2.3 on 4 processors. They utilized exact belief propagation, which is used to run inference over tree structured graphs.

Chapter 3

PREDICTING DOMAIN-DOMAIN INTERACTIONS WITH ASSOCIATION RULE MINING

With recent advances in modern biology and biotechnology, amount of biological data keeps accumulating in unprecedented speed. It is extremely important to analyze such a vast and diverse collection of data to understand biological processes. Data mining is one of the emerging areas to extract knowledge from large sets of data. In this section application of association rule mining to the prediction of protein-protein interactions is discussed.

In this study, relationships between *domains* of proteins are studied. Proteins can be characterized by combination of domains, and proteins interact with each other through their domains to carry out biological functions. Using databases of protein-protein interactions and databases of domain decompositions of proteins, it is possible to draw certain relationships. For example, if one can conclude rules such as "proteins having domain x generally interact with proteins having domain y", then this knowledge might help biologists to interpret biological processes better, and predict unknown interactions as well. Next sections give a summary of association rule mining technique, the method that is used to adapt the protein interaction and domain data to association rule mining and the results.

3.1 Association Rule Mining

Association rule mining is a data mining technique, which was proposed in [19]. It has emerged because of the need for extracting rules from the supermarket shopping basket data. With the use of barcode system, data about shopping baskets became very easy to collect and collection of shopping basket information accumulated in large databases. Such databases include a set of transactions. Every transaction represents the shopping basket of a customer, which simply includes a list of different items that are bought. Finding the frequent item sets that occur in these transactions can help to determine the items that are bought together by the customers. If such information is expressed in forms of rules, total number of items sold can be increased by taking appropriate actions (i.e. rearranging the products accordingly).

An example association rule can be "If a customer has bought milk and cheese, she often buys bread, too". This rule expresses the *association* between the items that the customer has bought. The rule states that considering some items a customer has bought, one can be confident within a limit that this customer has also bought another specific item.

There are two standard criteria that measure the dependability of association rules. These measures are the *support* and *confidence* of association rules.

3.1.1 Support of Association Rules

Let T be the set of all transactions in a database, e.g. let T be the set of all shopping basket records in a supermarket database. Support of an item set S is then the percentage of transactions in T, which contains S. In the supermarket example, this is the number of shopping baskets which contain $S = \{milk,\ cheese\}$ divided by the total number of shopping baskets in T. Let U denote the set of all transactions that contain S, then

$$support(S) = (|U|/|T|) * 100\% \tag{3.1}$$

where $|U|$ and $|T|$ are the number of elements in U and T respectively. Finally, support of an association rule is defined as the support of the item set that contains all the items that appear in the rule.

3.1.2 Confidence of Association Rules

Confidence of an association rule is the second measure to assess the quality of an association rule. For an association rule like $X,\ Y \rightarrow Z$, confidence is defined as the support of all items that appear in the rule divided by the support of the *if part* of the rule:

$$confidence\ (X,\ Y \rightarrow Z) = (support\ (\{X,\ Y,\ Z\})/support\ (\{X,\ Y\})) * 100\% \tag{3.2}$$

In other words, confidence of an association rule is the percentage of the number of cases where the rule is correct relative to the number of cases where the rule is applicable. For a rule like *milk, cheese* $\rightarrow$ *bread*, if a customer has bought milk and cheese, then the rule is applicable. If the customer has also bought bread, then the rule is correct for that transaction.

Below is a sample database of transactions, where $T = \{T_1, ..., T_5\}$ is the set of transactions and $I = \{i_1, ..., i_7\}$ is the set of distinct items that can appear in a typical transaction.

$T_1 = \{i_1,\ i_2,\ i_3,\ i_4\}$
$T_2 = \{i_1,\ i_2,\ i_3,\ i_5\}$
$T_3 = \{i_2,\ i_3,\ i_4\}$
$T_4 = \{i_1,\ i_2,\ i_5,\ i_7\}$
$T_5 = \{i_1,\ i_2,\ i_3\}$

An example association rule that can be extracted from this database is: $i_1,\ i_2 \rightarrow i_3$. For this rule, the support value is $(3 \div 5) * 100\% = 60\%$ and the confidence value is $(3 \div 4) * 100\% = 75\%$.

While mining databases of transactions for association rules, rules that have at least a certain probability of being true are searched. For this, minimum support and confidence parameters are given as input to the algorithm.

Finding association rules over a large database of transactions is time consuming. However, recent methods that use clever algorithms, efficient data structures and parallel algorithms cope well with the problem. Both parallel and sequential efficient algorithms were implemented.

3.2 Proposed Method

In this section our method is presented [28]. It adapts the protein-protein interaction data and domain data to be used in association rule mining. First of all, format of the protein-protein interaction data is not same as the supermarket basket data. Thus, layout of the data should be modified to make it suitable to be used in association rule mining.

A database of supermarket basket data simply includes a list of transactions, in which items per shopping basket are stored. However, interaction data is different. Every interaction involves two proteins, such as protein A and B. A typical protein interaction database entry is like: $A \Longleftrightarrow B$.

On the other side, there are domain databases, where the information is stored like: $A = \{d_i,\ d_j,\ d_k, ...\}$. This is to say that A is composed of the domains $\{d_i,\ d_j,\ d_k, ...\}$.

Using this second database, which stores the domain decompositions of the proteins, database of interacting proteins is converted to a format, where every protein is substituted with its set of domains. After all of the interacting proteins are decomposed into their domains, final protein interaction data entry looks like: $\{d_i,\ d_j,\ d_k...\} \iff \{d_l,\ d_m,\ d_n, ...\}$. In this data, LHS and RHS of the $\iff$ declaration correspond to the set of domains for proteins A and B, respectively.

Further, LHS and RHS of the above interactions should be collapsed into a single set in order to make it applicable for association rule mining. However, it would be impossible to interpret the output of the association rule miner if two sets are merged into one set. For instance, assuming that the LHS and RHS of proteins A and B are merged into one set, resulting transactions are such as: $T = \{d_i,\ d_j,\ d_k, ...d_l,\ d_m,\ d_n\}$. If interaction data is given to the association rule miner in this form, output rules will be like $d_i,\ d_j, ... \rightarrow d_k,\ d_l....$ The problem is that it is impossible to identify which domain refers to which side of an interaction in the original data. LHS and RHS information is eventually lost in such kind of approach, which is not desirable.

Solution is to put tags on the domains, which make us able to differentiate between LHS and RHS domains. After putting the tags, an interaction looks like: $\{d_{iL},\ d_{jL},\ d_{kL}...\} \iff \{d_{lR},\ d_{mR},\ d_{nR}, ...\}$. Now RHS and LHS of this interaction can be collapsed, which results in: $T = \{d_{iL},\ d_{jL},\ d_{kL}, ...d_{lR},\ d_{mR},\ d_{nR}\}$. Then, when interpreting rules like: $d_{iL},\ d_{jL}, ... \rightarrow d_{kL},\ d_{lR}...$, it can be determined if the domains in the LHS or RHS of the rule come from LHS or RHS of an interaction.

Once it is known which domain is a LHS domain and which is a RHS domain by looking at the elements in the resulting rules, a final modification is to add the reverse of every interaction to the set of interactions. This is because an interaction among two proteins is not a directional relation and $A \iff B$ is the same as saying $B \iff A$. Also, results of association rule miner can be biased in cases where some of the domains are often seen in LHS of the interactions.

When this final form of data is given as an input to the association rule miner, various types of rules are expected in the output. Let α and β denote the set of domains that appear in the LHS and RHS of an association rule, respectively. Then, rules expected are in the form: $\alpha \rightarrow \beta$. Rules which have only LHS domains on one side and only RHS domains on the other side and vice versa are the most meaningful ones, because they imply a rule that involve the opposite sides of an interaction. Let these rules be *type I* rules. Finding such rules enables us to make predictions like "if a protein has α domains and another protein has β domains, they will probably interact". These rules are described as:

$$\alpha \rightarrow \beta : \alpha \in LHS \ \&\ \beta \in RHS \text{ or } \alpha \in RHS \ \&\ \beta \in LHS$$

Another rule type can be the following (*type II*), where LHS and RHS of the interaction is composed of the same kind of domains (i.e all LHS domains):

$$\alpha \rightarrow \beta : \alpha \in LHS \ \&\ \beta \in LHS \text{ or } \alpha \in RHS \ \&\ \beta \in RHS$$

Third type (*type III*) is more complex. Namely, at least one side of rules is composed of different kinds of domains (i.e. left side of the rule is composed of LHS and RHS domains.) Following is an example rule of this type, which is called as the *composite rules.*:

$$\alpha \rightarrow \beta : \alpha \in LHS \ \&\ \beta \in LHS \cup RHS$$

3.3 Protein - Protein Interaction Data Source

Protein interaction data from the DIP (Database of Interacting Proteins) database [11] was used in this study. DIP is a database of experimentally determined protein-protein interactions. The core of the database is composed of three relational database tables: *node table* stores information about the proteins, *edge table* stores the interactions between proteins and a third table stores the details of the experiments that were carried out to identify the interactions. Most of the reported interactions were identified by yeast two-hybrid studies. The database stores protein interaction information of more than 107 types of organisms, but approximately 65% of these interactions are between yeast proteins.

3.4 Domain Data Source

Domain information from Pfam [17] database was used in this study. Pfam is a comprehensive collection of protein domains and families, with a range of established uses including

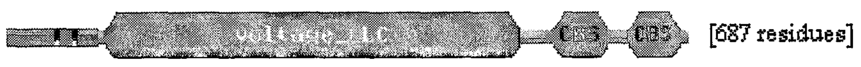


Figure 3.1: A typical example of a protein and its domains according to Pfam database.

genome annotation. Pfam contains curated multiple sequence alignments for each protein family, along with profile hidden Markov models for finding these domains in new sequences. For each protein family in Pfam, it is possible to look at multiple alignments, view protein-domain architectures, examine species distribution, follow links to other databases and view known protein structures. Figure 3.1 shows an example protein and its domain decomposition, as showed in Pfam.

Pfam has two main parts: Pfam-A and Pfam-B. Pfam-A is the curated part of Pfam, which consists of over 7459 protein families. All the protein families in Pfam-A has a structural function associated with them. To give Pfam more comprehensive sequence coverage, Pfam-B is automatically generated by using the ProDom [18] database, and is updated monthly. The functionality for viewing the protein-domain architectures of Pfam was used in this thesis, which is provided by a branch of Pfam called *Swisspfam*. Swisspfam lists the domain decompositions of SWISSPROT/TrEMBL proteins.

3.5 Results and Discussion

23910 interactions were given to the association rule miner and various experiments were carried out with varying support and confidence values. For the association rule miner, an implementation [21] of *Apriori* [20] is used. As the support and confidence values get more strict, number of rules found by the algorithm decreases. However, it is important to decide on which support and confidence value pair gives the closest match to a set of rules, which consists of valuable rules. For example, number of rules generated given a minimum support of 0.1% and a minimum confidence of 10% is around 130,000, which is very high. Indeed, when examined more closely, one can see that most of these rules are trivial and their dependability measures are low. In order to find more meaningful rules, one should increase the minimum support and confidence variables. However, then, there is the risk of missing some valuable, but not so frequent rules. Figure 3.2 shows the change in number of

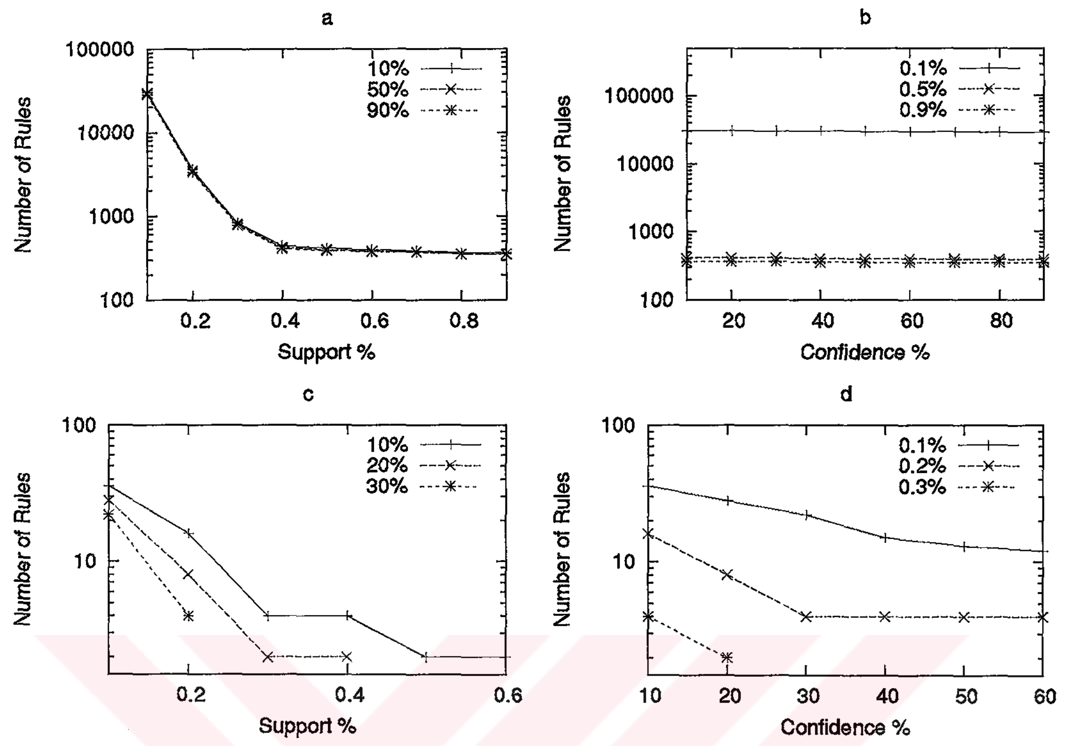


Figure 3.2: **a.** Number of Rules vs. support, with fixed confidence. **b.** Number of Rules vs. confidence, with fixed support. Change in confidence does not effect number of rules much. However, as the support parameter gets higher, number of rules goes down rapidly. **c.** Number of Rules vs. support, with fixed confidence for type I rules. **d.** Number of Rules vs. confidence, with fixed support for type I rules. For type I rules, effect of confidence parameter on number of rules gets more important. However, support parameter still causes more dramatic changes in number of rules.

rules with varying support and confidence parameters.

It is seen from the figures that the number of rules is gradually decreasing as the minimum support requirement is increased linearly. This observation is also true with the minimum confidence value, but the figures prove that the support is a more constraining variable than the confidence value. As the support increases linearly, a rapid decrease in the number of rules is observed.

Still in the search for optimum support and confidence pair for our data, other conclusions are drawn by looking at the rules. For example, looking at the number of composite rules may give us an idea about the dependability of the resulting rules. Composite rules, by their nature, contain both RHS and LHS domains in either side of the rule. Although composite rules were observed, it is hard to associate a biologically meaningful explanation for such rules. A careful look at the data, however, leads to an explanation. Assume that the rules $L_1 \rightarrow R_1$ and $L_1 \rightarrow R_1R_2$ have been already produced. Then the algorithm will produce $L_1R_1 \rightarrow R_2$ as well since the following is always true: $support(L_1) \geq support(L_1R_1) \geq support(L_1R_1R_2)$. It was observed that the number of composite rules also gradually decreases with increasing support and confidence requirements. In our experiments, it was observed that beyond 0.2% support and 20% confidence, these rules totally disappeared from the resulting rule sets.

Following are two of the rules that were detected by association rule mining method:

proteasome $\longrightarrow$ *proteasome*,	support: 0.3%, confidence: 46.5%
pkinase $\longrightarrow$ *cyclin*,	support: 0.2%, confidence: 34.0%

The domain named *proteasome* of the first rule is the Proteasome A-type and B-type domain annotated in Pfam. It is claimed that members of this domain form a large ring based complex, which verifies that proteins that contain this domain interact with each other. Second rule, on the other hand, is related with two distinct domains: cyclin, which is the N-terminal domain and pkinase, which is the protein kinase domain. It is mentioned in Pfam that cyclins regulate the cell division in eukaryotes and protein kinases form a complex with them. Tables 3.1 and 3.2 show some of the other rules that were discovered by the method.

Table 3.1: Some of the rules identified in association rule mining.

Rule	*Support*	*Confidence*
IL8 $\longrightarrow$ 7tm_1	0.3%	92.0%
SH3 $\longrightarrow$ SH3	0.3%	16.6%
HLH $\longrightarrow$ HLH	0.2%	55.9%
Glyco_transf_20 $\longrightarrow$ Glyco_transf_20	0.2%	54.5%
ig $\longrightarrow$ ig	0.2%	43.4%
WD40 $\longrightarrow$ cpn60_TCP1	0.2%	33.1%
AAA $\longrightarrow$ AAA	0.2%	19.1%
Glyco_transf_20 $\longrightarrow$ Trehalose_PPase	0.1%	83.3%
TNFR_c6 $\longrightarrow$ TNF	0.1%	65.2%
Sm $\longrightarrow$ Ribosomal_S28e	0.1%	53.6%

Table 3.2: Annotations of the domains involved in the rules listed in Table 3.1.

*P*fam Accession	*Domain Name*	*Annotation*
PF00001	7tm_1	7 transmembrane receptor (rhodopsin family)
PF00048	IL8	Small cytokines (intecrine/chemokine), interleukin-8 like
PF00018	SH3	SH3 domain
PF00982	Glyco_transf_20	Glycosyltransferase family 20
PF00010	HLH	Helix-loop-helix DNA-binding domain
PF00047	ig	Immunoglobulin domain
PF00400	WD40	WD domain, G-beta repeat 11-49 74-113 158-195
PF00118	cpn60_TCP1	TCP-1/cpn60 chaperonin family
PF00004	AAA	ATPase family associated with various cellular activities
PF01423	Sm	Sm protein
PF01200	Ribosomal_S28e	Ribosomal protein S28e
PF00229	TNF	TNF(Tumor Necrosis Factor) family
PF00020	TNFR_c6	TNFR/NGFR cysteine-rich region
PF02358	Trehalose_PPase	Trehalose-phosphatase

Chapter 4

INTEGRATING DOMAIN-DOMAIN INTERACTIONS WITH MICROARRAY DATA

In order to refine the gene expression clustering results, domain-domain interaction rules from the study in Chapter 3 is integrated into a method that takes gene expression and protein interaction data as input. The motivation behind this integration is based on protein interactions. When two proteins interact, usually there is a physical interaction between their domains. If two proteins have domains that are likely to interact, they are also expected to show similar gene expression profiles. This additional information is used to improve on the results of gene expression data clustering.

Integrating domain-domain interaction prediction rules with microarray data is based on UPIM [27], which integrates protein interaction and microarray data. Next section first describes this method. Then, our method (UDIM), which integrates domain-domain interaction rules into UPIM, is presented.

4.1 Gene Expression - Protein Interaction Model

This section introduces UPIM. The method relies on the assumption that genes which are the members of the same molecular pathway often show similar expression profiles and their protein products often interact. The model unifies two independent probabilistic models; one for gene expression data and the other for protein-protein interaction data.

The unified model defines a joint probability distribution over n genes: $G = \{g_1, ..., g_n\}$. The model assumes that every gene belongs to one of the k pathways. Every g_i is associated with a discrete random variable $g_i.C \in \{1, ..., k\}$. $g_i.C = j$ means that gene g_i is assigned to pathway j. Since it is not observed which gene is assigned to which pathway in the data, variables $g_1.C, ..., g_n.C$ are hidden variables and determining values of them are one of the main goals of this method.

4.1.1 Gene Expression Model

For modeling the gene expression data, Naive Bayes classifier is used. Naive Bayes classifiers are mostly used in classification problems such as text classification. In Naive Bayes classifier, there is a set of instances with several attributes and each instance belongs to one of the disjoint classes. Detailed information about Naive Bayes classifiers can be found in Appendix C.

For the case of gene expression data, instances are the genes and attributes are the experiments. There are n genes and a gene has m continuous valued attributes, which are the expression levels of that gene in m experiments. Attribute set of a gene is denoted by the vector $g.\mathbf{E} = \{g.E_1, ..., g.E_m\}$, where $g.E_j$ represents the gene expression level of the gene in jth experiment. Hypotheses searched for are the separation of n genes into k disjoint classes.

Naive Bayes model defines a joint distribution over pathway assignment variables $g.C$ and experiment values $g.\mathbf{E}$ as the following (assuming that the expression levels of genes in different experiments are conditionally independent given the pathway assignments):

$$P(g.C|g.E_1, ..., g.E_m) = P(g.C) \prod_{j=1}^{m} P(g.E_j|g.C). \tag{4.1}$$

Random variable $g.C$ is distributed as a multinomial distribution which is parameterized by the vector $\theta = \{\theta_1, ..., \theta_k\}$ where all θ_i's sum to 1 and $P(g.C = p) = \theta_p$.

Each conditional probability distribution (CPDs) $P(g.E_j|g.C)$ is modeled by a normal distribution $N(\mu_{pj}, \sigma^2_{pj})$, where μ_{pj} is the mean of the normal distribution given that the gene is assigned to pathway p and the experiment modeled is the jth experiment.

4.1.2 Protein Interaction Model

Protein interaction model is built so that the interacting proteins assigned to the same pathway. The model uses a probabilistic model which is known as *Markov networks* [5]. Detailed information about Markov networks can be found in Appendix E.

Let $V = \{V_1, ..., V_n\}$ be a set of discrete random variables, then a Markov network defines a joint distribution $P(V)$. The model can be represented with an undirected graph whose nodes correspond to variables in V. Edges in the graph, which are denoted with ε represent direct probabilistic relationships between the variables. Each variable V_i in V is assumed to be a clique and is associated with a clique potential $\phi_i(V_i)$. Likewise, each edge $[V_i - V_j] \in \varepsilon$ is associated with a non-negative *compatibility potential* $\phi_{i,j}(V_i, V_j)$. The joint distribution is then defined as follows:

$$P(V_1, ..., V_n) = \frac{1}{Z}\prod_{i=1}^{n}\phi_i(V_i) \prod_{[V_i - V_j]\in\varepsilon} \phi_{i,j}(V_i, V_j), \tag{4.2}$$

where Z is the normalization factor required to make the distribution sum to 1. Intuitively speaking, $\phi_i(V_i)$ is the probability of different assignments to the variable, ignoring the underlying relational connections (edges) between the variables (nodes) in V. This can also be called the *local belief* at node V_i. On the other hand $\phi_{i,j}(V_i, V_j)$ specifies how compatible are the two assignments of V_i, V_j are.

In the case of protein interactions, nodes of the graph correspond to proteins and the edges correspond to the interactions between the proteins. Every protein has an associated random variable V_i, which is the discrete random variable that identifies the pathway assignment of *ith* protein. Since for every protein, the V_i random variable comes from the same type of distribution, it suffices to define $\phi_1(V_i)$ and $\phi_2(V_i, V_j)$. Furthermore, since proteins correspond to genes, variables in V correspond to the pathway assignment variables $g_1.C, ..., g_n.C$ of the genes.

The model implements the intuition that if two genes g_i and g_j interact, it is likely that they play role in the same biological process, so they are assigned to the same pathway. The compatibility potential function $\phi_2(g_i.C = p, g_j.C = q)$ promotes the cases that the two arguments (genes) are assigned to the same pathway. So, its value is higher if $p = q$ and lower otherwise. Compatibility potential function is defined as follows:

$$\phi_2\ (g_i.C = p, g_j.C = q) = \left\{ \begin{array}{cc} \alpha & if\ p = q, \\ 1 & otherwise \end{array} \right\}, \tag{4.3}$$

where $[g_i - g_j] \in \varepsilon$ and $\alpha > 1$. When $\alpha = 1$, the compatibility potential function has no effect over the joint distribution $P(g_1.C, ..., g_n.C)$. As it gets greater, the influence will be greater and the model will promote the cases where interacting proteins are in the same pathway.

4.1.3 Unified Model

Unified model is constructed by integrating the gene expression model into protein interaction model. This is accomplished by using the $g_i.C$ variables that are common to both models. Note that given the expression data and conditional probability distributions (CPDs) in the gene expression model, one can calculate the probability of a single gene to belong to a specific pathway, independent from the other genes. This probability is used as the node potential function in the protein interaction model.

Resulting model is a partially directed model with interactions specifying the undirected edges and the gene expression values specifying the directed edges. In this model, there are $m+1$ random variables for each of the genes: one $g_i.C$ variable, plus m random variables for $g_i.E_j$, where $j = \{1, ..., m\}$. The variable $g_i.C$ is controlled by a multinomial distribution with parameters $\theta = \{\theta_1, ..., \theta_k\}$. Additionally, there are $k * m$ CPDs $P(g_i.E_j|g_i.C = p)$, which are controlled by normal distributions $N(\mu_{pj}, \sigma^2_{pj})$. Final component of the model is the compatibility potential function $\phi_2(g_i.C = p, g_j.C = q)$, which is only defined for the pair of genes that are connected via an edge in the interaction network. The model defines a joint distribution over the random variables as follows:

$$\begin{aligned} P(\mathbf{G.C}, \mathbf{G.E}|\varepsilon) = \\ & \frac{1}{Z}\left(\prod_{i=1}^{n} P(g_i.C) \prod_{j=1}^{m} P(g_i.E_j|g_i.C)\right) \\ & \cdot \left(\prod_{[g_i-g_j]\in\varepsilon} \phi_2(g_i.C, g_j.C)\right), \end{aligned} \tag{4.4}$$

where Z is the normalization constant as to make the distribution sum to 1 and ε is the interactions between genes.

Finally, one should remember that protein interaction model trivially assigns all proteins to same pathway as long as the protein interaction network is connected. This is because the model promotes the case that all the interacting proteins are assigned to the same pathway. Figure 4.1 shows three different pathway assignments. If gene expression data is not incorporated, then the situation in Figure 4.1.b is the most probable one. However, once the gene expression data is integrated, this case is no more the most probable one.

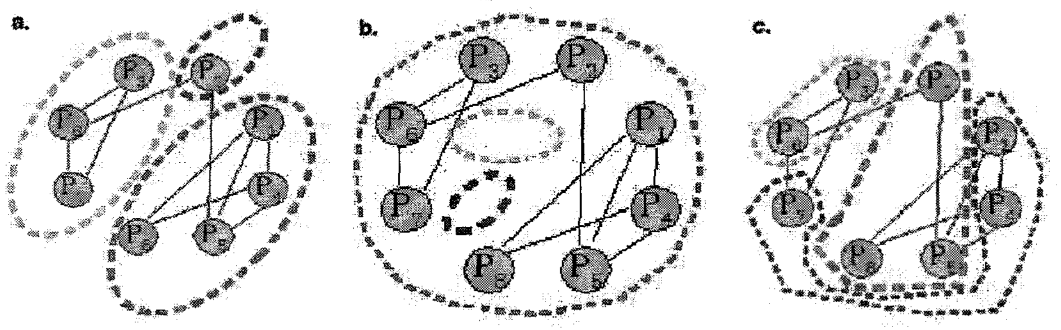


Figure 4.1: Different assignments of proteins to pathways. If the interaction model is used without integrating gene expression data, the situation in b is most probable, and the situation c in is least probable.

4.1.4 Training the Model

After the model is set, it is trained with the gene expression and protein interaction data to learn the values of parameters. The major issue in training is that the $g_i.C$ variables are hidden variables. This is because there is no observation in the data of which gene is assigned to which pathway. Values of these parameters are learned at the same time with the values of observed random variables.

Following parameters are to be learned: $\mathbf{G.C} = \{g_1.C, ..., g_n.C\}$, $N(\mu_{pj}, \sigma^2_{pj})$, and multinomial coefficients $\theta = \{\theta_1, ..., \theta_k\}$. There is an additional parameter α, which parameterizes the compatibility potential function, however since it is given to the model directly, it is not learned in the training process. Let all this parameters be in set Θ. Besides, let D be the data set that contains the protein interactions ε, and the gene expression data.

Since the assignments of $\mathbf{G.C}$ variables to pathways are not observed in the data, likelihood function $P(D|\Theta)$ does not have a unique local maxima. In case of unobserved variables, there are various ways to learn model parameters. Gradient descent, Monte-Carlo simulations and EM algorithm are some of these methods. Here the EM algorithm [9] is chosen. Detailed information about EM algorithm is given in Appendix D.

EM algorithm iterates in two steps. In the estimation step (E-Step), it uses the current estimate of the parameters to compute the distribution over hidden variables, given observed data. In this case, probabilistic inference is used to calculate marginal probabilities

$P(\mathbf{G}.\mathbf{C}|D,\Theta^{(t-1)})$. Since it is infeasible to calculate marginal probabilities exactly, *belief propagation*, which is an approximate inference algorithm, is used.

In the maximization step (M-Step), model parameters are re-calculated to maximize the likelihood of the data. Following formulas are used in this step. Let

$$\begin{aligned} q(g,p) = P(g.C = p|D,\Theta(t-1)) \quad &; \quad \overline{N_p} = \sum_{g\in\mathbf{G}} q(g,p)\ ; \\ \overline{x^1}_{pj} = \sum_{g\in\mathbf{G}} q(g,p).g.e_j \quad &; \quad \overline{x^2}_{pj} = \sum_{g\in\mathbf{G}} q(g,p).g.e_j^2. \end{aligned} \tag{4.5}$$

Then the parameters are re-estimated according to the equations below:

$$\theta_p = \frac{\overline{N_p}}{\sum_{p'=1}^{k} \overline{N_{p'}}} \quad ; \quad \mu_{pj} = \frac{\overline{x^1}_{pj}}{\overline{N_p}} \quad ; \quad \sigma_{pj}^2 = \frac{\overline{x^2}_{pj}}{\overline{N_p}} - \mu_{pj}^2. \tag{4.6}$$

Belief Propagation Algorithm

BP algorithm is based on passing messages among the nodes of a network. In BP algorithm, there are variables such as $m_{ij}(x_j)$, which can be defined as the message going from node i to node j, about what state node j should be in. This message is a vector of the same dimensionality as the random variable x_j. In this vector, each dimension measures the likelihood of node x_j being in a different state. Belief at node i is calculated as the product of messages coming to that node from its neighbors and the local evidence $\phi_i(x_i)$ on that node:

$$b_i(x_i) = k\phi_i(x_i) \prod_{j\in N(i)} m_{ji}(x_i). \tag{4.7}$$

Messages are calculated by the following message update rules:

$$m_{ij}(x_j) \leftarrow \sum_{x_i} \phi_i(x_i)\psi_{ij}(x_i,x_j) \prod_{k\in N(i)\backslash j} m_{ki}(x_i), \tag{4.8}$$

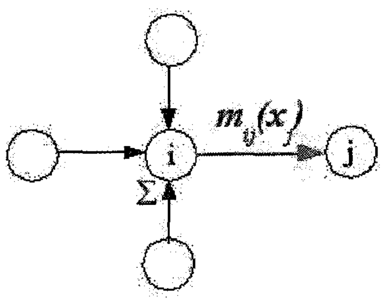


Figure 4.2: Graphical illustration of message update rule: $m_{ij}(x_j) \leftarrow \sum_{x_i} \phi_i(x_i)\psi_{ij}(x_i, x_j) \prod_{k \in N(i)\backslash j} m_{ki}(x_i)$. To calculate the message from node i to node j, messages coming into node i are multiplied with the local belief at node i and summed over all states of x_i.

where $k \in N(i)\backslash j$ means neighbors of node i except node j. Figure 4.2 illustrates this calculation scheme. In our setting of protein interactions, $\psi_{i,j}$ corresponds to the edge compatibility function $\phi_2(g_i.C, g_j.C)$ and $b_i(x_i)$ corresponds to $P(g_i.C|g.\mathrm{E})$.

Appendix E covers a proof that the message update rules give the beliefs that are exact if there are no loops in the graph [12].

Loopy Belief Propagation Algorithm

BP algorithm, as explained above works only for undirected graphs with no loops. Beliefs are calculated by propagating messages from a root node down to leaves in a single iteration. However, in a graph with loops, BP is not guaranteed to converge. On the other hand, it was experimentally proven successful in some applications [13] and was concluded that BP did converge to good approximations in many cases. Since then, BP has successfully been applied to many problems in machine learning and computer vision. Algorithm 1 illustrates this scheme.

4.2 Integrating Domain-domain Interaction Rules into the UPIM

This section introduces our method (UDIM), which integrates domain-domain interaction rules into UPIM. In case of the protein-protein interaction and gene expression data model, there is a network of proteins which can be represented as an undirected graph. In order

Algorithm 1 *LoopyBP()*

```
1: Let $\varepsilon$ be the set of edges in the undirected belief network
2: Initialize the messages $m_{ij}^{0}(x_j)$ for all $(i,j) \in \varepsilon$
3: for $t = 1, 2, \ldots$ until convergence do
4:     Update $m_{ij}^{t}(x_j)$ by using $m_{ki}^{t-1}(x_i)$ for all $(i,j) \in \varepsilon$ and $k \in N(i)$.
5: end for
6: Update all beliefs $b_i(x_i)$ using equation 4.7
```

to integrate the domain-domain interaction rules data into the model, this graph structure is altered. However, since interaction between two domains is defined by association rules, which depend on support and confidence parameters, edges in the interaction network are assigned weights by using a weight calculation algorithm.

Weights in the new interaction network depend on the number of domain-domain interaction rules among the proteins and the support parameters of the rules that imply these interactions.

Let A and B be two proteins. Edge weight among these proteins are determined using Algorithm 2.

Algorithm 2 *CALCULATE WEIGHT (P_A, P_B)*

```
1: Let $D_A$ and $D_B$ be the sets of domains of proteins $P_A$ and $P_B$, respectively.
2: Let $\Lambda$ be the database of association rules
3: Let $W$ be the edge weight
4: $W \Leftarrow 0$
5: for all $d_A \in D_A$ do
6:     for all $s_B \in$ subsets of $D_B$ do
7:         Let $R$ be the association rule such that $R$: $d_A \rightarrow s_B$
8:         if $\Lambda$ contains $R$ then
9:             $W \Leftarrow W +$ Support of $R$
10:        end if
11:    end for
12: end for
```

The algorithm takes two proteins P_A and P_B as an argument. Then, it generates all possible association rules that can be created by using domains of P_A and P_B. Since the association rules are in the form: $d_i \rightarrow d_j, d_k, ...$ where d_i, d_j, d_k are particular domains, domains of P_A are taken as the single domain (d_i) on LHS and domains of P_B are used to generate all possible subsets that can be taken as RHS ($d_j, d_k, ...$) of the association rule. Note that this algorithm only considers the association rules where LHS of the rule comes from P_A. So, the algorithm is run two times, exchanging order of the arguments P_A and P_B. Finally, weights that come from this two calculations are added up and assigned to the weight of the edge between P_A and P_B.

Weight calculation is done for all pairs of proteins and the pairs whose weights are assigned a non-zero value are connected with an edge and assigned the calculated weight. Algorithm 3 summarizes this procedure.

Algorithm 3 *CREATE INTERACTION GRAPH (Γ : list of proteins)*

```
1: Let $\varepsilon$ be data structure that stores the lists of interactions among the proteins
2: $\varepsilon \Leftarrow \emptyset$
3: for $i = 0; i < |\Gamma|; i++$ do
4:   Let $P_i$ be the $ith$ protein in $\Gamma$
5:   for $j = i+1; j < |\Gamma|; j++$ do
6:     Let $P_j$ be the $jth$ protein in $\Gamma$
7:     $W \Leftarrow$
       CALCULATE WEIGHT ($P_i$, $P_j$) + CALCULATE WEIGHT ($P_j$, $P_i$)
8:     if $W > 0$ then
9:       insert $\varepsilon$ edge $[P_i - P_j]$ with weight $W$
10:    end if
11:  end for
12: end for
```

After running the *CREATE INTERACTION GRAPH* algorithm, a new interaction graph of weighted edges is obtained.

Existing model is modified to accept this weighted graph. The modification is done to the edge compatibility function of the interaction model $\Phi_{i,j}$ (Equation 4.3). Original interaction model used a fixed parameter α for specifying the $\Phi_{i,j}$ function, which generated the same outputs for all protein interactions. If both interacting proteins were placed in the same pathway, output of $\Phi_{i,j}$ was α and if they are placed in different pathways, output was 1.

Here, this function is modified so that it outputs different values for the interactions between different pairs of proteins. For this, weights calculated by the *CALCULATE WEIGHT* (Algorithm 2) are used. Also an alternative protein interaction graph, which is constructed by using the *CREATE INTERACTION GRAPH* (Algorithm 3) is used instead of using the original protein-protein interaction data.

4.3 Results

Experiments were made with three methods: First, k-means algorithm was used to cluster gene expression data. Then, UPIM was run on the same gene expression data with integrating protein-protein interaction data. Finally, UDIM, which integrates domain-domain interaction rules was run. Resulting clusters were compared using different metrics.

The same data set that was used in [27] is used in the experiments. This is the *S. cerevisiae* gene expression dataset of 173 microarrays, measuring the responses of cells to various stress conditions [1]. For the protein interaction data, DIP [11] dataset was used.

Experiments were carried out for 3587 genes, fixing the number of pathways to be learned to 60, which was the number of pathways aimed in [27]. The reason for choosing the number of pathways as 60 is to be able to compare the results from this study and the results from UPIM. Although DIP database gained in volume recently, an older version of DIP was used because of the same reason. This older version contained 10142 interactions among the 3587 genes that was studied.

Quality of the clusters was measured by looking at p-values of the GO (Gene Ontology) terms that annotate the genes in the resulting clusters. GO [26] is a database, which provides a structured vocabulary used to describe the biological features of gene products. The database has three different vocabulary sets related with the *molecular function* of gene products, *biological process* in which the gene products participate and *cellular compartment* where the gene products are located. Between these three vocabulary sets, largest one is the *biological process*, which is used in this study to assess the quality of resulting clusters.

The Perl module "GO Term Finder" provided by SGD [25] was used to measure the quality of gene clusters. This module finds all the GO terms that annotate the genes in a given gene cluster. It can also report the GO terms that match the given genes with a p-value, which is below a specified threshold.

P-values provide information about the amount of chance, by which the genes listed in a gene cluster, are annotated with particular GO terms. They vary in the range $[0, 1]$. The lower the p-value for a GO term is, the more important information it provides is. In other words, if the p-value corresponding a GO term is lower, then it means that this match is a significant one, because it is not statistically common. Appendix F covers the method used to compute the p-values. In all of the experiments, minimum p-value was chosen to be 0.05.

Clusters were compared by using the p-values in a number of ways. First way is to find the number of GO terms, that match the genes in each cluster with a p-value below 0.05 and to sum this value for all clusters. Results of this measurements are given in Figure 4.3. Values of this calculation for k-means clustering, UPIM and UDIM was 688, 802 and 825, respectively. Last two numbers are from the results of the experiments with α parameter, which gave the best results. Both UPIM and UDIM performed better than k-means. UDIM performed slightly better than UPIM with higher number of GO terms. Figure 4.4 shows the average number of protein interactions between the genes of clusters for both methods. Number of interactions is rapidly increasing with α in UPIM, whereas it is gradually increasing with α in UDIM. However, the quality of clusters, when measured using p-values, are comparable. This shows that average interaction density is not closely related with biological significance of clusters.

As a second way, best p-value that can be obtained in all the clusters for the GO terms that match the genes in the resulting clusters are extracted. Then, to compare results from two studies, these p-values are compared. It was observed that UDIM performed better than k-means (Figure 4.5). However, UPIM performed better than UDIM, when compared with this measurement (Figure 4.6).

Finally, negative p-values of the top three clusters (which have most number of GO terms matched to their genes below the p-value threshold) are calculated. Then p-values from two studies are compared only for the GO terms that match top clusters. UDIM performed slightly better than UPIM when compared with this metric (Figure 4.7).

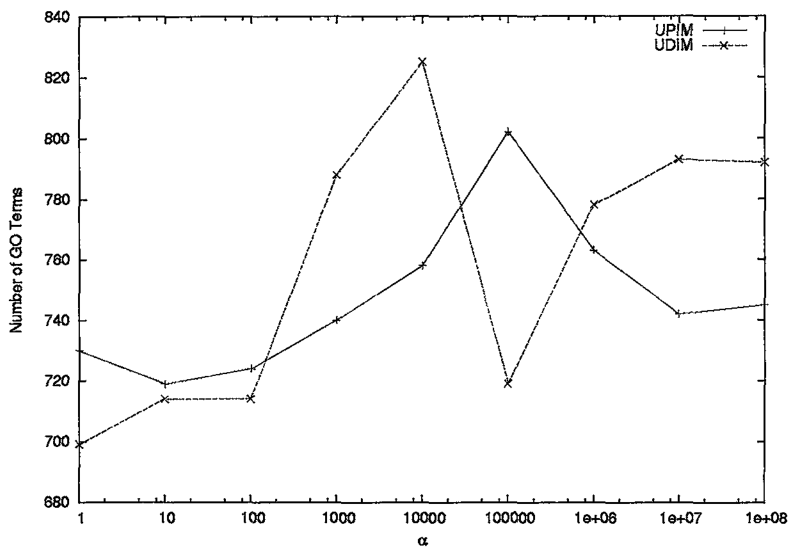


Figure 4.3: Change of number of GO terms below the p-value threshold versus α parameter for UDIM and UPIM. For the α value that gives best results, UDIM performs slightly better.

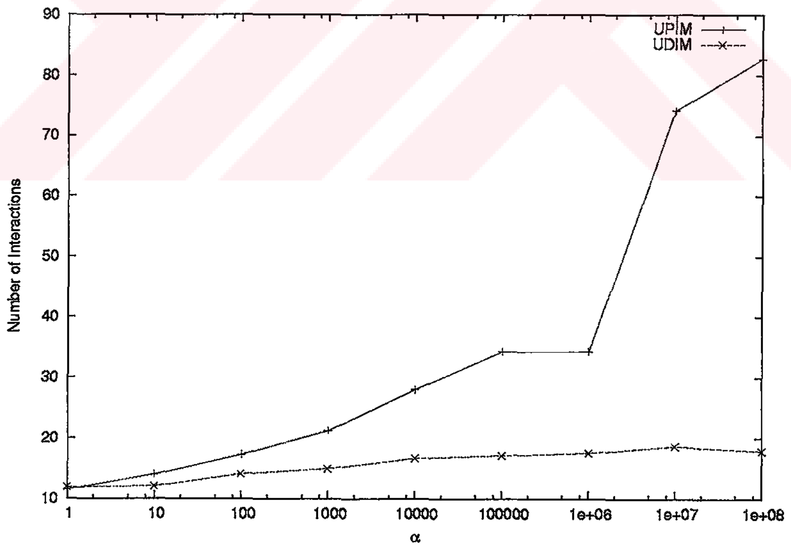


Figure 4.4: Change of average number of interactions among the genes in the resulting clusters versus α parameter for UDIM and UPIM. As the parameter α increases, interaction density is affected more in UPIM.

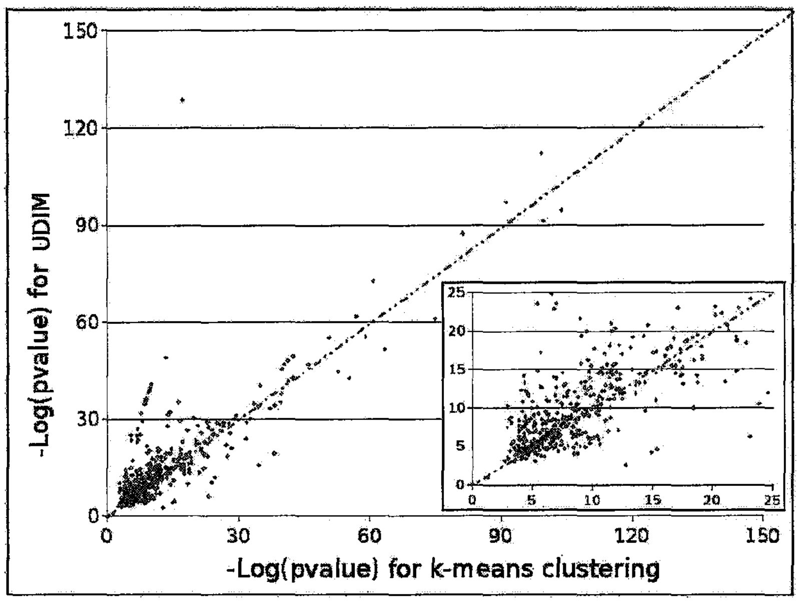


Figure 4.5: Negative log p-value calculated for best p-values for the matched GO terms for all clusters (y-axis) versus negative log p-value for p-values in k-means method. UDIM performs better than k-means with a higher number of data points above diagonal.

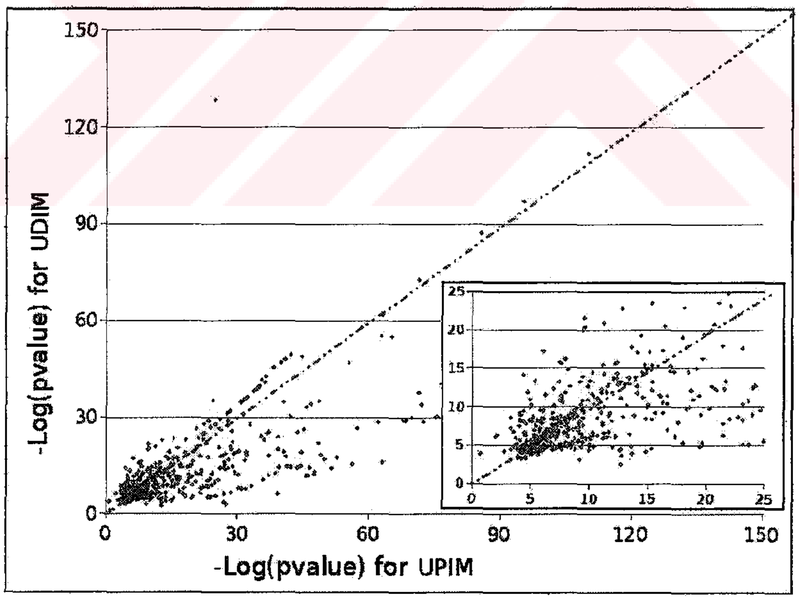


Figure 4.6: Negative log p-value calculated for best p-values for the matched GO terms for all clusters (y-axis) versus negative log p-value for p-values in UPIM. UPIM performs better than UDIM with a higher number of data points below diagonal.

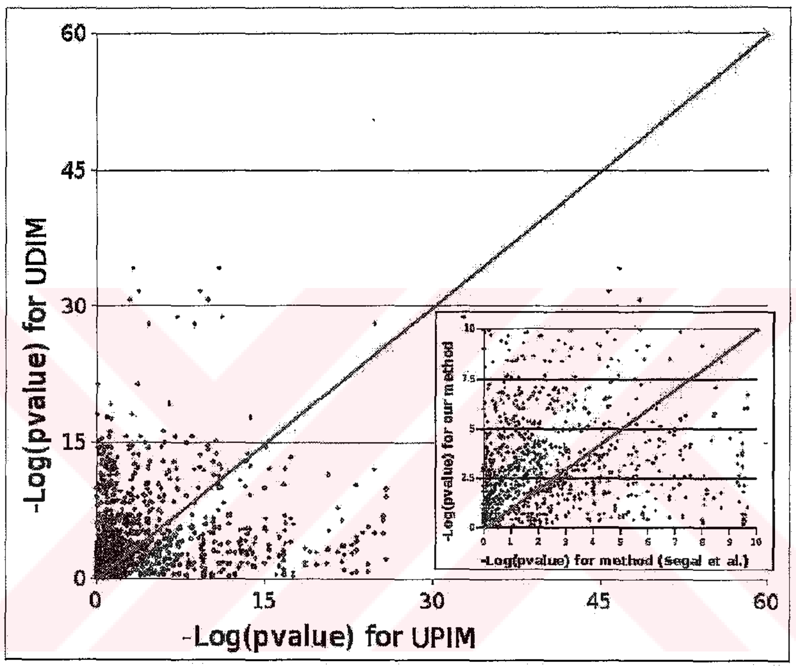


Figure 4.7: Negative log p-values calculated for top 3 pathways for the matched GO terms (y-axis) in UDIM versus negative log p-value for top 3 pathways in UPIM. UDIM performs slightly better with 901 GO terms above diagonal and 710 GO terms below diagonal.

Domain-domain interaction weights were calculated from an association rule mining set, where minimum support for a rule was limited by 0.0000493. This number is calculated by taking inverse of the two times the number of interactions in the interaction data. If a minimum support value which is higher than 0.00493% is used, a protein interaction with an interacting domain pair that is observed only once in the data will be absent in the protein interaction network that is created using the edge weights. For instance, let's say that there exists an interaction between proteins A and B, which are single domain proteins that have domains x and y, respectively. If there is no other interaction that has these two domains in the interacting pairs, then the support value for the association rule $x \rightarrow y$ will be 0.00493%. If a support more than this value is used, interaction edge between protein A and B will be absent in the graph induced by calculated edge weights, which results in loss of valuable information.

Number of association rules used to generate the domain-domain interaction weights was 20038. This created 7906 interactions among 3587 genes, after using the CREATE INTERACTION GRAPH function (Algorithm 3).

Chapter 5

PARALLELIZATION OF BELIEF PROPAGATION ALGORITHM

5.1 Introduction

Solving inference problems is usually computationally expensive. In our case, even though approximate inference is performed, the algorithm takes considerable time to terminate. This is mainly because inference should be run at every iteration of the EM algorithm. EM runs until the θ values converge. With the data set used, the algorithm took from 5 hours to 28 hours[1]. Most time consuming operation was the calculation of messages until convergence in BP (Belief Propagation) algorithm. Usually, BP ran for about 10 iterations until convergence. Rarely, BP did not converge. Number of iterations for BP in these cases was an upper limit that is given as an input to the algorithm.

Time complexity of BP is closely related with the number of pathways to be learned[2]. It is also proportional to the number of edges in the interaction network. In our case, interaction network has an average node degree of 6.31. Distribution of degrees over the number of nodes is given in Figure 5.1.

Table 5.1 shows the running times of different sections of the code for 5 EM iterations and 4 BP iterations for every EM iterations. According to this table, BP Algorithm takes 90% of the running time of the program. Next section covers a parallel BP algorithm for distributed memory machines.

[1]Run time varies with parameters.

[2]See Algorithm 12 in Appendix G.

Table 5.1: Statistics for the sequential version of the algorithm. Times given in milliseconds.

*N*ode	*Bp*	*BpInit*	*M-Step*
NODE 0	1301049	68962	62987
		Run Time:	1435192

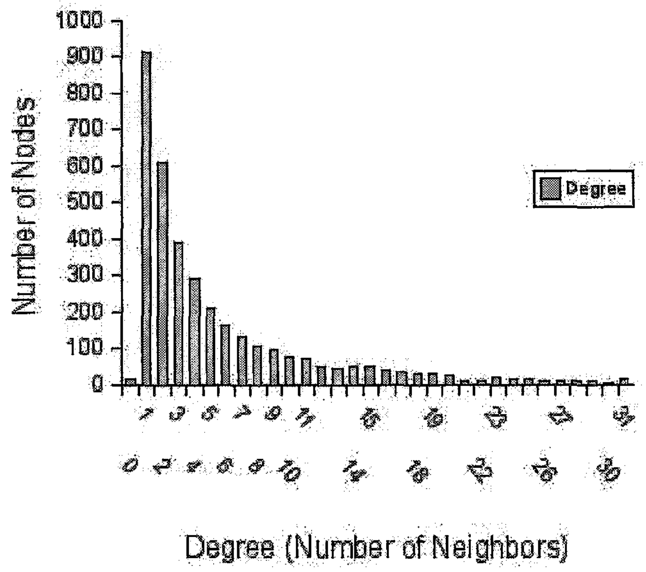


Figure 5.1: Distributions of the degrees of nodes in the protein-protein interaction network.

5.2 *Parallelization of BP Algorithm*

In order to execute BP on many processors, nodes of the graph must be distributed to processors. Each node requires information from neighboring nodes. Messages that come to i from its neighbors, or messages that go to neighbors from i can be stored in each node. Former scheme is chosen, which implies that a node can calculate a message it sends to its neighbor by using its local information, but it should then send this message to its neighbor. To get the best speed-up, computational load should be distributed equally and communication between processors has to be minimized as well.

For example Figure 5.2 illustrates a distribution of nodes of a prototype graph to two processors *Processor 1* and *Processor 2*. Since there are two edges that are cut by the partitioning, four messages should be sent while calculating messages m_{15}, m_{51}, m_{25}, m_{52}. All other messages can be calculated without the need for communication. According to this scheme, beliefs can be calculated in parallel, by multiplying the incoming messages at a node with local beliefs.

This scheme can easily be implemented with a message passing library. However, it has a number of disadvantages. First, number of messages exchanged between two processors

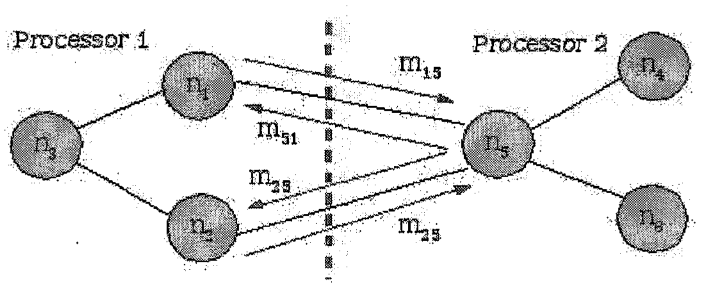


Figure 5.2: A simple network distributed to two processors.

is proportional to the number of edges between these processors, which are cut by the partitioning. Second, redundant sends and receives are made. For example, in Figure 5.2, to calculate the message m_{54}, *Processor 2* receives messages from nodes n_1 and n_2. Likewise, it has to receive the same messages in order to calculate the message m_{56}. Third, the need for communication brings up the need for synchronization, which slows down the calculation of the messages. A final remark is that the amount of required modification to existing sequential code is high.

In a second scheme, *proxy nodes* and *home nodes* are introduced. This scheme aims to avoid the drawbacks of the former scheme by delaying the communication between processors until the end of the iteration and sending all the messages that should go from one processor to another in a single message. Figure 5.3 illustrates this scheme.

In this type of communication pattern, communication overhead is very low compared to the first scheme. Because at most one send and one receive operation is made between two processors, number of messages between processors is no longer proportional to the number of edges cut by the partitioning. However, total communication volume between processors is still the same.

Additionally, using proxy nodes makes it possible to make all calculations of a single iteration locally on each processor. Since the communication is delayed until the end of the iteration, processors do not need to get synchronized during an iteration of BP. Proxies also allow us to reuse the sequential code. Amount of required modification to the sequential code is just to implement the *Manager* class and its *UpdateProxies()* method.

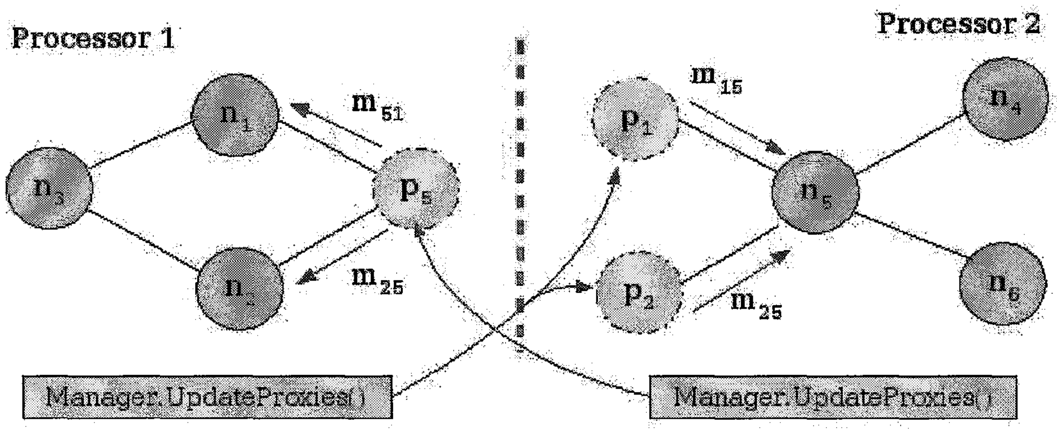


Figure 5.3: Network with proxy and home nodes. p_1, p_2 and p_5 are proxies and others are home nodes. Manager object is responsible for updating proxies (i.e. sending the newly calculated message values of every processor). Manager class knows which home node has proxy nodes at which processor. By using this information, it packs all the messages that will go to appropriate processors and sends them as a single message after each iteration of BP.

5.3 *Distributing Computation to Processors*

In order to get the expected speed-up, work should be distributed to processors in a balanced manner. In our case, work to be distributed is the genes in the interaction network. Genes are distributed to processors. However, one should take into account that not every gene requires same amount of computation.

When distributing the nodes to processors, there are three kinds of computations that should be balanced. First one is the iterations of BP algorithm, where every processor calculates the messages coming to its home nodes from their neighbors. According to the message update rules of BP algorithm, complexity of computing the message from node x_i to node x_j is proportional to $k\sum_{x_p \in Neighbors(x_i)} Degree(x_p)$. The multiplier k comes from the dimension of message vectors, which is the number of pathways. Summation term is the sum of degrees of all neighbors of node x_i. This summation corresponds to the steps required to multiply all incoming messages to node x_i except from x_j plus multiplication of the local belief of x_i with this product. Since the k is the same for all nodes, it can be dropped and this weight is calculated as $W_{bp} = \sum_{x_p \in Neighbors(x_i)} Degree(x_p)$.

Second computation that should be balanced comes from the normalization of messages. After every iteration of BP algorithm, messages are normalized so that the messages that come to a node from one of its neighbors sum up to 1. Also the messages of home nodes are initialized before every EM iteration. Time it takes to complete these two routines are the same and is proportional to: $W_{msgNorm} = Degree(x_i)$ for and node x_i.

Third computation is the initialization of the belief arrays and normalization of beliefs. Time it takes to complete these computations are the same for all nodes, and is denoted with a uniform weight: $W_{bpInit} = 1$.

Table 5.2: Explanation of the three weighting schemes used.

Weighting Scheme	*Weights*
W_{bpInit}	Uniform weights for all nodes
$W_{msgNorm}$	Degree of the node
W_{bp}	Sum of the degrees of node neighbors

While partitioning the interaction network to distribute genes to processors, one should use a graph partitioning algorithm, which balances the three weights and also minimizes the total edge cut induced by the partitioning. A final improvement would be to balance the communication among the processors.

Metis [22] "Family of Multilevel Partitioning Algorithms" package was used to partition the interaction graph. Metis has two standalone programs for partitioning graphs: *p-metis* [24] and *k-metis* [23]. P-metis is preferred for partitioning graphs into number partitions less than 8. Both algorithms try to balance the load and minimize the total edge cut. However, p-metis produces more balanced partitions, while k-metis allows a load imbalance up to 3%. K-metis also tries to balance the communication among the processors.

Interaction network was partitioned with various weighting schemes. All subsets of the three weighting schemes were experimented with both algorithms (p-metis and k-metis) in different precedence to find the best partitioning. Results from some experiments for 5 EM iterations and 4 BP iterations per 1 EM iteration are given at Table 5.3. Looking at the running times, one can make the following comments: worst partitioning is the one with uniform weights (scheme W_{bpInit}). This is because with uniform weights on vertices, both algorithms just balance the number of nodes per partitioning, which is not enough to balance the computation.

Table 5.3: Running time of the program for different weighting schemes: W_{bp}, $W_{msgNorm}$ W_{bpInit}. Experiments made for 8 processors, times in seconds.

Weighting Scheme	*p-metis*	*k-metis*
W_{bpInit}	255.0	251.6
W_{bp}	213.1	214.9
$W_{msgNorm}$	209.1	202.9
W_{bp}, W_{bpInit}	206.7	205.3
W_{bpInit}, W_{bp}	203.8	212.6
W_{bp}, $W_{msgNorm}$	201.5	202.3
$W_{msgNorm}$, W_{bp}	198.7	201.4
$W_{msgNorm}$, W_{bpInit}	205.4	210.7
W_{bpInit}, $W_{msgNorm}$	208.8	203.7
W_{bp}, $W_{msgNorm}$, W_{bpInit}	199.3	199.4
W_{bpInit}, W_{bp}, $W_{msgNorm}$	207.9	203.3

Best partitioning scheme is: $W_{msgNorm}$, W_{bp}. The reason scheme W_{bp}, $W_{msgNorm}$ is not good enough is because of the balance of the partitions. P-metis reports load unbalance of 0%, 1% for scheme $W_{msgNorm}$, W_{bp} and 1%, 1% unbalance for scheme W_{bp}, $W_{msgNorm}$. One of the weights are balanced worse than the other one in the latter scheme and this accumulates in the total run time by a difference of about 2.5 seconds.

Another interesting result is that the partitioning scheme $W_{msgNorm}$, W_{bp} is even better than scheme W_{bp}, $W_{msgNorm}$, W_{bpInit}. Although this scheme includes all weights, it results in a poor partitioning because it is harder to balance the three weights at the same time. P-metis reports 1% of unbalance in this scheme for all weights. Likewise, performance of the scheme W_{bpInit}, W_{bp}, $W_{msgNorm}$ is worse than W_{bp}, $W_{msgNorm}$, W_{bpInit}, because the partitioning is more unbalanced with (1%, 2%, 1%).

Table 5.4: Statistics for the algorithm after Parallelization of BP part. Times in milliseconds.

Node	*Bp*	*Update*	*BpInit*	*M-Step*	*Up+Bp*	*Idle*
0	156125	15679	22142	50595	171804	14257
1	164350	8900	20449	50596	173250	7598
2	162631	9458	20599	50597	172089	8295
3	156058	14890	22551	50594	170948	13059
4	164497	9396	18505	50596	173893	8388
5	161955	10740	21022	50596	172695	9647
6	163215	12958	17511	50595	176173	11749
7	167705	8018	16656	50596	175723	7043
					Run Time:	246095

5.4 *Improving Performance of the Program*

Upon the completion of the parallelization of BP algorithm, it was observed that even though the speed-up of BP algorithm were as high as 7.4 on 8 processors, overall speed-up of the program was about 5.5. This was mainly because of the sequential bottlenecks (m-step) and some load imbalance (Tables 5.1, 5.4).

All the measurements in Tables 5.4, 5.5, 5.6 and 5.7 are in milliseconds. Each table cell shows the running time of a task in the corresponding node in a PC cluster of eight nodes. Weighting scheme used for these measurements is: $W_{msgNorm}$, W_{bp}.

5.4.1 Parallelizing M-Step

In the m-step of the EM algorithm, first new theta values are calculated by using the newly calculated beliefs and then means and standard deviations of $k*m$ CPDs are updated. This update operation takes $O(kmn)$ time, where n is the number of genes. This operation is done in a loop shown in Algorithm 4.

To distribute the load, it is chosen to distribute the pathways to processors. However, distribution of genes to processors is irrelevant for this solution. Thus, the algorithm uses an alternative load distribution scheme (i.e. distribution of pathways to processors instead of genes to processors) at the m-step routine. From the start, every processor knows for

Algorithm 4 *M-STEP( ) (Sequential)*

```
1: Calculate new theta values
2: for $pw = 0$ to $k$ do
3:   for $exp = 0$ to $m$ do
4:     $xpj1[pw][exp] = 0$; $xpj2[pw][exp] = 0$;
5:     for $gid = 0$ to $n$ do
6:       $xpj1[pw][exp] += Belief(gid, pw) * ExpressionValue(gid, exp)$;
7:       $xpj2[pw][exp] += Belief(gid, pw) * ExpressionValue(gid, exp)^2$;
8:     end for
9:   end for
10: end for
11: Update the means and standard deviations using $xpj1$ and $xpj2$ arrays.
```

which pathways it will calculate the values of $xpj1$ and $xpj2$ arrays. Then, processors compute their division of mean and standard deviations independently, and in the end a single MPI_Allgather for every pathway solves the problem (i.e. speed-up = 6, Tables 5.5 vs 5.1).

5.4.2 Interleaving Sends and Receives in the UpdateProxies Routine

Second observation was the idle times the processors spent during the updating of proxies which takes place after every iteration of the BP algorithm. In this routine, every processor sends messages to other nodes, which has a proxy node that corresponds to its home nodes. Other processors then receive the message and update their proxy nodes.

In the first version of UpdateProxies (Algorithm 5), first all the messages were packed into send buffers. Then, messages are received and sent by non-blocking receives and sends. Once the communication is complete, received messages are unpacked.

In order to decrease the idle times spent during the waiting of send/receive operations, UpdateProxies routine were improved by interleaving the sends to different processors. In the first version (Algorithm 5), messages that will be sent to all processors were packed and sent for all processors. In the second version, (Algorithm 6), send routine is executed as soon as the messages that will be sent to a node are packed. Likewise, it is not needed to

Table 5.5: Statistics for the algorithm after Parallelization of M-Step part. Times in milliseconds.

Node	*Bp*	*Update*	*BpInit*	*M-Step*	*Up+Bp*	*Idle*
0	155739	28438	22842	10886	184177	24104
1	161550	25787	21091	9421	187337	20113
2	162896	24230	21204	10135	187126	19391
3	180164	8486	19068	10899	188650	4530
4	172326	18211	18039	10133	190537	13282
5	155550	31309	22945	8680	186859	24395
6	162535	24605	21698	9425	187140	19121
7	166456	24788	17126	8674	191244	18645
					Run Time:	220832

Algorithm 5 *UPDATE-PROXIES_1( )*

```
1: Let $P$ be the *Number of Processors*
2: for $i = 0$ to *Number of Home Nodes* do
3:    for *location* ∈ Proxy Locations of home node $i$ do
4:       Pack messages array of home node $i$ to send buffer of *location*
5:    end for
6: end for
7: for *location* = 0 to $P$ do
8:    Receive messages from *location*.
9: end for
10: for *location* = 0 to $P$ do
11:    Send messages to *location*.
12: end for
13: for *location* = 0 to $P$ do
14:    Wait Send / Receive operations for *location* to finish.
15: end for
16: for *location* = 0 to $P$ do
17:    Unpack messages received from node *location*.
18: end for
```

wait for the pack operations to complete to receive the messages from other nodes. This way, it was observed some improvement in idle times processors spent in the update proxies routine. Difference can be seen by comparing Tables 5.5 and 5.6.

Algorithm 6 *UPDATE-PROXIES_2( )*

```
1: Let $P$ be the Number of Processors
2: for location = 0 to $P$ do
3:     Receive messages from location.
4: end for
5: for location = 0 to $P$ do
6:     Pack messages that will be sent to processor location
7:     Send messages to location
8: end for
9: for location = 0 to $P$ do
10:    Wait Receive operations for location to finish.
11:    Unpack messages received from location.
12: end for
13: for location = 0 to $P$ do
14:    Wait Send operations for location to finish.
15: end for
```

5.4.3 Parallelization of Bp Initializations

Last sequential bottleneck was the routine that initializes the belief and message arrays before each EM iteration. Normally, every processor initialized the arrays of its own home nodes and proxy nodes. However, the number of proxy nodes on each processor was usually comparable to the number home nodes. Because of this fact, speed-up of this part (compare Tables 5.4 and 5.1) was about 3.1. If there was no initializations made for proxy nodes (which must be done), then a speed-up of 6 would be expected because of the comparable number of proxy nodes and home nodes. Solution was to initialize the proxy nodes by using communication instead of computation. Processors initialize their home nodes, and then update their proxies by sending initial message and belief values to other processors.

Table 5.6: Statistics for the algorithm after interleaving of send/receive operations in UpdateProxies part. Times in milliseconds.

Node	*Bp*	*Update*	*BpInit*	*M-Step*	*Up+Bp*	*Idle*
0	158626	15238	22866	10878	173864	11090
1	165051	11213	19125	10882	176264	7070
2	163914	11181	21281	10155	175095	6303
3	164502	14764	18113	10149	179266	9800
4	163271	12972	21761	9423	176243	7438
5	162329	13749	21140	9420	176078	8257
6	156945	18372	23372	8672	175317	11916
7	167430	14020	17172	8668	181450	7704
					Run Time:	210667

This communication made the algorithm avoid the redundant computations and the speed-up at this part has improved from 3.1 to 4.8 (Table 5.7). On 8 processors, it took 21 seconds on the average to initialize the BP arrays by computing the initial values of home and proxy nodes. It took 14 seconds on the average to initialize the BP arrays by first computing the initial values of home nodes and communicating these values with other processors.

5.5 Results and Discussion

The program was ran in two PC clusters; one with 8 nodes and a gigabit switch (1 Gbit/sec), and the other with 32 nodes and a Fast Ethernet switch (100 Mbit/sec).

Maximum speed-up on 8 processors was 7.22 on the first cluster. Because of the gigabit switch, communication overhead did not affect the performance much. Indeed, most of the computation was exchanged with communication, because at some routines like the initializations of BP arrays, it was faster to communicate the initial belief values of home nodes to their proxies than to compute the values of proxy nodes on the processors.

On the other hand, maximum speed-up on 8 processors on the second cluster with Fast Ethernet switch was 5.9. This decrease in the speed-up is because of the slowness of the

Table 5.7: Statistics for the algorithm after parallelization of BP initializations part. Times in milliseconds.

Node	*Bp*	*Update*	*BpInit*	*M-Step*	*Up+Bp*	*Idle*
0	157550	7066	14206	10851	164616	5900
1	161013	7915	13860	9394	168928	5296
2	163092	7851	14181	10097	170943	6335
3	155244	9417	14215	8660	164661	5138
4	165249	6216	13427	10852	171465	6520
5	164110	8403	14061	9395	172513	5806
6	163631	7390	13559	10095	171021	8135
7	166254	8145	13454	8655	174399	7342
					Run Time:	198769

switch. Average size of messages that a processor receives during updating of the proxy nodes was in the order of Megabytes. Since messages were large (around 10 Megabytes), speed difference between Fast Ethernet and gigabit switch affected the speed of the program significantly. Table 5.8 shows the amount of data sent from each processor while updating of the proxies in a single iteration of BP. According to the table, total communication volume is: 184.55 Megabytes, which explains why the performance of the switch is crucial.

In order to be convinced that the decreasing of speed-up was due to communication bottleneck, communications part was skipped, which resulted in a speed-up of about 31 on 32 processors.

With 32 and 16 processors, maximum speed-up was 16.9 and 10.5, respectively. Tables 5.9 and 5.10 show the partitioning scheme that yields the highest speed-up on different machines and different number of processors.

Finally, speed-ups were calculated by distributing the nodes to processors randomly. Table 5.11 shows the speed-ups on cluster with 8 processors and gigabit switch. As expected, as the number of nodes increase, speed-up decreases more rapidly with random partitioning.

Table 5.8: Amount of data sent (in Megabytes) from each processor in the *UpdateProxies* procedure.

Node	*Data*
0	21.50
1	36.09
2	14.22
3	19.47
4	20.92
5	21.63
6	21.59
7	29.14

Table 5.9: Maximum Speed-up values for runs on network with gigabit switch.

Nodes	*Speed-up*	*Weighting Scheme*	*Partitioning Program*
8	7.22	$W_{msgNorm}$ W_{bp}	p-metis
4	3.76	$W_{msgNorm}$ W_{bp}	p-metis
2	1.90	$W_{msgNorm}$ W_{bp}	p-metis

Table 5.10: Maximum Speed-up values for runs on network with Fast Ethernet switch.

Nodes	*Speed-up*	*Weighting Scheme*	*Partitioning Program*
32	16.91	W_{bpInit} W_{bp}	k-metis
16	10.5	W_{bp} $W_{msgNorm}$	k-metis
8	5.9	$W_{msgNorm}$ W_{bp} W_{bpInit}	k-metis
4	3.14	$W_{msgNorm}$ W_{bp}	p-metis
2	1.72	$W_{msgNorm}$ W_{bp}	p-metis

Table 5.11: Speed-up values for runs on network with gigabit switch and random partitioning.

Nodes	*Speed-up*
8	3.58
4	3.37
2	1.91

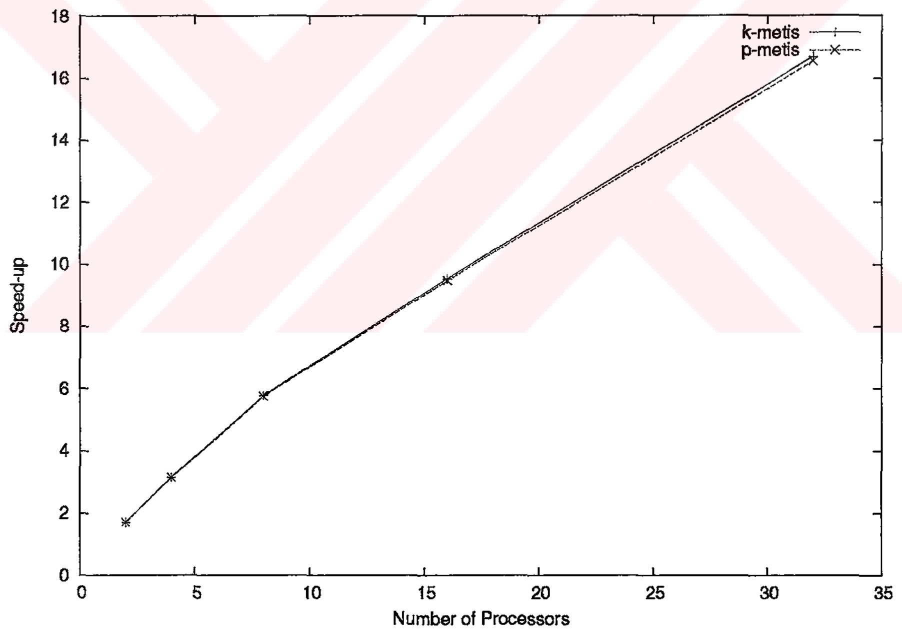


Figure 5.4: Speed-up graph for 32 processors. Experiment made with weighting scheme W_{bp} $W_{msgNorm}$. Performance for both p-metis and k-metis partitioning algorithms are shown. Partitioning with k-metis works slightly better.

Chapter 6

CONCLUSION

Integrating biological information from various sources to infer new knowledge is beneficent in bioinformatics. This is because amount of biological data is high, but the connection between data is not obvious. In order to extract valuable information, one often needs to unite data from various sources. In this thesis, integrating biological data from various sources to infer knowledge about the pathways, in which proteins act was studied.

First method presented uses association rule mining to integrate protein-protein interaction data with domain decompositions of proteins. Using information from both protein-protein interaction and domain databases, the method outputs domain-domain interaction rules that imply putative interactions between proteins. Rules can be summarized as "a protein that has particular domains *often* interacts with another protein that has another set of particular domains". Dependability of the rules varies with support and confidence parameters. Application of association rule mining to protein-protein interaction data and domain data required a set of modifications to the layout of the data. According to the proposed method, interacting proteins were substituted with their sets of domains. Then, domains were labeled in order to identify if they belonged to a LHS protein or RHS protein. Finally, LHS and RHS of the interactions were collapsed into a single set. It was observed that the support parameter was more limiting than the confidence parameter. Number of rules output by the algorithm decreased rapidly when the minimum support parameter was increased. Some of the rules discovered were confirmed by biological literature.

As a future work, different features can be incorporated along with the domain decompositions of the proteins. For example, motifs, amino acid patterns, and expression profiles can yield to interesting rules. Also, rules that are generated frequently with reasonable support and confidence value pairs can be checked with laboratory experiments. By this way, it can be understood if the method can discover novel protein-protein interactions.

The second method presented (UDIM) uses the results from the first method to integrate domain-domain interaction rules with gene expression data clustering. The method is based on a probabilistic model (UPIM), which is trained using EM algorithm and BP. Domain-domain interaction rules are integrated into UPIM by calculating the weights of edges between the proteins. Weights are based on the support parameters of the domain-domain interaction rules between the protein pairs. Results of the method were compared with k-means clustering algorithm, which was run using gene expression data only. Number of GO terms that match the genes in clusters with the best p-value, was higher than that of k-means. This showed an improvement over the k-means clustering. Also, number of GO terms that match the genes in top three clusters with the best p-value was higher than that of the original method (UPIM). As a future work, it can be worked on a reverse procedure, where domain-domain interaction rules are refined using gene expression data.

Finally, a parallel algorithm for UDIM was presented. Since the running time of UDIM was in the order of tens of hours, parallelization was required to get the results of the method as quickly as possible. Protein interaction network was partitioned and distributed to processors. Weights were assigned to each node in the graph, which represented the computation required in BP, message normalizations and initializations parts. Partitioning was done by using a graph partitioning tool, which balanced the weights given. Different weighting schemes were experimented. For example, when a single, uniform weight was used, the algorithm performed worst, however when two weighting schemes that represent BP and message normalizations were used together, it performed the best. Results showed almost linear speed-ups on a PC cluster with 8 nodes and a gigabit switch. Because of the communication bottleneck, the same speed-ups were not observed on another PC cluster with 32 nodes and a Fast Ethernet switch. For example, with 8 processors, maximum speed-up was 7.22 and 5.9 on clusters with gigabit switch and Fast Ethernet switch, respectively. This shows that the communication bandwith is more important than the latency for the performance of the algorithm.

Appendix A

DETAILS OF BEST GENE CLUSTERS

Table A.2 lists the GO terms that matches the genes in the top clusters of k-means clustering and Table A.1 lists the genes in this cluster. Best cluster is chosen as the one that has the highest number of GO terms matched to its genes, below a p-value of 0.05. Similarly, Tables A.4 and A.3 list the same information for the best cluster from the results of UDIM.

Table A.2: GO terms and their p-values that matches the best cluster in k-means.

GO Id	*GO term*	*p-value*
GO:0008283	cell proliferation	6.99E-016
GO:0000067	DNA replication and chromosome cycle	1.16E-015
GO:0007049	cell cycle	1.18E-015
GO:0006260	DNA replication	2.03E-011
GO:0006261	DNA-dependent DNA replication	4.31E-011
GO:0006271	DNA strand elongation	8.80E-011
GO:0006259	DNA metabolism	7.42E-010
GO:0008151	cell growth and/or maintenance	8.52E-009
GO:0006270	DNA replication initiation	1.01E-008
GO:0050875	cellular physiological process	3.53E-008
GO:0006139	nucleobase, nucleoside, nucleotide and nucleic acid metabolism	7.03E-008
GO:0009987	cellular process	1.27E-007
GO:0006325	establishment and/or maintenance of chromatin architecture	5.81E-007
GO:0006323	DNA packaging	5.81E-007
GO:0006265	DNA topological change	7.01E-007

Table A.2 – continued

GO Id	*GO term*	*p-value*
GO:0050789	regulation of biological process	1.66E-006
GO:0006351	transcription, DNA-dependent	1.81E-006
GO:0000279	M phase	1.81E-006
GO:0006350	transcription	1.84E-006
GO:0006997	nuclear organization and biogenesis	3.33E-006
GO:0006338	chromatin remodeling	3.76E-006
GO:0016582	non-covalent chromatin modification	3.76E-006
GO:0007001	chromosome organization and biogenesis (sensu Eukarya)	4.79E-006
GO:0000278	mitotic cell cycle	6.60E-006
GO:0040029	regulation of gene expression, epigenetic	1.39E-005
GO:0016568	chromatin modification	1.53E-005
GO:0006273	lagging strand elongation	2.53E-005
GO:0019219	regulation of nucleobase, nucleoside, nucleotide and nucleic acid metabolism	3.20E-005
GO:0000280	nuclear division	5.34E-005
GO:0006355	regulation of transcription, DNA-dependent	5.49E-005
GO:0045814	negative regulation of gene expression, epigenetic	6.73E-005
GO:0006342	chromatin silencing	6.73E-005
GO:0016458	gene silencing	6.73E-005
GO:0045449	regulation of transcription	7.68E-005
GO:0006267	pre-replicative complex formation and maintenance	8.87E-005

Table A.4: GO terms and their p-values that matches the best cluster in UDIM.

GO Id	*GO term*	*p-value*
GO:0009987	cellular process	1.40E-015
GO:0050875	cellular physiological process	1.79E-014
GO:0008151	cell growth and/or maintenance	4.66E-014

Table A.4 – continued

GO Id	*GO term*	*p-value*
GO:0008283	cell proliferation	3.11E-013
GO:0007049	cell cycle	6.41E-013
GO:0050789	regulation of biological process	4.53E-009
GO:0006342	chromatin silencing	8.75E-008
GO:0016458	gene silencing	8.75E-008
GO:0045814	negative regulation of gene expression, epigenetic	8.75E-008
GO:0000067	DNA replication and chromosome cycle	1.51E-007
GO:0040029	regulation of gene expression, epigenetic	1.57E-007
GO:0000279	M phase	3.05E-007
GO:0019219	regulation of nucleobase, nucleoside, nucleotide and nucleic acid metabolism	4.08E-007
GO:0006323	DNA packaging	4.66E-007
GO:0006325	establishment and/or maintenance of chromatin architecture	4.66E-007
GO:0019222	regulation of metabolism	5.16E-007
GO:0050791	regulation of physiological process	5.16E-007
GO:0009892	negative regulation of metabolism	1.00E-006
GO:0016043	cell organization and biogenesis	1.04E-006
GO:0045449	regulation of transcription	1.09E-006
GO:0006259	DNA metabolism	1.54E-006
GO:0016571	histone methylation	1.78E-006
GO:0016481	negative regulation of transcription	2.03E-006
GO:0045934	negative regulation of nucleobase, nucleoside, nucleotide and nucleic acid metabolism	2.43E-006
GO:0016568	chromatin modification	2.43E-006
GO:0000280	nuclear division	2.43E-006
GO:0006997	nuclear organization and biogenesis	2.64E-006
GO:0006355	regulation of transcription, DNA-dependent	3.01E-006
GO:0007001	chromosome organization and biogenesis (sensu Eukarya)	3.88E-006

Table A.4 – continued

GO Id	*GO term*	*p-value*
GO:0006479	protein amino acid methylation	4.36E-006
GO:0008213	protein amino acid alkylation	4.36E-006
GO:0016569	covalent chromatin modification	4.97E-006
GO:0016570	histone modification	4.97E-006
GO:0000278	mitotic cell cycle	5.41E-006
GO:0006260	DNA replication	6.44E-006
GO:0006348	chromatin silencing at telomere	9.40E-006
GO:0045892	negative regulation of transcription, DNA-dependent	1.09E-005
GO:0016582	non-covalent chromatin modification	1.93E-005
GO:0006338	chromatin remodeling	1.93E-005
GO:0007067	mitosis	2.62E-005
GO:0006261	DNA-dependent DNA replication	2.80E-005
GO:0000087	M phase of mitotic cell cycle	3.04E-005
GO:0006350	transcription	4.82E-005
GO:0007582	physiological process	4.82E-005
GO:0006351	transcription, DNA-dependent	5.16E-005
GO:0006334	nucleosome assembly	6.87E-005
GO:0006333	chromatin assembly/disassembly	7.92E-005
GO:0040020	regulation of meiosis	0.01
GO:0007093	mitotic checkpoint	0.01

Table A.1: Genes in the best cluster of k-means clustering.

*G*ene Names
YER011W YIL011W YGR035C YBR186W YCR059C
YGR208W YNL078W YNL210W YNL226W YNL207W
YFR038W YDR044W YOR204W YNL216W YBR115C
YPR018W YNR027W YJL115W YOL006C YJL075C
YIL149C YKL042W YNL102W YPR120C YGL175C
YLR103C YMR144W YBR141C YBR086C YER168C
YFL008W YOR355W YNL262W YNR025C YBL035C
YKL068W YJR053W YHR031C YKL033W YMR179W
YBL014C YKL045W YBR202W YML023C YDL164C
YHR204W YJL025W YOR217W YIL129C YLR433C
YDR331W YGR004W YDR448W YCR052W YMR270C
YBR049C YLR357W YKR010C YPR104C YLR451W
YKR036C YMR211W YMR209C YER171W YGL145W
YMR288W YIL085C YER149C YMR221C YJL135W
YER164W YBL052C YBR156C YER032W YJL187C
YMR078C YML109W YNL068C YCR054C YPL210C
YML046W YNL273W YIL150C YJL074C YJR030C
YPL255W YBL031W YFR031C YGL256W YLR353W
YHR061C YLR210W YMR277W YCL024W YLR383W
YIL009W YCR065W YJR031C YKL072W YNL126W
YPL128C YJL011C YJL061W YMR266W YNL088W
YML037C YCR024C-A YKL008C YGL116W YOR038C
YNR017W YLR045C YNR018W YIL141W YLR223C
YNL221C YNL227C YML065W YDL093W YBL004W
YPL209C YMR199W YHR120W YDL171C YLL004W
YLR406C YOR025W

Table A.3: Genes in the best cluster of UDIM.

Gene Names				
YAR003W	YBL008W	YBL053W	YBR103W	YBR115C
YBR158W	YBR175W	YBR195C	YBR198C	YBR234C
YBR281C	YCL039W	YCR012W	YCR067C	YCR084C
YDL127W	YDL145C	YDL195W	YDR080W	YDR142C
YDR146C	YDR364C	YEL055C	YEL056W	YER066W
YER082C	YER107C	YER124C	YFL009W	YFR021W
YGL003C	YGL004C	YGL028C	YGL100W	YGL116W
YGL137W	YGL190C	YGL213C	YGR089W	YGR108W
YGR128C	YGR154C	YGR175C	YGR180C	YGR200C
YGR210C	YHL022C	YHR119W	YHR129C	YHR186C
YIL046W	YIL047C	YIR012W	YJL035C	YJL112W
YJL115W	YJR033C	YJR086W	YJR112W	YKL018W
YKL021C	YKL042W	YKL045W	YKL130C	YKL213C
YKR036C	YLL004W	YLR015W	YLR045C	YLR055C
YLR129W	YLR208W	YLR263W	YLR403W	YLR429W
YML065W	YML071C	YML102W	YML104C	YMR032W
YMR038C	YMR092C	YMR093W	YMR102C	YMR146C
YMR199W	YMR214W	YMR251W	YMR319C	YNL006W
YNL035C	YNL050C	YNL078W	YNL191W	YNL218W
YNL253W	YNL317W	YNR017W	YNR018W	YNR027W
YNR029C	YNR042W	YOL005C	YOL006C	YOL018C
YOL090W	YOL103W	YOL136C	YOR025W	YOR026W
YOR038C	YOR047C	YOR074C	YOR212W	YOR229W
YOR269W	YOR360C	YPL036W	YPL139C	YPL151C
YPL158C	YPR018W	YPR069C	YPR175W	YPR178W

Appendix B

DNA MICROARRAY PROCEDURE

Basic idea behind the microarrays is to put a part of the sequences of the genes of an organism on a special chip, then to *hybridize* these sequences with pre-synthesized cDNA sequences that are simply a complementary sequence of the mRNAs extracted from cells. Hybridization is the name given to the procedure at which cDNAs bind to the sequences on the microarray chip.

There are two types of microarrays, oligonucleotide arrays and cDNA microarrays. Although the principle behind both types is the same, because of structural reasons, oligonucleotide arrays are used to monitor the genes of one cell at a time and cDNA arrays are used to monitor and compare the gene expression profiles of a control sample and experiment sample on the same chip. The procedure used with cDNA arrays is presented here. An experiment using a cDNA microarray involves the preparation of two samples for hybridization to the array [56]. One for the control and one for the experiment sample. These samples are created by extracting the mRNAs from the cells and reverse transcribing these mRNAs to cDNAs, which are single strand complement of the mRNAs synthesized with DNA nucleotides. During the reverse transcription, a fluorescent dye is incorporated into the newly formed cDNA. Different dye is used for control samples and experiment samples. Usually a green-fluorescing dye (cy3) and a red-fluorescing (cy5) dye are used for this purpose. Because of the different labeling, both cells can be hybridized on the same array. After scanning of the array, at a given spot, the color of the sample whose gene is expressed more will be dominant. So, if the control sample and experiment sample are labeled with red and green dye, respectively, a red spot tells us that that gene was expressed more in the control sample and a green spot tells that that gene was expressed more in the experiment sample. A yellow spot, which is the result of same amount of red and green labeled cDNAs means that the corresponding gene is expressed in both samples. Finally, a dark spot is the result of no hybridization, which means that the gene was not expressed in either samples.

Appendix C

NAIVE BAYES CLASSIFIERS

In Naive Bayes classifier, there is a set of instances with several attributes. Each instance belongs to a class and the classes are disjoint. After training, each class gets associated with a probability distribution which is defined over the attributes of the instances and the method classifies the future data by using the experience it gained from the training data. One problem with this approach is that when there are many attributes, in order to learn the probability distribution that a class is associated, an exponential number of instances are needed, which is usually not available. To get over this problem, Naive Bayes classifiers assume that the attributes of the instances are conditionally independent given the class they belong to. Thus, theoretically, Naive Bayes is optimal when observations of all the instances are independent from each other. Although this assumption is relaxed, the method works well in practice and it has proven to be quite successful in classifying various types of data [3, 4].

Let h denote the hypothesis, which classifies the data into disjoint classes and D denote the training data. Naive Bayes classifiers are then, based on the following formula, which calculates $P(h|D)$, the probability of a hypothesis given the data in terms of $P(D|h)$, the probability of data given the hypothesis, $P(h)$, the prior probability of hypothesis h and $P(D)$, probability of seeing data D.

$$P(h|D) = \frac{P(D|h).P(h)}{P(D)} \tag{C.1}$$

This equation simply tells us that the probability of a hypothesis given a data set increases with probability of seeing that data in a universe where the hypothesis is assumed to be true. It also increases with the probability of seeing that hypothesis among all other hypotheses in the hypothesis space. Finally, it decreases with the probability of seeing the data. So, when an instance which supports our hypothesis is seen, the information gained from the data is low if the data is a common one. The independence assumption is needed to calculate $P(D|h)$. What's more usually the factor $P(D)$ is dropped in Naive Bayes classifier calculations since it is the same for all hypotheses, so the equation becomes:

$$P(h|D) = P(D|h).P(h) \tag{C.2}$$

Using the independence assumption, usually this equation becomes like:

$$P(h|D) = \prod_{i=1}^{n} P(D_i|h).P(h) \tag{C.3}$$

where $P(D_i|h)$ is the probability of ith attribute given the hypothesis is true and n is the number of attributes. This separation of attributes simplifies the calculations to compute the probability $P(h|D)$. Finally, the hypothesis that maximizes this probability is chosen.

Appendix D

EM ALGORITHM

General statement of the EM algorithm is as follows [10]:

Let $X = \{x_1, ..., x_m\}$ denote the observed data in a set of m independently drawn instances , $Z = \{z_1, ..., z_m\}$ denote the unobserved data in these set of instances and $Y = X \cup Z$ denote the full data.

Note that Z can be treated as a random variable, whose distribution depends on unknown parameters Θ and the observed data X. Similarly, Y can also be treated as a random variable because it is expressed in terms Z, which is a random variable. Let h denote the current hypothesis. EM searches for a maximum likelihood hypothesis h' by seeking an h' that maximizes $E[lnP(Y|h')]$. This expectation value is calculated by using a probability distribution governing Y, which is due to unknown parameters Θ. Here, $P(Y|h')$ is the likelihood of full data in a universe where the hypothesis h' is true and maximizing the logarithm of this expectation also maximizes $P(Y|h')$.

Given that the full data Y is a combination of the observed data X and unobserved data Z, EM averages over possible values of unobserved random variable Z, weighing each possibility according to its probability. The distribution that governs Y is determined by X and the distribution that governs Z, which is not known. EM tries to estimate this distribution by using the current hypothesis h and the observed data X. It does so, by using the current hypothesis h in place of the unknown parameters Θ.

Let

$$Q(h|h') = E[lnP(Y|h')|h, X] \tag{D.1}$$

Here, Q gives $E[lnP(Y|h')|h, X]$ as a function of h', under the assumption that $\Theta = h$ and given X. Then, the following steps are applied until convergence:

1. **Step 1 - Estimation (E-Step):**

   Calculate $Q(h|h')$ using the current hypothesis h and other observed data X, to estimate the probability that governs Y.

2. **Step 2 - Maximization (M-Step):**

   Replace the hypothesis h with h' that maximizes this Q function.

$$h \longleftarrow argmax_{h'} Q(h'|h) \tag{D.2}$$

When the function Q is continuous, EM converges to a stationary point of the likelihood function $P(Y|h')$. If the function $P(Y|h')$ has a single maximum, EM is guaranteed to find it. Otherwise, it is guaranteed to find a local maximum. EM shares some of the same limitations as other optimization methods like the gradient descent, line search and conjugate gradient.

Appendix E

BELIEF PROPAGATION ALGORITHM AND MARKOV NETWORKS

Belief propagation is an algorithm to solve inference problems. The algorithm can be used for either exact inference or approximate inference. BP is one of the many methods to solve inference problems that come up in different fields.

Before explaining BP algorithm in detail, it is necessary to go over *Bayesian networks*, which is one of the most popular graphical models in AI literature. a small subset of genes and connections among them is taken as an example. For each gene, gene expression values in different experiments and other genes that it interacts with is known. An interaction between protein A and protein B increases the chance that A and B are in the same pathway. Each protein can be in one of the k pathways. The interactions among the proteins specify an independence relationship between the random variables that control the pathway assignments of the proteins. This independence relationships are used while computing the joint probabilities of all proteins to be assigned into particular pathways: $P(g_1.C, ..., g_n.C)$. Normally in Bayesian networks, these independence relations are described by directed edges. For example, an edge from gene i to gene j specifies the conditional probability $P(g_j.C = p|g_i.C = q)$, which is the probability of gene j to belong to pathway p given that gene i belongs to pathway q. While computing the joint probabilities, following equation [12] can be used:

$$P(g_1.C, ..., g_n.C) = \prod_{i=1}^{n} P(g_i.C|parents(g_i.C)) \tag{E.1}$$

where $parents(g_i.C)$ denotes the parents of gene i at the "directed acyclic" belief network. If gene $g_i.C$ has no parents, then $P(g_i.C|parents(g_i.C)) = P(g_i.C)$ is used.

Our ultimate goal in this inference problem is to calculate the *marginal* probabilities. This is to compute the probability that gene i belongs to a particular pathway in the light of the information from other genes. In order to do this, a sum over all variables other than $g_i.C$ in the joint probability function $P(g_1.C, ..., g_n.C)$ must be performed. This will require an enormous number of steps if all the random variables are dependent to each other. This is like the case where all the proteins interact with each other. However, since the number of interactions among the genes is much less than the number of possible protein pairs, the independence information from the graph can be used in order to calculate the sum:

$$P(g_n.C = k) = \sum_{g_1.C} \sum_{g_2.C} \cdots \sum_{g_{n-1}.C} P(g_1.C, ..., g_n.C) \tag{E.2}$$

In small Bayesian nets, this summation can be made directly, however, for bigger ones, this summation is intractable since it requires exponential number of steps. At this point, BP can be used. However, note that "Bayesian networks" are directed acyclic graphs, but our interaction network is an undirected graph. So, in our case, there are bi-directional dependency relationships between the random variables and the corresponding belief network has loops. BP is shown to be effective in approximating the marginal probabilities in graphical models with loops. [13]

BP algorithm is based on passing messages among the nodes in the graph. In BP algorithm, there are variables such as $m_{ij}(x_j)$, which can be defined as the message going from node i to node j, about what state node j should be in. This message is a vector of the same dimensionality as the random variable x_j. In this vector, each dimension measures the likelihood of node j being in a different state. In BP algorithm, the belief at a node i is calculated as the product of the messages coming to that node from its neighbors and the local evidence $\phi_i(x_i)$ on that node:

$$b_i(x_i) = k\phi_i(x_i) \prod_{j \in N(i)} m_{ji}(x_i) \tag{E.3}$$

where k is the normalization factor to make the beliefs sum to 1 and $N(i)$ denotes the neighbors of node i in the undirected dependency graph.

Messages are calculated by the following message update rules:

$$m_{ij}(x_j) \leftarrow \sum_{x_i} \phi_i(x_i)\psi_{ij}(x_i, x_j) \prod_{k \in N(i)\backslash j} m_{ki}(x_i) \tag{E.4}$$

where $k \in N(i)\backslash j$ means neighbors of node i except node j. To calculate the message from node i to node j, product of messages coming to node i except from node j is taken and

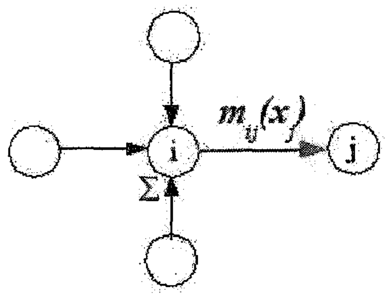


Figure E.1: Graphical illustration of message update rule: $m_{ij}(x_j) \leftarrow \sum_{x_i} \phi_i(x_i)\psi_{ij}(x_i, x_j) \prod_{k\in N(i)\backslash j} m_{ki}(x_i)$. To calculate the message from node i to node j, messages coming into node i are multiplied with the local belief at node i and summed over all states of x_i.

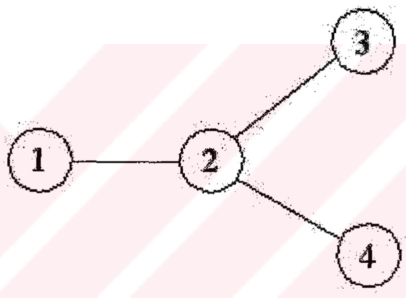


Figure E.2: Sample network with 4 nodes.

it is multiplied with the local belief on node i and the edge compatibility function between node i and j. Figure E.1 illustrates this calculation scheme.

It is easy to show that these message update rules give the beliefs that are exact if there are no loops in the graph[12]. For example, consider the network in Figure E.2.

$$b_1(x_1) = k\phi_1(x_1)m_{21}(x_1) \tag{E.5}$$

Using the message update rules, $m_{21}(x_1)$ is expanded to get:

$$b_1(x_1) = k\phi_1(x_1)\sum_{x_2} \psi_{12}(x_1, x_2)\phi_2(x_2)m_{32}(x_2)m_{42}(x_2) \tag{E.6}$$

Using the message update rules again, $m_{32}(x_2)$ and $m_{42}(x_2)$ are expanded to get:

$$b_1(x_1) = k\phi_1(x_1)\sum_{x_2}\psi_{12}(x_1,x_2)\phi_2(x_2)\sum_{x_3}\phi_3(x_3)\psi_{23}(x_2,x_3)\sum_{x_4}\phi_4(x_4)\psi_{24}(x_2,x_4) \quad \text{(E.7)}$$

If the sums are reorganized, it is easy to see that the belief at node 1 is the same as the exact marginal probability at node 1.

$$b_1(x_1) = k\sum_{x_2,x_3,x_4} p(\{x\}) = p_1(x_1) \quad \text{(E.8)}$$

Markov Networks

A Markov network is an undirected graphical model that is composed of a graph G and a set of functions $\phi = \{\phi_1, \phi_2, ..., \phi_n\}$ defined for n cliques of G. For a graph G, a clique is a set of nodes $\mathbf{V}_c$ in $\mathbf{V}$, not necessarily maximal, such that each $\mathbf{V}_i, V_j \in \mathbf{V}_c$ is connected by an edge in G. (Note that a single node by itself is also a clique). $\mathbf{V}$ denotes a set of discrete random variables for each clique in G and $\mathbf{v}$ denotes an assignment to variables $\mathbf{V}$. A Markov network then defines a joint distribution over $\mathbf{V}$. [7]

Let G=($\mathbf{V}$,E) be an undirected graph of nodes $\mathbf{V}$, edges E and a set of cliques C(G). Then each c $\in$ C(G) is associated with a set of nodes and a clique potential $\phi_c(V_c)$, which is a non-negative function defined on the joint domain of $\mathbf{V}_c$. Let $\Phi = \{\phi_c(V_c)\}_{c\in C(G)}$. The Markov network (G, Φ) then defines the distribution:

$$P(\mathbf{v}) = \frac{1}{Z}\prod_{c\in C(G)}\phi_c(\mathbf{v}_c) \quad \text{(E.9)}$$

Here, Z is the normalization constant given by:

$$Z = \sum_{\mathbf{v}'}\prod \phi_c(\mathbf{v}'_c) \quad \text{(E.10)}$$

Appendix F

P-VALUES

The Perl module "GO::TermFinder" that was used to measure the quality of clusters determines whether the observed annotations for a group of genes are significant within the context of annotation for all genes within a reference set of genes. Following is an explanation taken from the original documentation of the module.

Let N be the total number of genes that are studied, and M be the number of genes that match a particular GO term out of these N genes. If x genes are observed with that annotation, in a sample of n genes, then the probability of that observation can be calculated, using the hypergeometric distribution as:

$$p = \frac{\binom{M}{x}\binom{N-M}{n-x}}{\binom{N}{n}} \tag{F.1}$$

To find a p-value, one should ask the question: What is the probability of having 5 or more out of 10 genes with this annotation, given that 42 out 6000 have it. This is what a p-value is - the chance of seeing your observation, or better, given the background distribution. This is calculated by summing the probabilities for 5 out of 10, 6 out of 10, 7 out of 10 etc. Thus the probability of seeing x or more genes with an annotation, out n, given that M in the population of N have that annotation, is:

$$p - value = \sum_{j=x}^{n} \frac{\binom{M}{j}\binom{N-M}{n-j}}{\binom{N}{n}} \tag{F.2}$$

Appendix G

THE ALGORITHM FOR UDIM

The program is composed of 7 Classes. Relationship between the classes can be seen at Figure G.1 and a detailed diagram of individual classes with their attributes and operations can be found in this section. Algorithms are given in pseudocode. Refer to the source code for the procedures whose pseudocode is not given.

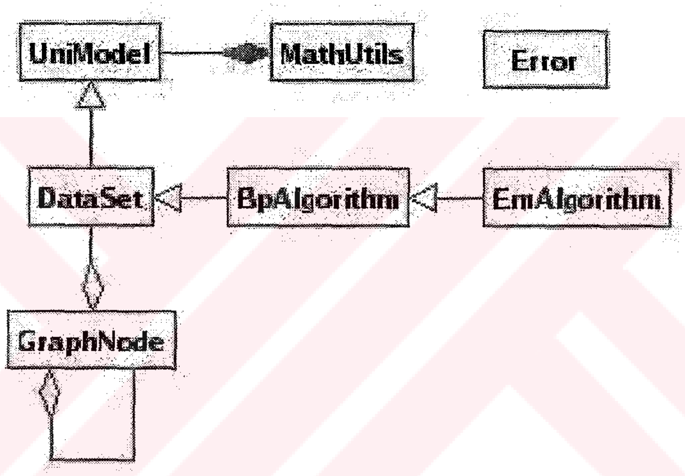


Figure G.1: Classes and their relations.

Algorithm 7 *EmModel()*

ht

```
1: Read the parameters from configuration file.
2: Initialize the $\theta$ values from file.
3: Read the Interaction and Expression data from file
4: Construct an EmAlgorithm Object em by the parameters and the data that is initialized.
5: em.RunEm()
```

Algorithm 8 *RunEm()*

```
1: if (!usingInitialClustering) then
2:     Initialize the means and standard deviations of CPDs randomly
3: end if
4: $numIterations = 0$
5: $isConverged = false$
6: while !$isConverged$ && $numIterations$++ != $numMaxIterations$ do
7:     EStep()
8:     $isConverged$ =MStep();
9: end while
10: Output beliefs to file
```

Algorithm 9 *EStep()*

```
1: StartBp()
```

Algorithm 10 *StartBp()*

```
1: InitBeliefArrs()
2: NormalizeBeliefs()
3: $numIterations = 0$
4: $isConverged = false$
5: while !$isConverged$ && $numIterations$ ++ != $numMaxIterations$ do
6:     RunBp();
7:     $isConverged$ = NormalizeMessages()
8: end while
9: UpdateBeliefs()
10: NormalizeBeliefs()
11: GetBeliefsInTenBase()
```

Algorithm 11 *InitBeliefArrs()*

```
1: for $gid = 0$ to $N$ do
2:   $node = interactionsArr[gid]$
3:   for $k = 0$ to $K$ do
4:     $node.probJointArr$ = ProbJointGcE($gid$, $k$)
5:   end for
6:   $node$.InitBeliefs($node.probJointArr$)
7:   for all $neighNode \in Neighbors\ of\ node$ do
8:     $node$.InitMessages(0.0)
9:   end for
10: end for
```

Algorithm 12 *RunBp()*

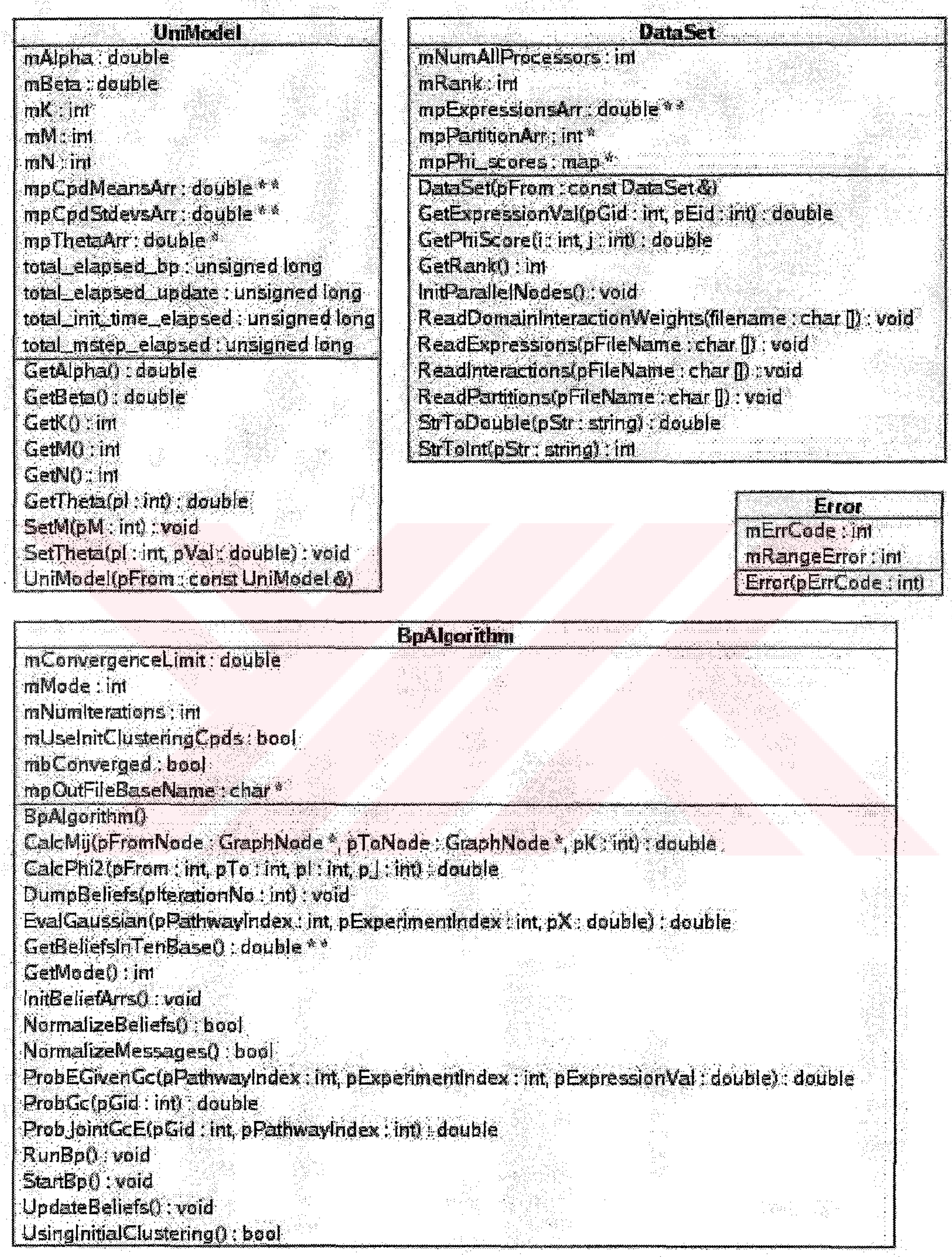
```
1: for $gid = 0$ to $N$ do
2:   GraphNode $node = interactionsArr[gid]$
3:   for all $neighNode \in Neighbors\ of\ node$ do
4:     $messageArr$ = $node$.GetMessagesFromNew()
5:     for $k = 0$ to $K$ do
6:       $messagesArr[k]$ = CalcMij($neighNode$, $node$)
7:     end for
8:   end for
9: end for
10: for $gid = 0$ to $N$ do
11:   GraphNode $node = interactionsArr[gid]$;
12:   $node$.SwapMessages();
13: end for
```

Algorithm 13 *CalcMij(GraphNode fromNode, GraphNode toNode, int pK)*

```
1: Let $msgArr$ be array of size $K$
2: for $k = 0$ to $K$ do
3:   $message = 0$
4:   for all $neighNode \in Neighbors\ of\ node$ do
5:     if ($neighNode$ != $toNode$) then
6:       $message$ += GetMessagesFromOld($neighNode$, $node$, $k$);
7:     end if
8:     $message$ += $fromNode$.GetBeliefsOld($k$);
9:     CalcPhi2($fromNode$, $toNode$, $k$, $pK$)
10:    $msgArr[k] = message$;
11:  end for
12: end for
13: return LogArrSum($msgArr$);
```

Algorithm 14 *MStep()*

```
1: Calculate Np Values
2: Check if $\theta$ values are converged
3: $sum_n p = \sum_{i=0}^{K} Np[i]$
4: for $k = 0$ to $K$ do
5:     $\theta_k = Np_i / sum_n p$
6: end for
7: for $k = 0$ to $K$ do
8:     for $m = 0$ to $M$ do
9:         double $xpj = 0$, $xpj2 = 0$;
10:        for $gid = 0$ to $N$ do
11:            $GraphNode\ node = interactionsArr[gid]$;
12:            $beliefArr = node.GetBeliefsOld()$
13:            $expressionVal = GetExpressionVal(gid, m)$;
14:            $xpj = beliefArr[k] * expressionVal$
15:            $xpj2 = beliefArr[k] * expressionVal^2$
16:        end for
17:        $mpCpdMeansArr[k][m] = xpj / Np[k]$
18:        $mpCpdStdevsArr[k][m] = xpj2/Np[k] - (xpj/Np[k])^2$
19:    end for
20: end for
```



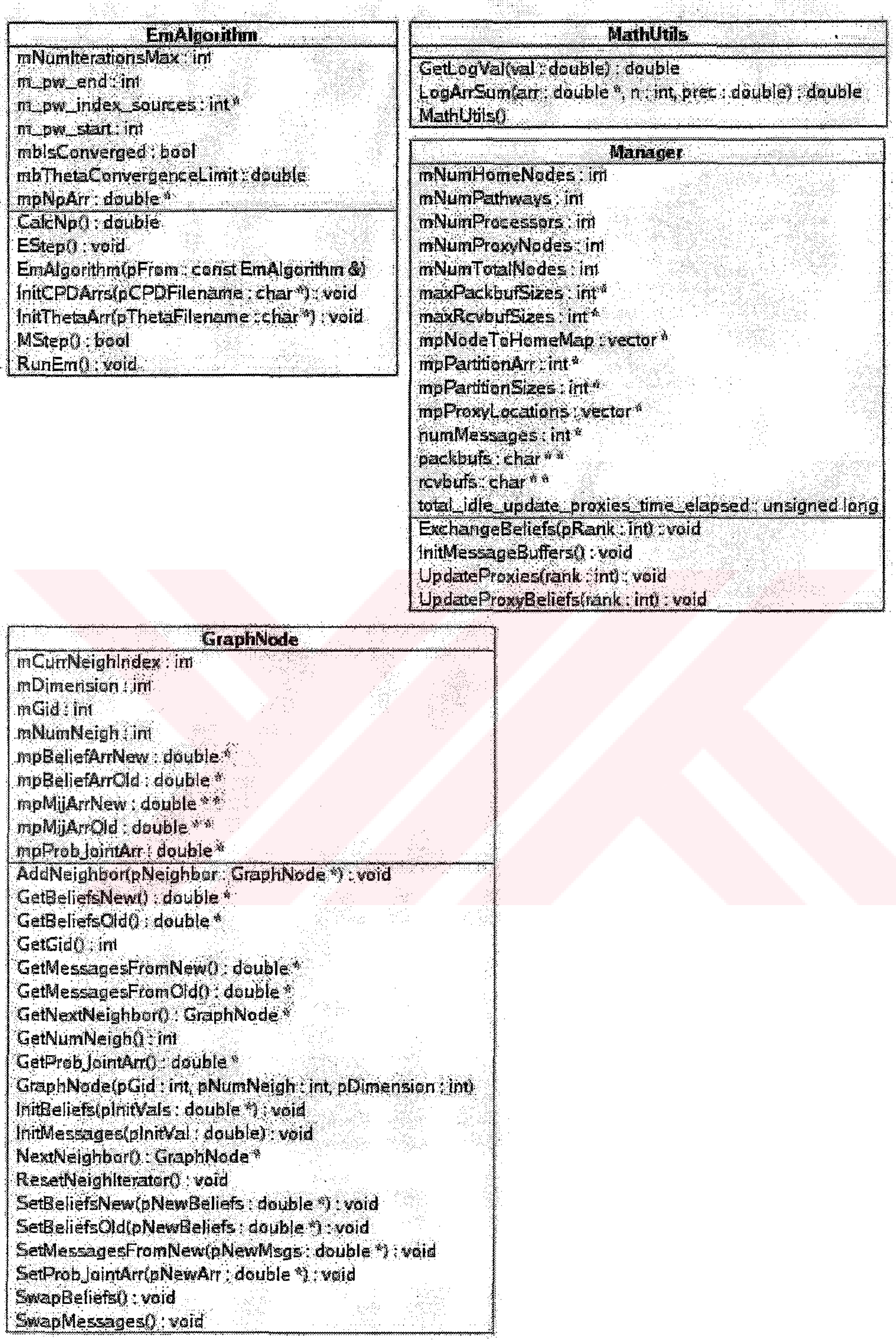
```
UniModel
mAlpha : double
mBeta : double
mK : int
mM : int
mN : int
mpCpdMeansArr : double * *
mpCpdStdevsArr : double * *
mpThetaArr : double *
total_elapsed_bp : unsigned long
total_elapsed_update : unsigned long
total_init_time_elapsed : unsigned long
total_mstep_elapsed : unsigned long
GetAlpha() : double
GetBeta() : double
GetK() : int
GetM() : int
GetN() : int
GetTheta(pI : int) : double
SetM(pM : int) : void
SetTheta(pI : int, pVal : double) : void
UniModel(pFrom : const UniModel &)
```

```
DataSet
mNumAllProcessors : int
mRank : int
mpExpressionsArr : double * *
mpPartitionArr : int *
mpPhi_scores : map *
DataSet(pFrom : const DataSet &)
GetExpressionVal(pGid : int, pEid : int) : double
GetPhiScore(i : int, j : int) : double
GetRank() : int
InitParallelNodes() : void
ReadDomainInteractionWeights(filename : char []) : void
ReadExpressions(pFileName : char []) : void
ReadInteractions(pFileName : char []) : void
ReadPartitions(pFileName : char []) : void
StrToDouble(pStr : string) : double
StrToInt(pStr : string) : int
```

```
Error
mErrCode : int
mRangeError : int
Error(pErrCode : int)
```

```
BpAlgorithm
mConvergenceLimit : double
mMode : int
mNumIterations : int
mUseInitClusteringCpds : bool
mbConverged : bool
mpOutFileBaseName : char *
BpAlgorithm()
CalcMij(pFromNode : GraphNode *, pToNode : GraphNode *, pK : int) : double
CalcPhi2(pFrom : int, pTo : int, pI : int, pJ : int) : double
DumpBeliefs(pIterationNo : int) : void
EvalGaussian(pPathwayIndex : int, pExperimentIndex : int, pX : double) : double
GetBeliefsInTenBase() : double * *
GetMode() : int
InitBeliefArrs() : void
NormalizeBeliefs() : bool
NormalizeMessages() : bool
ProbEGivenGc(pPathwayIndex : int, pExperimentIndex : int, pExpressionVal : double) : double
ProbGc(pGid : int) : double
ProbJointGcE(pGid : int, pPathwayIndex : int) : double
RunBp() : void
StartBp() : void
UpdateBeliefs() : void
UsingInitialClustering() : bool
```

Figure G.2: Full class diagram of the program.



```
EmAlgorithm
mNumIterationsMax : int
m_pw_end : int
m_pw_index_sources : int*
m_pw_start : int
mbIsConverged : bool
mbThetaConvergenceLimit : double
mpNpArr : double *
CalcNp() : double
EStep() : void
EmAlgorithm(pFrom : const EmAlgorithm &)
InitCPDArrs(pCPDFilename : char *) : void
InitThetaArr(pThetaFilename : char *) : void
MStep() : bool
RunEm() : void
```

```
MathUtils
GetLogVal(val : double) : double
LogArrSum(arr : double *, n : int, prec : double) : double
MathUtils()
```

```
Manager
mNumHomeNodes : int
mNumPathways : int
mNumProcessors : int
mNumProxyNodes : int
mNumTotalNodes : int
maxPackbufSizes : int *
maxRcvbufSizes : int *
mpNodeToHomeMap : vector *
mpPartitionArr : int *
mpPartitionSizes : int *
mpProxyLocations : vector *
numMessages : int *
packbufs : char * *
rcvbufs : char * *
total_idle_update_proxies_time_elapsed : unsigned long
ExchangeBeliefs(pRank : int) : void
InitMessageBuffers() : void
UpdateProxies(rank : int) : void
UpdateProxyBeliefs(rank : int) : void
```

```
GraphNode
mCurrNeighIndex : int
mDimension : int
mGid : int
mNumNeigh : int
mpBeliefArrNew : double *
mpBeliefArrOld : double *
mpMijArrNew : double * *
mpMijArrOld : double * *
mpProbJointArr : double *
AddNeighbor(pNeighbor : GraphNode *) : void
GetBeliefsNew() : double *
GetBeliefsOld() : double *
GetGid() : int
GetMessagesFromNew() : double *
GetMessagesFromOld() : double *
GetNextNeighbor() : GraphNode *
GetNumNeigh() : int
GetProbJointArr() : double *
GraphNode(pGid : int, pNumNeigh : int, pDimension : int)
InitBeliefs(pInitVals : double *) : void
InitMessages(pInitVal : double) : void
NextNeighbor() : GraphNode *
ResetNeighIterator() : void
SetBeliefsNew(pNewBeliefs : double *) : void
SetBeliefsOld(pNewBeliefs : double *) : void
SetMessagesFromNew(pNewMsgs : double *) : void
SetProbJointArr(pNewArr : double *) : void
SwapBeliefs() : void
SwapMessages() : void
```

Figure G.3: Full class diagram of the program (Continued).

BIBLIOGRAPHY

[1] Spellman,P.T., Sherlock,G., Zhang,M.O., Iyer,V.R., Anders,K., Eisen,M.B., Brown,P.O., Botstein,D. and Futcher,B. (1998) Comprehensive identification of cell cycle-regulated genes of the yeast Saccharomyces cerevisiae by microarray hybridization. Mol. Biol. Cell, 9(12), 3273 3297.

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

[3] Cheeseman,P. and Stutz,J. (1995) Bayesian classification (Auto- Class): Theory and results. In Advances in Knowledge Discovery and Data Mining. AAAI Press, Menlo Park, CA, pp. 153 180.

[4] Duda,R.O., Hart,P.E. and Stork,D.G. (2000) Pattern Classification. Wiley, New York.

[5] Kindeman, R. and Snell (1980) Markov Random Fields and Their Applications. American Mathematical Society, Providence, RI. Pearl,J.

[6] Pearl,J. Probabilistic Reasoning in Intelligent Systems. Morgan Kaufmann. (1988)

[7] Taskar B., Abbeel P., Koller D. Discriminative probabilistic models for relational data In Proc UAI, 2002

[8] Barber D., Probabilistic Modeling and Reasoning, Master's Course Notes, 2004-2004,

[9] Dempster, A.P., Laird,N.M. and Rubin,D.B. Maximum likelihood from incomplete data via the EM algorithm. J. Roy. Stat. Soc. B, 39, 1 39., 1977

[10] Mitchell Tom. Machine Learning, New York : McGraw-Hill, c1997.

[11] Xenarios I, Rice DW, Salwinski L, Baron MK, Marcotte EM, Eisenberg D (2000) DIP: The Database of Interacting Proteins. NAR 28:289-91

[12] Yedidia, J.S.; Freeman, W.T.; Weiss, Y., Understanding Belief Propagation and Its Generalizations, Exploring Artificial Intelligence in the New Millennium, ISBN 1558608117, Chap. 8, pp. 239-236, January 2003

[13] K. Murphy, Y. Weiss, and M. Jordan. Loopy belief propagation for approximate inference: An empirical study. In Proceedings of the Fifteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–99), pages 467–475, San Francisco, CA, 1999. Morgan Kaufmann Publishers.

[14] Mewes HW, Frishman D, Gruber C, Geier B, Haase D, Kaps A, Lemcke K, Mannhaupt G, Pfeiffer F, Schuller C, Stocker S, Weil B. MIPS: a database for genomes and protein sequences. Nucleic Acids Res. 2000 Jan 1;28(1):37-40.

[15] P E Hodges, A H McKee, B P Davis, W E Payne, and J I Garrels The Yeast Proteome Database (YPD): a model for the organization and presentation of genome-wide functional data. Nucleic Acids Res. 1999 January 1; 27 (1): 6973

[16] Babul J. Evolution and the structural domains of proteins. Arch Biol Med Exp (Santiago). 1987;20(3-4):333-41.

[17] Alex Bateman, Ewan Birney, Richard Durbin, Sean R. Eddy, Kevin L. Howe and Erik L. L. Sonnhammer The Pfam Protein Families Database Nucleic Acids Research, 2000, Vol. 28, No. 1 263-266

[18] Servant F, Bru C, Carrre S, Courcelle E, Gouzy J, Peyruc D, Kahn D (2002) ProDom: Automated clustering of homologous domains. Briefings in Bioinformatics. vol 3, no 3:246-251

[19] R. Agrawal, T. Irnielinski, and A. Swami: Mining Association Rules between Sets of Items in Large Databases. Proceedings of A CM SIGMOD, 207-216, May 1993

[20] Rakesh Agrawal, Ramakrishnan Srikant: Fast Algorithms for Mining Association Rules. Proc. 20th Int. Conf. Very Large Data Bases, VLDB 1994

[21] Christian Borgelt's Software Page: http://fuzzy.cs.uni-magdeburg.de/~borgelt/

[22] George Karypis and Vipin Kumar, Multilevel Algorithms for Multi-Constraint Graph Partitioning Technical Report TR 98-019 Department of Computer Science, University of Minnesota, 1998.

[23] George Karypis and Vipin Kumar, Multilevel k-way Partitioning Scheme for Irregular Graphs Journal of Parallel and Distributed Computing, 48(1):96-129, 1998.

[24] George Karypis and Vipin Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs SIAM Journal on Scientific Computing, 1998.

[25] Dolinski, K., Balakrishnan, R., Christie, K. R., Costanzo, M. C., Dwight, S. S., Engel, S. R., Fisk, D. G., Hirschman, J. E., Hong, E. L., Nash, R., Oughtred, R., Theesfeld, C. L., Binkley, G., Lane, C., Schroeder, M., Sethuraman, A., Dong, S., Weng, S., Miyasato, S., Andrada, R., Botstein, D., and Cherry, J. M. "Saccharomyces Genome Database" http://www.yeastgenome.org/

[26] The Gene Ontology Consortium, Gene Ontology: tool for the unification of biology. (2000) Nature Genet. 25: 25-29

[27] E. Segal, H. Wang, and D. Koller. Discovering molecular pathways from protein interaction and gene expression data Bioinformatics 2003 19: 264i-272i.

[28] A. Kocatas, A. Gursoy, R. Cetin-Atalay, Application of Data Mining Techniques to Protein-Protein Interaction Prediction, Lecture Notes in Computer Science, vol. 2869, pp. 316-323 (2003)

[29] T. Oyama, K. Kitano, K. Satou and T. Ito Extraction of knowledge on protein–protein interaction by association rule discovery Bioinformatics Vol. 18 no. 5 2002

[30] Joel R. Bock and David A. Gough Predicting protein-protein interactions from primary structure Bioinformatics, May 2001; 17: 455 - 460.

[31] Toshihide Ono, Haretsugu Hishigaki, Akira Tanigami, and Toshihisa Takagi Automated extraction of information on protein-protein interactions from the biological literature Bioinformatics, Feb 2001; 17: 155 - 161.

[32] See-Kiong Ng, Zhuo Zhang, and Soon-Heng Tan Integrative approach for computationally inferring protein domain interactions Bioinformatics, May 2003; 19: 923 - 929.

[33] Shawn M. Gomez, William Stafford Noble, and Andrey Rzhetsky Learning to predict protein-protein interactions from protein sequences Bioinformatics, Oct 2003; 19: 1875 - 1881.

[34] Jansen R, Yu H, Greenbaum D, Kluger Y, Krogan NJ, Chung S, Emili A, Snyder M, Greenblatt JF, Gerstein M. A Bayesian networks approach for predicting protein-protein interactions from genomic data. Science. 2003 Oct 17;302(5644):449-53

[35] Dirk Husmeier Sensitivity and specificity of inferring genetic regulatory interactions from microarray experiments with dynamic Bayesian networks Bioinformatics, Nov 2003; 19: 2271 - 2282.

[36] Nakaya,A., Goto,S. and Kanehisa,M. (2001) Extraction of correlated gene clusters by multiple graph comparison. Genome Informatics, 12, 44 53.

[37] Vert,J.-P. and Kanehisa,M. (2003) Graph-driven features extraction from microarray data using diffusion kernels and kernel CCA. Advances in Neural Information Processing Systems 15. MIT Press, Cambridge, MA.

[38] Y. Yamanishi, J.-P. Vert, A. Nakaya, and M. Kanehisa Extraction of correlated gene clusters from multiple genomic data by generalized kernel canonical correlation analysis Bioinformatics, Jul 2003; 19: 323 - 330.

[39] Stanley Letovsky and Simon Kasif Predicting protein function from protein/protein interaction data: a probabilistic approach Bioinformatics, Jul 2003; 19: 197 - 204.

[40] Minghua Deng, Zhidong Tu, Fengzhu Sun and Ting Chen, Mapping gene ontology to proteins based on protein-protein interaction data Bioinformatics, Apr 2004; 20: 895 - 902.

[41] Tom M. W. Nye, Carlo Berzuini, Walter R. Gilks, M. Madan Babu, and Sarah A. Teichmann Statistical analysis of domains in interacting protein pairs Bioinformatics, Advance Access published on October 27, 2004; doi: 10.1093/bioinformatics/bti086.

[42] Florian Sohler, Daniel Hanisch, and Ralf Zimmer New methods for joint analysis of biological networks and expression data Bioinformatics, Jul 2004; 20: 1517 - 1521.

[43] Y. Yamanishi, J.-P. Vert, and M. Kanehisa Protein network inference from multiple genomic data: a supervised approach Bioinformatics, Aug 2004; 20: i363 - i370.

[44] Eisen MB, Spellman PT, Brown PO and Botstein D. (1998). Cluster Analysis and Display of Genome-Wide Expression Patterns. Proc Natl Acad Sci U S A 95, 14863-8.

[45] Liping Ji and Kian-Lee Tan Mining gene expression data for positive and negative co-regulated gene clusters Bioinformatics, Nov 2004; 20: 2711 - 2718.

[46] David R. Bickel Robust cluster analysis of microarray gene expression data with the number of clusters determined biologically Bioinformatics, May 2003; 19: 818 - 824.

[47] Alexander V. Lukashin and Rainer Fuchs Analysis of temporal gene expression profiles: clustering by simulated annealing and determining the optimal number of clusters

[48] Feng Luo, Latifur Khan, Farokh Bastani, I-Ling Yen, and Jizhong Zhou A dynamically growing self-organizing tree (DGSOT) for hierarchical clustering gene expression profiles Bioinformatics, Nov 2004; 20: 2605 - 2617.

[49] Mario Medvedovic and Siva Sivaganesan Bayesian infinite mixture model based clustering of gene expression profiles Bioinformatics, Sep 2002; 18: 1194 - 1206.

[50] Gad Getz, Hilah Gal, Itai Kela, Daniel A. Notterman, and Eytan Domany Coupled two-way clustering analysis of breast cancer and colon cancer gene expression data Bioinformatics, Jun 2003; 19: 1079 - 1089.

[51] Peter N. Robinson, Andreas Wollstein, Ulrike Bhme, and Brad Beattie Ontologizing gene-expression microarray data: characterizing clusters with Gene Ontology Bioinformatics, Apr 2004; 20: 979 - 981

[52] Boris Adryan and Reinhard Schuh Gene-Ontology-based clustering of gene expression data Bioinformatics, Nov 2004; 20: 2851 - 2852.

[53] Daniel Hanisch, Alexander Zien, Ralf Zimmer, and Thomas Lengauer Co-clustering of biological networks and gene expression data Bioinformatics, Jul 2002; 18: 145 - 154.

[54] Hong, X., Liu, W. and Adamson, K. (2003) A parallel approach to evidence combination on qualitative Markov trees. Proceedings of the 4th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT), IEEE Press.

[55] Schena M. 2003. Microarray Analysis. Hoboken, NJ: Wiley-Liss. 630p.

[56] Bradley Coe and Christine Antler, Spot your Genes-An Overview of the Microarray, http://www.bioteach.ubc.ca/MolecularBiology/microarray/

[57] Ulrika Kahl. Nerve Cell Signaling.
http://www.hubin.org/facts/brain/texts/nervcell_signaling_se.html

[58] Gwen V. Childs. Smooth Muscle.
http://www.cytochemistry.net/microanatomy/muscle/smooth_muscle_2001.htm

[59] Solmaz Sobhanifar, Jiang Long.
The Yeast Two-Hybrid Assay An Exercise in Experimental Eloquence.
http://www.bioteach.ubc.ca/MolecularBiology/AYeastTwoHybridAssay/

[60] Amarda Shehu. Macromolecular Structure and Biomolecular Interactions. http://cnx.rice.edu/content/m11612/latest/

[61] The UK's National Measurement Laboratory. Validation of comparative fluorescence. www.npl.co.uk/biotech/ validfluo.html