THE UNIVERSITY OF CHICAGO

EXPLORING HIGHER-ORDER RELATIONSHIPS IN MICROBIAL PROTEIN NETWORKS

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCE DIVISION
IN CANDIDACY FOR THE DEGREE OF
MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

BY
RIDA ASSAF

CHICAGO, ILLINOIS
APRIL 01, 2015

Dedicated to

Epigraph Text

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

ABSTRACT

With the advancement in genome sequencing technology and the ever-growing set of fully sequenced genomes made available to researchers, genomic context analysis has attracted much interest. Investigating protein networks is a major focus of genome research, and the methods used to predict these interactions have been put on a quantitative basis. That is because they have been shown to be equally effective to experimental methods. Among the various methods that have been proposed to predict functional interactions between proteins based on the genomic context of their genes, we focus our attention to phylogenetic profiling methods which link proteins based on the high correlation between their phyletic distributions across a set of genomes. One of the goals of this research is to discover novel pathways in meta-genomic datasets. We show examples of current tools on the web that provide similar functionalities but are limited by some factors. We present our methods that alleviate some of these limitations and demonstrate their effectiveness by predicting a well-known pathway.

CHAPTER 1

INTRODUCTION

Studying the networks of molecular interactions that underlie cellular function is a major focus of genome research[2]. The methods used to predict these interactions have been put on a quantitative basis. That is because since they seem to be at least on an equal footing with genomics experimental data, the thing that was shown by a survey of experimentally confirmed predictions[8]. The sequencing of genomes from different species made it possible not only to get a consensus of all proteins but also to analyze their functions. [2, 14, 31, 6, 1, 37, 35]. The abundance of fully sequenced genomes and high throughput algorithms in comparative genomics created the opportunity to predict functionally linked proteins and to transfer the functionality of annotated proteins to un-annotated ones [43]. Context-based function prediction is complementary to homology-based function prediction[7, 9]. For a given protein, homology-based methods predict the molecular function of that protein, whereas context-based methods may predict a higher order function like what other proteins it interacts with or which pathway it plays a role in [7]. Other than direct physical binding, proteins may also interact indirectly. Examples of indirect interactions are sharing a substrate in a metabolic pathway, regulating other proteins transcriptionally, or forming multi-protein assemblies [38, 19, 16, 9]. The use of Protein networks is not limited to the increase of statistical power in human genetics[24, 4], but extends to aiding in drug discovery[28, 12], closing gaps in metabolic enzyme knowledge[10, 20], and predicting phenotypes and gene functions[33, 39, 13]. Among the various methods that have been proposed to predict functional interactions between proteins based on the genomic context of their genes are phylogenetic profiling methods and the domain fusion method[7]. Given a set of fully sequenced genomes, phylogenetic profiling methods link proteins based on the high correlation between their phyletic distribution across that set [32, 25, 41]. Whereas the domain fusion method identifies protein pairs that fuse into a single peptide in another organism [43, 5, 36]. Another method makes use of the observation that potential operons(genes that fall in each other's

proximity on genomes) encode functionally interacting proteins. [30, 21, 22, 18, 34, 40]. The method we adopt in this thesis is the phylogenetic profiling approach [2, 26, 27, 42] that detects functional relationships between proteins that show statistically similar patterns of presence or absence. Given a protein, we can search for its homologs across the different sequenced genomes in our database to determine the pattern describing its absence or presence. We can represent that pattern as a vector, storing 1 where a protein is present and 0 otherwise. This vector is referred to as the phylogenetic profile of that protein. Original implementations of the phylogenetic profile method considered proteins with similar profiles to be potentially functionally related [26]. Other ideas suggest that proteins may be linked if their profiles represented the negation of each other [17, 29]. These ideas represent a simple relation between two proteins in a cell, with the presence of one protein affecting the presence or absence of the other. The simplicity of such patterns might be limited to cases when two proteins are required to form a structural complex or when two proteins carry out sequential steps in an un-branched metabolic pathway. In order to describe the full complexity of cellular networks that involve branching, parallel, and alternate pathways, we expect the existence of higher order logic relationships involving a pattern of presence or absence of multiple proteins[2].

Other than inferring protein networks that might help us understand protein functions and figuring out what genes are associated with each other, among the many reasons why we care about genomic context analysis is hoping to devise a way that would aid in the discovery of novel pathways in meta-genomic datasets. We are also interested in having a way to estimate operons and their occurrence in given genomes, in addition to finding out protein or gene co-occurrences that are signatures of a given clade or genera. The implications of such discoveries would be tremendous and take us steps forward towards unmasking more mysteries in the biology domain. The rest of this paper is organized as follows, first we present some background of the statistical tools we used in this work. Then, we present some previous work in the literature, and show what of these questions they have

tackled and where their limitations lie. Then, we present our work that presents new robust methods to answer some of these questions, and alleviate some limitations the current work in the literature has. We present our results and findings along the way, and finish with some concluding remarks.

CHAPTER 2

BACKGROUND

One way to evaluate the correlation between two proteins is by calculating their conditional probability. The probability of seeing protein X given the probability of seeing protein Y is defined by:

$$P(X|Y) = \frac{P(X,Y)}{P(Y)}$$

In the event that P(X,Y) = P(X)P(Y) or $P(X|Y)$ = P(X), X and Y are considered to be independent. Else, X and Y are correlated.

The mutual information also provides a score for the co-occurrence of two genes. It is maximal when both genes occur in about 50% of the genomes (the individual entropies of the genes are maximal) and the genes occur always together (the combined entropy is minimal)[7].

Mutual Information can be defined as:

$$MI(X,Y) = H(X,Y) - H(X|Y) - H(Y|X)$$

Where H(X) and H(Y) are the marginal entropies, H($X|Y$) and H($Y|X$) are the conditional entropies, and H(X,Y) is the joint entropy of proteins X and Y. Another identity of the Mutual Information of proteins X and Y can be defined as:

$$MI(X,Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) log \left(\frac{p(x,y)}{P(x)(P(y)} \right)$$

Where p(x,y) is the joint probability distribution function of X and Y, and p(x) and p(y) are the marginal probability distribution functions of X and Y respectively.

Note that using genomic-context methods to design an experiment based on scores obtained using any of the two metrics mentioned above is not that trivial. This is because these methods may predict an interaction between proteins, but they do not predict the type of that interaction. For example, in the case of a metabolic pathway, these methods may tell us that a protein belongs to a certain pathway, but they do not say where in that pathway we can place the hypothetical protein. One way to increase the prediction specificity is to take the evolutionary conservation of a genomic-context pattern into account. For example, the more often that the genes occur in each other's proximity, the more likely that the proteins they encode functionally interact in the most direct way [8, 15]. Gene clusters are known to be prominent features of bacterial chromosomes. Demerec and Hartman postulated in 1959 that "regardless of how the gene clusters originated, natural selection must act to prevent their separation" and the "mere existence of such arrangements shows that they must be beneficial, conferring an evolutionary advantage on individuals and populations which exhibit them." [23].

CHAPTER 3

LITERATURE REVIEW

LAPP(logic analysis of phylogenetic profiles) is a computational approach for identifying detailed relationships between proteins on the basis of genomic data. Logic analysis of phylogenetic profiles identifies triplets of proteins whose presence or absence obey certain logic relationships. For example, protein C may be present in a genome only if proteins A and B are both present[2]. There are eight possible logic relationships relating two phylogenetic profiles to another. By calculating the uncertainty coefficients and using Mutual Information, the authors uncovered 750,000 new protein relationships for triplets where the individual pairwise uncertainty scores of the first two proteins described the third poorly but the logically combined profile described it well. (In other words: $H(Z|X) < 0.3$ and $H(Z|Y) < 0.3$ but $H(Z|(X,Y)) > 0.6$) where X,Y, and Z are the proteins of interest and H is the uncertainty score used in the Mutual Information equation. Different tools on the web provide different services for genomic context analysis. N-Browse takes a seed gene as an input and returns all those related to it. I2D takes one protein at a time and gives known and predicted mammalian and eukaryotic protein-protein interactions. BioPixie is a system for analysis and visualization of biological network predictions in S. Cerevisiaie. Given a protein or group of proteins as a query set (the expert-driven search component), a novel probabilistic algorithm considers the integrated pair-wise relationships to build a local process-specific network around the query proteins. Even though these tools provide some insight about protein functionality and correlation, we are usually interested in higher order relationships and more than pairwise correlation between them as previously mentioned. We are also interested in exploring protein networks from recovered meta-genomic data and prokaryotes. A tool that alleviates this limitation is GeneMANIA, that takes as input a query of seed genes(not limited to 1), and returns the genes that are related to them. One problem with these tools however, is that they only return the genes/proteins that are directly related to the query ones. We are often interested in the set of genes/proteins that

are indirectly related to our query genes/proteins as well, as this may uncover a broad set of new functional correlation and help us gain a better understanding of pathways. A tool that helps with that is VisANT, which starts from an initial gene/drug of interest, and uses exploratory navigation that allows you to walk through the interactions one by one. The interactions between genes are either physical or genetic, and the interactions between genes and drugs indicate the corresponding binding targets of the drugs(takes one at a time, and shows hierarchy). A tool that is somehow different is APID(Agile Protein Interaction Data-Analyzer), which helps analyze known information about protein-protein interactions that have been demonstrated by experimental methods. Some of these tools have limitations that are alleviated by others, like being able to provide only pairwise information, or only the directly related genes/proteins.

Now we discuss a tool that we believe offers a wider array of functions that are of interest in our research. STRING (search tool for recurring instances of neighboring genes) is a tool to retrieve and display the genes a query gene repeatedly occurs with in clusters on the genome[30]. STRING also retrieves genes that are indirectly(via other genes) related to the query gene. STRING takes a seed gene as input, gets all genes that match it, and treats these as seed genes in the next iteration. The process repeats until the number of iterations specified by the user is reached or until no new genes are found(convergence)[30]. Associations in STRING are provided with a probabilistic confidence score, which is derived by separately benchmarking groups of associations against the manually curated functional classification scheme of the KEGG database [11]. All scores and association data in STRING are precomputed[33]. Which might explain why it only supports queries for a single gene/protein at a time and thus does not offer a way to investigate higher dimension relationships.

Moreover, one limitation all these tools have in common is the lack of support for negated variables as input. In other words, sometimes we are interested in studying the implications of the absence of one protein on the presence/absence of others, especially when that protein is part of a larger input vector of interest. However, none of the mentioned tools or any

others that we know of provides this functionality.

As we mentioned before, one of the things we are interested in is discovering novel pathways. Two interesting tools that try to offer this functionality are LPAT(Linear Pathfinding with Atom Tracking) and SPABBATS. LPAT is an algorithm for finding atom conserving linear pathways in metabolic networks. LPAT takes as input a start compound, a target compound, the minimum number of atoms to be conserved, the number of pathways to return, k, and a special data structure, termed an atom mapping graph. LPAT is then guaranteed to return the k shortest pathways between the start and target compounds that conserve at least a given number of atoms. SPABBATS is an algorithm based on Boolean satisfiability (SAT) that computes alternative metabolic pathways between input and output species in a reconstructed network. In our journey of discovering novel pathways, we do not want to limit ourselves by having to specify the input and the target. What would be more interesting is having a map of different possible interactions that lead to an end result, and traversing that map to see what targets might be of interest and what input can lead to these targets. We find that to be a closer definition of novelty.

These tools and methods may help us infer protein networks, predict protein functions, gene associations, and novel pathways, but the help they offer is very limited. Besides, there is no obvious way in which we can use these tools to estimate operons and their occurrence in given genomes, or to find out protein or gene co-occurrences that are signatures of a given clade or genera. Thus, in the rest of paper, we present the approaches we took to alleviate some of these tools' limitations, and take a step further in answering a wider range of questions.

CHAPTER 4

METHODS

One way to think about the problem of getting the set of proteins that are potentially related to a query protein is by thinking about the very sparse matrix we need to traverse. The rows of that matrix will be the set of genomes that we have, and the columns will represent the proteins. An entry in row i and column j is set to 1 if protein j is found in genome i, and set to 0 otherwise. In the end, to find the proteins that are potentially related to the query protein, we traverse the matrix column wise and keep a count of how many times each of the proteins is found. We record the number of occurrences of every protein across the set of genomes, including the query protein(call it Q), in addition to the number of times every protein co-occurs with Q. In other words, we compute the frequency of co-occurrence of Q with all other proteins. In the end, one possible metric to get the potentially related proteins is to study the conditional probability of seeing each of the proteins given protein Q. The formula for conditional probability was given earlier, and can be summed up as the probability of seeing both proteins Q and some other protein, divided by the probability of seeing protein Q alone. If this probability is above a threshold set by the user, we would flag that protein as a protein of interest, and return it as a result representing a protein who is potentially functionally related to the query protein.

One can imagine how huge and sparse the matrix of interest can be. The data we currently have has 27,000 genomes , and there are over a million protein families. So to represent this whole matrix we need a lot of memory. There are tools that offer help in sparse matrix representation like Judy Arrays, but as the number of sequenced genomes increases, we need a more efficient way to tackle the situation, especially that we are expecting the number of sequenced genomes to rise up to 100,000 within the next couple of years. Thus, whatever methods we deploy have to be scalable and take into account the big data side of the application. Another memory issue stems when we consider storing the output. To put it simply, to store the conditional probability of each of the 1,000,000 proteins with all

others in a float(4 bytes), we need a total of 4 TeraBytes, and we will see how the number grows exponentially as we investigate more than just pairwise probabilities.

To tackle these issues and more, we decided to run the computations on disk. In other words, our results are not pre-computed, and are computed on-demand instead. We thought that this would be a better alternative, provided that we can achieve a high throughput. The simplest version of the program works as follows:

The program takes as input a query protein(Q) and a target protein(T). It then iterates through the database, which has the data clustered by genome. For every genome, we check whether or not Q or T occur in that genome, and if they co-occur in the genome. In the end, we have the frequency of occurrence and co-occurrence of Q and T across the set of all genomes, and we compute $P(T|Q)$. This task takes about 700ms for the dataset of 27,000 genomes using a Macbook Pro with 16GB RAM and a 2.5 GHz Intel Core(x4) i7. We found that to be a reasonable time to wait given the tradeoff in space that we are saving.

As we mentioned before, pairwise probabilities and relations are not enough to depict the complex molecular interactions that underlie cellular functions. Thus, to capture a broader image, we extended the program so that it supports taking a vector of proteins as an input(V), and a target protein (T), and returning $P(T|V)$ as the end result. The main difference in implementation is looping over all the proteins in the input vector as opposed to a single query protein when counting the occurrences in a genome and making sure to count a co-occurrence with the target protein only when all proteins in the input vector are witnessed in the same genome.

We described the basic protein interaction as having the presence of one protein affect the presence or absence of another. However, none of the tools we presented earlier takes the absence factor into consideration. Thus, to capture this, we allowed the input vector in our program to represent negated variables as well. This is due to the fact that we are often interested in the probability of seeing a protein given a set of proteins that may either be present or absent in the genomes.

The way the algorithm works is that it loads the database into two arrays, one storing the index of the last protein that belongs to each genome, and another saving the list of proteins of all genomes. A function is then called, that loops over the set of all genomes, and checks whether the input vector is fully encountered in each genome. In the case where the input vector is found, the program then increments a counter for every other protein that is seen in that genome. These counters represent the frequency of co-occurrence of every protein with the input proteins. In the end, the conditional probabilities are computed. It is obvious that the current state of the program requires significantly less memory than it would have if the entire matrix were to be computed. Having the program written this way allows us to answer more than a question about the probability of seeing a target protein given a query of proteins. We increased the functionality of the program by taking an extra parameter, which is the threshold set by the user. This way, since we already have all the frequencies of occurrences in the end, we iterate over all proteins calculating the respective conditional probabilities for each, and return those whose conditional probability falls above the specified threshold as the results, instead of limiting the program's functionality to find the probability of witnessing one target protein.

We are often interested in the predicting functionality of proteins whose corresponding genes are in certain proximity from each other on the genome. Thus, we had to take the distance between the corresponding genes into account. For this problem, the user can enter an additional parameter which is the distance threshold. We modified our computations in a way such that after the query protein is provided by the user, we pre-process the data to keep only the proteins that fall within the distance threshold away from the query protein. This helps us identify potential operons by computing sets of related proteins that are less than 300 base pairs away for example.

Another metric we can use to study study potentially related proteins is Mutual Information which was also discussed earlier. To study the difference between using conditional probability and using mutual information we compared the differences in the results that they

return. The program that uses mutual information is similar to the one we discussed earlier, and the main changes had to be made after acquiring the co-occurrence of the proteins. Specifically, to calculate the mutual information, we had to use the frequency of co-occurrence to compute the other factors in the summation presented in the equation(to cover all values every random variable(protein) can have, which are 0(absent) and 1(present)). This means that our program still runs in O(MN), where M is the number of genomes and N is the number of proteins. Here is an overview of how the program looks like when investigating the relationship between two proteins A and B:

```
- Process the database to obtain the phylogenetic profiles of the two proteins
- Process the phylogenetic profiles to compute the frequency of co-occurrence of:
    - A and B
    - ~A and B
    - A and ~B
    - ~A and ~B
- Use the frequencies to compute the mutual information between A and B
```

Mutual information is non-negative, and is bounded from above by 1 in this case. This is because the mutual information is bounded by the entropy of one random variable, which is log(number of values the variable can take), and since each variable can take value 0 or 1 in this case, the upper bound is $\log(2) = 1$. We use log base 2 to get the results in bits. A result of 0 bits means that the two variables are independent. We found out that mutual information sifts away most of the results that were returned by conditional probability. Probably because mutual information would not be affected by universal proteins. We will also show later how we used mutual information to recover a known classical pathway. Seeing that it was enough to answer our questions, we chose it as our metric to investigate the relationships between proteins.

As a naive test to our code, we computed the single probabilities of all proteins, sorted them in non-decreasing order, and plotted the results. The graph matched our expectations by being similar to the one we already had showing the frequency of occurrence of proteins across the genomes. We can see from the graph that most proteins barely show up in genomes while a few others are almost always there. This helped us design a more efficient approach to solving our next problem.

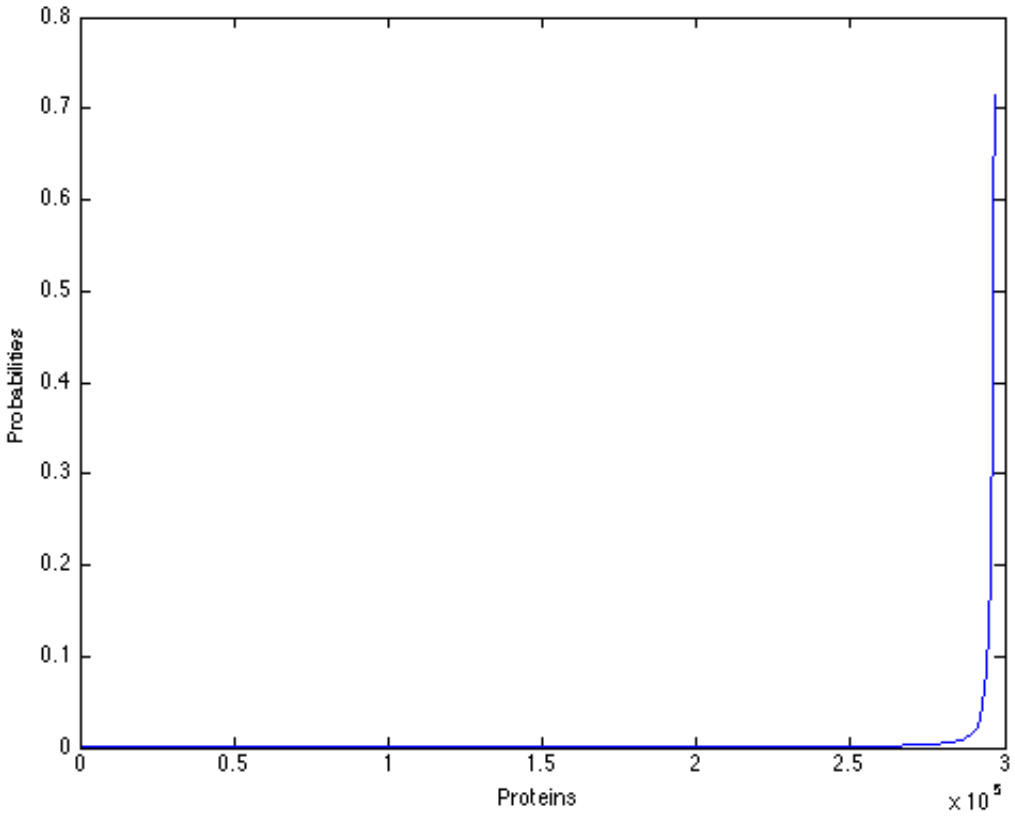

Figure 4.1: Protein's Probability Distribution

To calculate the probabilities of input vectors of different lengths with all other proteins, different parallel programming techniques were applied to improve the performance of our program. We adopted a distributed memory system approach when parallelizing our program by using the Message Passing Interface(MPI) to broadcast copies of the data to different processes, and distribute the set of proteins among the processes since these do not form any dependencies. In other words, the conditional probability of seeing one protein given the other is not affected by probabilities of other proteins. Seeing the probability of distribution of the data made it very clear that load balancing techniques have to be applied to achieve a better performance, so we made sure that our program scatters the workload appropriately across different processors.

We are often interested in building protein networks that involve proteins that are indirectly related to the query protein(s) of interest in addition to those that are directly returned by our program. To do this, we extended our program to accept a '*level*' as a parameter, which resembles how deep we want to look into the tree of indirectly related proteins. The program first gets all the proteins that are directly related by our query proteins, which will form the first level in our tree rooted by the query protein(s). Then, each of the proteins in the first level is considered these a new query protein. The program keeps running until the level specified by the user is reached, or until no new proteins are found. The number of proteins grows exponentially as we build more levels in our tree, but the program takes into account that a few proteins are mainly abundant by making sure not to redundantly re-compute results for a protein that has been considered a query protein at any point in its execution. This decreases the number of iterations needed for the program to converge.

Sometimes we are interested in doing our computations across genomes that belong to a certain genus, as opposed to considering the list of all sequenced genomes in our database. Thus, it is useful to be able to constraint the proteins in our search party to a specific genus. This also facilitates answering the question of what proteins are signatures of certain genera. To aid in this, our program takes as an extra parameter a genus of interest, and when parsing

the database to fill the input arrays, takes only the set of genomes that belong to that genus into consideration. To discover proteins that might be specific to a certain genus, or see the differences in the occurrences of proteins across different genera, we developed a program that takes two genera as additional input, finds the common proteins that occur across most of their genomes(the percentage of genomes the proteins must be seen in can also be passed as a parameter), and then outputs the pairs of proteins that maximize the difference between their conditional probabilities in the different genera. We will extend this approach to treat these as signatures specific to the genera. One reason we are interested in this is to solve the problem of classifying an unidentified genome and figuring out what genus it belongs to. So to solve this problem, we first extracted 10% of our database to treat as our training data. Then, using this data, we derived signatures for genera in the following manner: For each genus, we computed the occurrences of all its proteins across all the genomes that belong to that genus in our training dataset. Then, we created a signatures database, where each genus is followed by the probability of each of the proteins in that genus. We theorized that using these as our signatures would help us classify genomes into their respective genera. To put our theory to a test, we developed a program that, given a certain genome, goes over all the genera in the signatures database, and assigns a score that represents the likelihood of that genome belonging to that genus. The score is computed in a way that can be thought of as a longest common subsequence, where for every protein in the genome, the probability of witnessing that protein in the genus is added to the score, and for every protein that is in the genome but not present in the genus, a penalty is applied. The maximum score is kept across all genera, breaking ties according to the number of proteins that are in common between the genome and the genera, and the genus with the maximum score is returned as the genus to whom our input genome most likely belongs to. We tested this program on the training data, and it achieved 99.8% accuracy. Furthermore, we tested the program on our whole database of 27,000 genomes, and it achieved 98% accuracy. It also classified all of unclassified genomes that we had in our database into their corresponding genera with very

high confidence scores.

The problem of discovering novel pathways can be modeled as a very computationally expensive problem. If we limit the set of all possible input vectors of interest to be of length 6, the computational requirements dictated by the problem are enormous. With more than a million different proteins, the set of all possible input vectors of length 6 has cardinality $\binom{1,000,000}{6}$ which is $\quad 10^{36}$. If our program takes 10^{-24} ms to run as opposed to its current runtime of 700ms for such input length, it would need 30 years to compute all results. If we decide to store results whose probabilities lie above a certain threshold, we still wouldn't get over the fact that to store $10^{-21}\%$ of the results we would get, we need 4 PetaBytes. The situation gets much worse when we allow our input vector to have negated variables.

To get around this problem, we followed a different approach. In the paper [3], the authors identify a metabolic alternative to the classical Mevalonate pathway. To test whether our program can detect any of the two pathways, we used Mevalonate Kinase(MVK) as an input and returned the four proteins with maximum mutual information with MVK. We we thrilled to find that the input MVK together with these four returned proteins form the classical pathway mentioned in the paper. We are currently working on figuring out the FIGFAMs corresponding to the proteins in the alternative pathway to check whether or not our program detects that as well. Considering the fact that this may be enough to verify known pathways and detect novel ones, we decided to store the set of all vectors of length 6 representing every possible protein with the 5 other proteins whose mutual information is maximal with it. We are hoping that this could serve as a start to test the possibility of detecting novel pathways by treating these vectors as candidate ones. As a side application, seeing that mutual information is returning functionally related proteins, we matched a small subset of proteins whose functions were identified as 'hypothetical figfams' to other proteins with well known functions.

We are developing a tool that will be deployed as a web service to visualize our results. The tool takes a vector of proteins as an input, along with possible extra parameters like the threshold, distance threshold, specific genus, and the number of levels of indirectly related proteins. It then does the necessary computations and displays the results as a graph. If no threshold was specified, the program returns the 10 proteins whose mutual information with the query protein is maximal and non-zero(to ensure dependence).

We use the Mevalonate Kinase pathway mentioned earlier as an example.

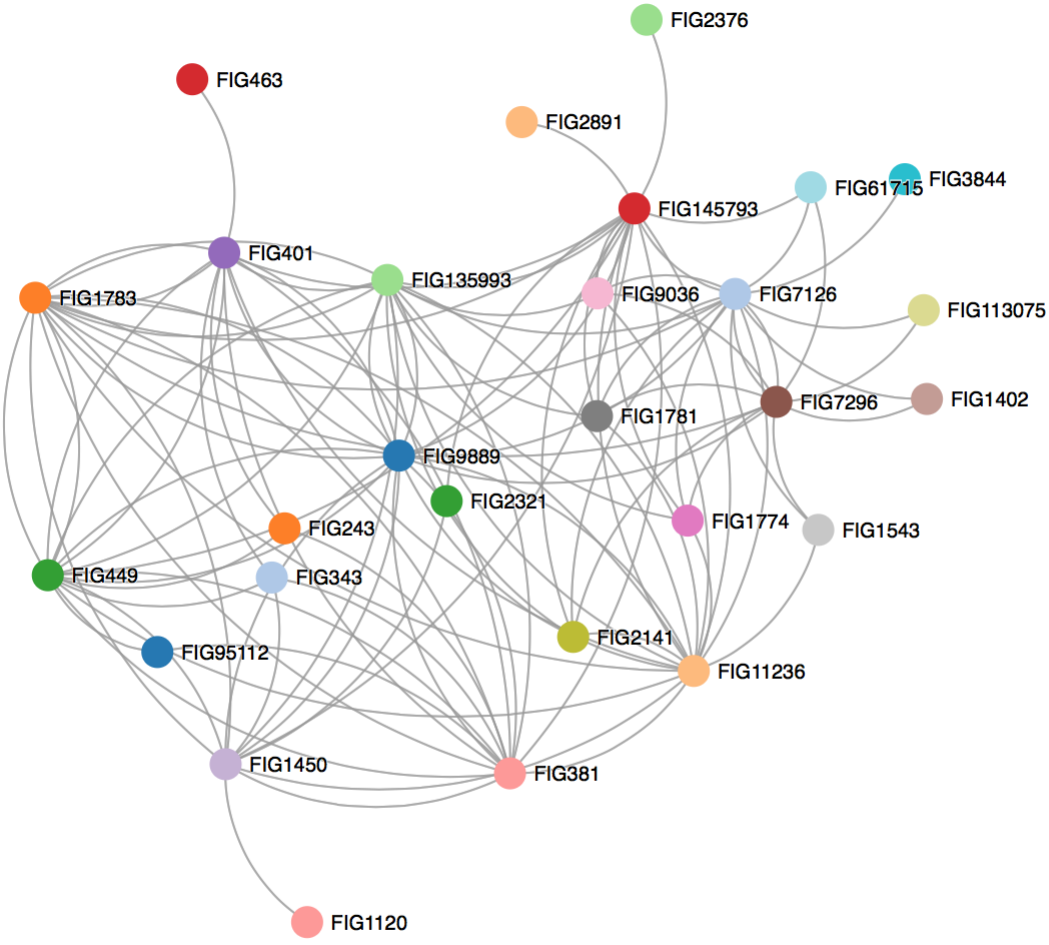

Figure 4.2: Protein Network with MVK as input

The image above shows the resulting graph when we provide FIG9889(MVK) as a query protein to our program with the '*level*' parameter set to 2, and without specifying a threshold, a genus, nor a distance threshold. It thus shows up to 10 related proteins to MVK, and up to 10 related proteins to each of those. We use curved edges to better show the connections among the nodes as the graph gets more dense. The nodes of the graph are colored according to the functions of the subsequent proteins. If two proteins have the same function, their nodes will have the same color. The thickness of an edge depends on the value of the correlation between the nodes. Thicker edges mean that the proteins are more strongly correlated(their mutual information is higher) than thinner proteins with thinner edges. To check the direct neighborhood of a protein of interest, we can click on a node to highlight it and its immediate neighbors, which would also show the corresponding mutual information value between the highlighted node and its neighbors. As an example, we highlight the FIG9889 node representing MVK, which shows the classical Mevalonate Kinase pathway as part of the highlighted neighbors.

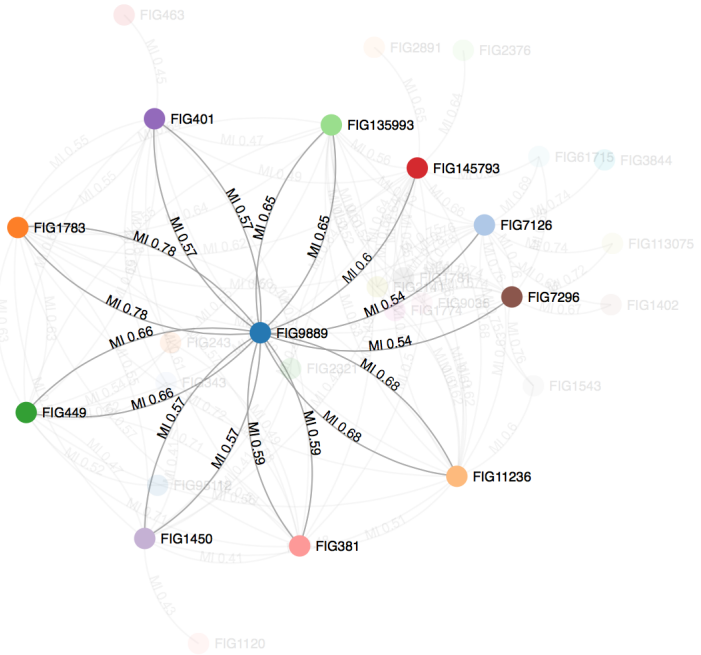

Figure 4.3: Protein Network with MVK highlighted

CHAPTER 5

CONCLUDING REMARKS

We presented a new program for genomic context analysis that extends the functionality of existing tools by allowing the input to be a vector of proteins with the possibility of investigating the absence of proteins and not just their presence. The program returns all proteins that are directly and indirectly related to the query proteins up to a level specified by the user(or until it converges). Aside from the threshold that specifies when two proteins are considered to be potentially related, the user can also specify the maximum distance the proteins can be away from the query protein to be considered potentially related to it. The user may also choose to specify the genus to which the program should limit its computations to when considering the whole set of sequenced genomes in our database. We demonstrated the accuracy of our program by using it to detect the known classical pathway of Mevalonate Kinase. We presented the interface of the tool that will incorporate our program's functionality. The tool takes the input from the user and returns the results in the form of a graph representing a protein network.

We also devised an application that classifies genomes across the classes of genera. The application classified our complete set of genomes in the database with 98% accuracy.
We present fragments of the code we used to develop our program in the appendix.

Appendices

APPENDIX A

```
1
2  for (i = 0; i < number_of_genomes; i++) {
3          genome_score = 0;
4          //record if any of the vector elements have been unlawfully seen
5          oops = 0;
6          //index of last protein for this genome
7          endg = genome_indices[i];
8          for (j = startg; j < endg; j++) {
9              current_protein = proteins[j];
10             if (current_protein >=0) {
11                 //count frequency of each protein
12                 countProteins[current_protein]++;
13                 //loop over input vector
14                 for (k = 0; k < Vlen; k++) {
15                    //if element of input vector is seen
16                     if (current_protein == FFV[k]) {
17                         //if we are counting its presence
18                         if (booleans[k]) {
19                                 /*count number of elements
20                                  in vector present in this genome*/
21                             genome_score++;
22                         }
23                         else { /*if we are counting its absence
24                                  the proteins was not supposed
25                                   to be seen in this form*/
26                             oops = 1;
```

```
27                          }
28                      }
29                  }
30              }
31          }
32          if (!oops) { //no protein was unlawfully seen
33              for (n = 0; n < Vlen; n++) {//loop over input vector
34                  //if we are counting this protein's absence
35                  if (!booleans[n]) {
36                          // element was rightfully absent from genome
37                      genome_score++;
38                  }
39              }
40              //If all input vector was seen in the genome
41              if (genome_score == Vlen) {
42                  countV++; //count frequency of occurrence of the vector
43                  //for all other proteins that were in that genome
44                  for (n = startg; n < endg; n++) {
45                      if (proteins[n] >= 0) {
46                          //count frequency of co-occurrence with the vector
47                          countVBs[proteins[n]]++;
48                      }
49                  }
50              }
51          }
52          startg = endg;
53      }
```

The program loops over all genomes, counting the frequency of occurrence of every protein, the frequency of occurrence of the input vector as a whole, and the frequency of co-occurrence of every vector with the other proteins.

APPENDIX B

```
1     for (i = 0; i < max_protein; i++) {
2         //P(V,B): probability of V and protein(call it B)
3         probability_VB = ((double)countVBs[i])/number_of_genomes;
4         //frequency of co-occurrence of !V(not V) and protein
5         countnVBs[i] = countBs[i] - countVBs[i];
6         //P(!V,B): probability of !V and protein
7         probability_nVB = ((double)countnVBs[i])/number_of_genomes;
8         //frequency of co-occurrence of V and !protein
9         countVnBs[i] = countV - countVBs[i];
10        //P(V,!B): probability of V and !protein
11        probability_VnB = ((double)countVnBs[i])/number_of_genomes;
12        //frequency of co-occurrence of !V and !protein
13        countnVnBs[i] = number_of_genomes -
14                        (countVBs[i] + countnVBs[i] + countVnBs[i]);
15        //P(!V,!B): probability of !V and !protein
16        probability_nVnB = ((double)countnVnBs[i])/number_of_genomes;
17
18        probability_B_given_V = probability_VB/probability_V; //P(B|V)
19        probability_B = ((double)countBs[i])/num_genomes; P(B)
20
21
22
23
24
25
```

```
26          //Compute different factors in Mutual Information(MI) summation
27          if (probability_VB > 0 && probability_V > 0 && probability_B > 0) {
28              double px = probability_VB + probability_VnB;
29              double py = probability_nVB + probability_VB;
30              double denomenator = px * py;
31              MI11 = probability_VB *
32                      (log(probability_VB/denomenator)/log(2.0));
33          }
34          else {
35              MI11 = 0;
36          }
37          if (probability_nVB > 0 && probability_V < 1 && probability_B > 0) {
38              double px = probability_nVB + probability_nVnB;
39              double py = probability_nVB + probability_VB;
40              double denomenator = px * py;
41              MI01 = probability_nVB *
42                      (log(probability_nVB/denomenator)/log(2.0));
43          }
44          else {
45              MI01 = 0;
46          }
47
48
49
50
51
52
```

```
53
54          if (probability_VnB > 0 && probability_V > 0 && probability_B < 1) {
55              double px = probability_VnB + probability_VB;
56              double py = probability_nVnB + probability_VnB;
57              double denomenator = px * py;
58              MI10 = probability_VnB *
59                      (log(probability_VnB/denomenator)/log(2.0));
60          }
61          else {
62              MI10 = 0;
63          }
64          if (probability_nVnB > 0 && probability_V < 1 && probability_B < 1) {
65              double px = probability_nVnB + probability_nVB;
66              double py = probability_nVnB + probability_VnB;
67              double denomenator = px * py;
68              MI00 = probability_nVnB *
69                      (log(probability_nVnB/denomenator)/log(2.0));
70          }
71          else {
72              MI00 = 0;
73          }
74          //MI(V,B): Mutual Information of V and B
75          MI = MI00 + MI01 + MI10 + MI11;
76  }
```

The program then checks one protein at a time, calculating its mutual information with the input vector. In the end, it returns the top 10 proteins, or those whose mutual information

falls above a certain threshold pre-specified by the user. These proteins will serve as query proteins in subsequent levels if the user chose looking at proteins that are indirectly related to the original query protein.

REFERENCES

[1] P. M. Bowers et al. Prolinks: a database of protein functional linkages derived from coevolution. *Genome Biol.*, 2004.

[2] P. M. Bowers et al. Use of logic relationships to decipher protein network organization. *Science*, 2004.

[3] N. Dellas et al. Discovery of a metabolic alternative to the classical mevalonate pathway. *eLIFE*, 2013.

[4] M. Emily et al. Using biological networks to search for interacting loci in genome-wide association studies. *Eur. J. Hum. Genet.*, 2009.

[5] A. J. Enright et al. Protein interaction maps for complete genomes based on gene fusion events. *Nature, 402:86-90*, 2001.

[6] L. Giot et al. A protein interaction map of drosophila melanogaster. *Science*, 2003.

[7] M. Huynen et al. Predicting protein function by genomic context: Quantitative evaluation and qualitative inferences. *Genome Res.*, 2000.

[8] M. Huynen et al. Function prediction and protein networks. *Current Opinion in Cell Biology*, 2003.

[9] M. A. Huynen et al. Exploitation of gene context. *Curr. Opin. Struct. Biology*, 2000.

[10] S. C. Janga et al. Network-based function prediction and interactomics: The case for metabolic enzymes. *Metab. Eng.*, 2010.

[11] M. Kanehisa et al. Kegg for representation and analysis of molecular networks involving diseases and drugs. *Nucleic Acids Res., 38, D355-D360*, 2010.

[12] E. Klipp et al. Biochemical network-based drug-target prediction. *Curr. Opin. Biotechnol.*, 2010.

[13] K. Lage et al. A human phenome-interactome network of protein complexes implicated in genetic disorders. *Nat. Biotechnol., 25, 309-316*, 2007.

[14] S. Li et al. A map of the interactome network of the metazoan c. elegans. *Science*, 2004.

[15] Huynen MA and Snel B. Exploiting the variations in the genomic associations of genes to predict pathways and reconstruct their evolution. *In Frontiers in Computational Genomics*, 2000.

[16] E. M. Marcotte. Computational genetics: finding protein function by nonhomology methods. *Curr. Opin. Struct. Biol., 10, 359-365*, 2000.

[17] E. Morett et al. Gecont: gene context analysis. *Nat. Biotechnol.*, 2004.

[18] A.R. Mushegian and E.V. Koonin. *Trends Genet., 12, 289-290*, 1996.

[19] Galperin M.Y. and Koonin E.V. Who's your neighbor? new computational approaches for functional genomics. *Nat. Biotechnol., 18, 609-613*, 2000.

[20] J.D. Orth and B.O. Palsson. Systematizing the generation of missing metabolic knowledge. *Biotechnol. Bioeng. 107, 403-412*, 2010.

[21] R. Overbeek et al. Use of contiguity on the chromosome to predict functional coupling. *In Silico Biol.*, 1998.

[22] R. Overbeek et al. The use of gene clusters to infer functional coupling. *Proc. Natl Acad. Sci. USA*, 1999.

[23] R. Overbeek et al. The use of gene clusters to infer functional coupling. *Proc. Natl. Acad. Sci. USA Vol. 96, pp. 2896-2901*, 1999.

[24] K.A. Pattin and J.H. Moore. Role for protein-protein interaction databases in human genetics. *Expert Rev. Proteomics*, 2009.

[25] M. Pellegrini et al. Assigning protein functions by comparative genome analysis: Protein phylogenetic profiles. *Proc. Natl. Acad. Sci. USA, 96:4285-4288*, 1999.

[26] M. Pellegrini et al. Assigning protein functions by comparative genome analysis: protein phylogenetic profiles. *Proc. Natl. Acad. Sci. U.S.A*, 1999.

[27] M. Pellegrini et al. Computational method to assign microbial genes to pathways. *J. Cell. Biochem. Suppl.*, 2001.

[28] A. Pujol et al. Unveiling the role of network and systems biology in drug discovery. *Trends Pharmacol. Sci.*, 2010.

[29] E. M. Marcotte. S. V. Date. Discovery of uncharacterized cellular systems by genome-wide analysis of functional linkages. *Nat. Biotechnol.*, 2003.

[30] B. Snel et al. String: a web-server to retrieve and display the repeatedly occurring neighbourhood of a gene. *Nucleic Acids Res.*, 2000.

[31] M. Strong et al. Inference of protein function and protein linkages in mycobacterium tuberculosis based on prokaryotic genome organization: a combined computational approach. *Genome Biol.*, 2003.

[32] Date S.V. and Marcotte E.M. Discovery of uncharacterized cellular systems by genome-wide analysis of functional linkages. *Nat. Biotechnol., 21(9):1055-1062*, 2003.

[33] D. Szklarczyk et al. The string database in 2011: functional interaction networks of proteins, globally integrated and scored. *Nucleic Acids Res.*, 2011.

[34] J. Tamames et al. *J. Mol. Evol., 44, 66-73*, 1997.

[35] A. H. Tong et al. Global mapping of the yeast genetic interaction network. *Science*, 2004.

[36] A. Vazquez et al. Global protein function prediction from protein-protein interaction networks. *Nat. Biotechnol., 21(6):697-700*, 2003.

[37] C. von Mering et al. String: a database of predicted functional associations between proteins. *Nucleic Acids Res.*, 2003.

[38] C. von Mering et al. String: a database of predicted functional associations between proteins. *Nucleic Acids Res.*, 2003.

[39] P.I. Wang and E.M. Marcotte. It?s the machine that matters: Predicting gene function and phenotype from protein networks. *J. Proteomics, 73, 2277-2289*, 2010.

[40] H. Watanabe et al. *J. Mol. Evol., 44, S57-S64*, 1997.

[41] J. Wu et al. Identification of functional links between genes using phylogenetic profiles. *Bioinformatics, 19(12):1524-1530*, 2003.

[42] J. Wu et al. Identification of functional links between genes using phylogenetic profiles. *Bioinformatics*, 2003.

[43] J. Wu et al. Deciphering protein network organization using phylogenetic profile groups. *Genome Informatics*, 2005.