PIPE3: A MASSIVELY PARALLEL PROTEIN-PROTEIN INTERACTION PREDICTION ENGINE

by
Andrew Schoenrock

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of

MASTER OF COMPUTER SCIENCE

CARLETON UNIVERSITY
Ottawa, Ontario

Your file Votre référence
ISBN: 978-0-494-71554-3
Our file Notre référence
ISBN: 978-0-494-71554-3

Abstract

Protein-protein interactions play a key role in many human diseases. A detailed knowledge of protein-protein interactions will accelerate the drug discovery process as well as reduce overall drug development costs. Biologists have a variety of methods to determine protein-protein interactions in the labratory, however these approaches are generally expensive, labour intensive, time consuming and have significantly high error rates. The Protein-protein Interaction Prediction Engine (PIPE) is a computational approach to prediction protein-protein interactions. The first version of PIPE proved that the underlying hypothesis that PIPE is built upon had merit. The second version of PIPE was a vast improvement over the first version and proved that a prediction over all possible yeast protein pairs was possible, however still had room for major improvements. The two major limitations of the PIPE2 implementation was that it was not very efficient with its memory use and it did not scale well due to a lack of a real parallel architecture.

This thesis presents PIPE3, a new version of PIPE designed to overcome the downfalls of the earlier versions. A new parallel architecture consisting of a mixed master/slave and all-slaves model was designed and implemented to address these issues. This overall approach exploits parallelism on two levels, the first by running multiple PIPE threads on a given cluster node and secondly by running many of these processes on different cluster nodes, all being fed work in an on-demand fashion. PIPE3's parallel architecture performed well overall. The all-slaves portion of the parallel architecture produced a significant speedup and the master/slave portion of the parallel architecture resulted in a linear speedup when more cluster nodes were used. PIPE3 was then used to produce some significant scientific results. These include a proteome-wide scan on the *Caenorhabditis Elegans* and *Homo Sapiens* organisms, which also resulted in the largest threaded calculation ever run at the High Performance Computing Virtual Laboratory (HPCVL).

Acknowledgements

First and foremost I would like to acknowledge my gratitude to my supervisor Dr. Frank Dehne from whom I received invaluable guidance and advice while pursuing this work. Without his help this work would not have been possible.

Secondly I would like to thank Dr. Ashkan Golshani, Dr. James Green, Dr. Sylvain Pitre, Dr. Maria DeRosa and everyone else in the Carleton Bioinformatics group. After joining the group shortly after starting graduate school, I was instantly inspired to research a Bioinformatics topic. Without the ability to ask basic biology questions in a relaxed environment, I never would've been able to even start my research in this field and for that I owe them a debt of gratitude.

Thirdly I would like to thank Dr. Hartmut Schmider, Bao Zhang, Chris MacPhee and everyone else on the HPCVL support team at Queens University. Without their insight, guidance and patience the development, deployment and eventual running of PIPE3 would've been a much more difficult endeavor.

Finally, I would like to thank my family and and my girlfriend for their unending support of my academic pursuits.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation and Background

Over the past decade, in the post-genomic era, protein-protein interactions have become a main interest of many. One of the major goals of functional genomics, biology, bioinformatics as well as other related fields is the elucidation of individual protein-protein interactions and complete interaction networks within model organisms as well as the understanding of various proteins' functions [23, 35, 4, 21]. It is the millions of proteins acting together, carrying out the numerous important biological processes, that make up the cellular machinery of the cell [17] and the overall state of a cell is largely determined by its protein interactions [23]. Identifying the proteins a given protein interacts with is a direct indicator of that protein's function and the complete elucidation of a protein-protein interaction map of the cell would be a large contributor to understanding the overall biology of the cell [26]. In general, a deep understanding of protein-protein interactions and their subsequent interaction networks within a cell are expected to have significant implications and have many practical applications, such as drug design [30]. Protein-protein interactions play a key role in many human diseases [2] by either involving endogenous proteins, proteins from pathogens, or both [34]. A wide range of approaches have been used to inhibit protein-protein interactions as a therapy for disease, however it is clear that a detailed knowledge of protein-protein interactions will only further this research [34]. Not only will this understanding give researchers a clearer view of the inner workings of human disease, but it will also accelerate the drug discovery process as well as reduce drug development costs [29].

Biologists have a variety of methods to determine protein-protein interactions in the laboratory. The Yeast Two-Hybrid based techniques were first introduced in 1989 [9].

The process takes two query proteins, X and Y, and fuses a DNA binding domain to X (the bait) and an activation sequence domain to Y (the prey). If X and Y interact it is expected to result in the assembly of the two domains, which leads to the transcription of a reporter gene. This reporter gene can catalyze and result in a noticeable colored compound to signal that X and Y did in fact interact [23]. This approach has been used to produce large scale scans of the *Saccharomyces cerevisiae* organism [39, 15], more commonly known as yeast, as well as other organisms. However, these scans contain very little overlap in their results making a single two-hybrid scan's results almost unreproducible. The false positive rates can range anywhere from 44% to 91% [22] and the false negative rates can range from 28% to 51% [14]. These error rates contribute to the fact that the two-hybrid approaches can only reliably map out a fraction of the overall proteome for a given organism [13]. Another method to determine protein-protein interactions is Tandem Affinity Purification (TAP). The basic idea behind the process is as follows. A TAP tag is associated with the protein of interest and placed on a column with various substances to aid in the experiment. Assorted other proteins are then introduced to the column and then the column is put through a purification process to remove any non-interacting proteins. The protein complex left on the column is made up of a series of proteins that interact with the protein of interest. These proteins are then analyzed and identified by a mass spectrometer [23]. Although this process is significantly more expensive than two-hybrid approaches, it allows for a wider scan of protein interactions in one experiment. However, the purification process could remove proteins involved in weaker interactions, meaning that the interaction would not be reported [23] which contributes to an overall error rate of roughly 15% [11]. Various other methods exist and continue to get better over time, however all of the biological methods to determine protein-protein interactions are plagued by common downfalls. In general, these approaches are expensive, labour intensive, time consuming and have significantly high error rates [36].

More recently, various groups of computer scientists have been working with biologists and bioinformaticians to come up with methods to predict protein-protein interactions. These methods generally fall loosely into the following categories: approaches

based on genomic information, approaches based on evolutionary relationships, approaches based on 3D protein structure, approaches based on protein domains and, finally, approaches based on primary protein structure [10, 30]. Overall, these approaches try to take advantage of the wealth of protein-protein interaction data that is currently available to make predictions on novel protein-protein interactions or to confirm laboratory experiments, some with more success than others.

1.2 Previous Work: The Protein-protein Interaction Prediction Engine (PIPE) Versions 1 & 2

The Protein-protein Interaction Prediction Engine, or PIPE for short, is a computational approach to prediction protein-protein interactions that falls squarely into the category of approaches based on primary protein structure. In other words, all PIPE needs to make a prediction on two query proteins is the query proteins' primary structures as well as a list of known protein-protein interactions with their primary structures. This is a significant advantage over many other approaches which need much more information to make a prediction. Also, due to PIPE's relative simplicity, it can be applied to any and all possible protein pairs in a given organism, which many other approaches can not do. The first version of PIPE was published in July of 2006 and proved that the underlying hypothesis that PIPE is built upon had merit [31]. The second version of PIPE was published two years later and was a vast improvement over the first version. Due to a few major algorithm improvements, the sensitivity and the overall running time was so drastically improved that a prediction over all possible yeast protein pairs was possible [32]. This was a first in the biology world and showed PIPE's true potential.

Even though PIPE2 was a vast improvement over the original version of PIPE, there was still room for major improvements. In particular there were two major limitations of the PIPE2 implementation. The first was that it wasn't very efficient with its memory use, which caused it to crash when trying to make predictions on protein pairs of more complex organisms. The second had to do with an overall poor load balancing scheme which meant it could not scale well to larger and larger clusters

or supercomputers easily. These two issues would have to be addressed if PIPE was going to be applied to larger and more complex organisms.

1.3 Summary of Contributions

This thesis presents PIPE3, a new version of PIPE designed to overcome the downfalls of the earlier versions. A new parallel architecture consisting of a mixed master/slave and all-slaves model was designed and implemented to address these issues. First, a conscience effort was made to reduce load imbalances by building in a work distribution system (using a master/slave model) to ensure that certain cluster nodes were not sitting idle while others had a large amount of work to do. By doing this it also allowed PIPE to scale easily to any physical architecture which enabled PIPE to take full advantage of any and all available resources. Secondly, to address the memory inefficiencies, a shared memory approach was introduced to allow processes to share, instead of simply duplicate, data that they both needed (using an all-slaves model). This all-slaves model was used to run multiple threads on a single cluster node, essentially running predictions on multiple protein pairs in parallel. This overall approach exploits parallelism on two levels, the first by running multiple PIPE threads on a given cluster node and secondly by running many of these processes on different cluster nodes, all being fed work in an on-demand fashion.

PIPE3's parallel architecture performed well overall. PIPE3 was thoroughly benchmarked on three vastly different environments, the Carleton Bioinformatics Lab cluster, the High Performance Computing Virtual Laboratory's (HPCVL) Beowulf cluster as well as HPCVL's Victoria Falls cluster. The all-slaves portion of the parallel architecture allowed PIPE3 to make full use of each node on any given cluster and produced a significant speedup. The master/slave portion of the parallel architecture resulted in a superlinear speedup when more cluster nodes were used on the Bioinformatics Lab cluster as well as on the Beowulf cluster and and a roughly linear speedup on the Victoria Falls cluster. This overall linear speedup allows PIPE3 to efficiently use all available resources on a given large scale compute cluster. It would be these new efficiencies and scalability that would allow PIPE3 to make some previously impossible

proteome-wide protein-protein interaction predictions.

PIPE3 was then used to produce some significant scientific results. The first major task was to run a proteome-wide scan on all 280,454,086 possible protein pairs in the *Caenorhabditis Elegans* organism. PIPE had never attempted a proteome-wide scan on any organism of this complexity, which represents an order of magnitude more protein pairs than in the yeast proteome. This run represents the first ever proteome-wide protein-protein interaction prediction for *C. Elegans* and resulted in over 32,000 predicted novel interactions. The second major scientific result, as well as PIPE's greatest accomplishment to date, was a complete scan of the *Homo Sapiens* proteome. Even though the number of *Homo Sapiens* proteins is roughly equal to the number of *C. Elegans*, the *Homo Sapiens* proteins are generally longer and the pairs are generally much harder to compute a prediction on. In fact, some pairs took several orders of magnitude longer than the hardest *C. Elegans* pair. The proteome-wide scan on the *Homo Sapiens* proteins was another first and resulted in over 130,000 predicted novel interactions. Both the *C. Elegans* and *Homo Sapiens* predictions are supported by the fact that a significant number of the pairs either reside in the same cellular component as one-another, are involved with the same biological process, perform the same molecular function, have another interaction partner in common or some combination of these.

1.4 Organization of Thesis

First, in *Chapter 2 - Review: Computational Methods of Predicting Protein-Protein Interactions*, we present a survey of the different approaches to predicting protein-protein interactions. In *Chapter 3 - Review: Protein-protein Interaction Prediction Engine: PIPE* the first two implementations of the PIPE project will be detailed as well as the major limitations of the PIPE2 implementation in particular. The proposed and implemented solutions to these limitations will be presented in the next chapter, *Chapter 4 - PIPE Version 3: Massively Parallel PIPE.* In *Chapter 5 - Benchmarking Results* the overall benchmarking scheme and results of the benchmarking tests of the new PIPE3 implementation will be detailed. Following this in *Chapter 7 -*

Scientific Results both the results of the *C. Elegans* and the *Homo Sapiens* proteome-wide predictions will be presented. Finally an overall summary of the results as well as potential future work will be covered in *Chapter 7 - Conclusions and Future Work.*

Chapter 2

Review: Computational Methods of Predicting Protein-Protein Interactions

In this chapter, the different categories of approaches to protein-protein interaction prediction will be discussed. A small sample of approaches will be examined, discussing the underlying method to these approaches and some general results will be presented, if available. It should be noted however that many of the approaches have wildly different testing schemes and testing data sets, which makes directly comparing approaches based on high level results problematic. It is for this reason that if an approach has a highly suspect testing regiment or if their results are too specific to be used to compare to other approaches then this information will be omitted.

2.1 Class of Approaches Based on Genomic Information

The approaches classified as genomic methods predict protein-protein interactions using information taken from complete genome sequencing. The basic idea behind this class of approaches is that by looking at the organization and relative ordering of genes within a certain organism, we can get an idea of which genes (and, by extension, the various proteins encoded by the protein-coding genes) are functionally related [10].

Overbeek *et al.* took evidence regarding the conservation of gene ordering and clustering between genomes to build a system to predict functional coupling between genes. First, a notion of closeness was defined to determine whether or not any two genes were "close" to one another in a given genome. Secondly, the notion of bidirectional best hit (BBH) was defined to be that, for two given genes X and Y from two genomes G_1 and G_2, X and Y form a BBH if and only if X and Y are relatively similar and there is no other gene in G_2 more similar to X than Y and there is no other gene in G_1 more similar to Y than X. Pairs of close bidirectional best hits between two pairs

of genes from two genomes (lets say X_a and Y_a from genome G_a and X_b and Y_b from genome G_b) are found under the following restrictions:

- Genes X_a and Y_a and genes X_b and Y_b are "close" to each other in their respective genomes, and

- Genes X_a and X_b and genes Y_a and Y_b are bidirectional best hits.

Nearly 60,000 pairs of close bidirectional best hits were found among 31 genomes. From these, 35% could be confirmed that functional relationships can be inferred from pairs of close bidirectional best hits, confirming the hypothesis that conserved gene clustering retain functional relationships [25].

Another popular approach under this classification uses the idea of gene-fusion. The main hypotheses of this approach is that certain distinct proteins in a given organism exist as fused proteins in another organism. Enright *et al.* devised a scheme to search for these gene fusion events based on sequence comparison with the underlying assumption that if two proteins in a given organism are orthologous (equivalent) to a "fused" protein in another organism then the two proteins are likely to interact. Using this approach, they were able to identify a number of these types of events in three genomes. Within their results, they found a number of previously identified, well-known interacting pairs as well as a number of unconfirmed cases which could be considered interesting, testable predictions [7].

Marcotte *et al.* also proposed a very similar approach which they termed "domain fusion analysis". With their analysis they were able to predict protein pairs that have related biological functions and also predict potential protein-protein interactions [18]. Marcotte *et al.* went on to combine their approach with two other independent genomic approaches to create a large-scale prediction of protein functions. The main idea was to take the union of the predictions of these approaches and then test the links created between given proteins. If protein X is linked to a certain group of functionally related proteins, then the function of X can be inferred based on the

functionality of the group of related proteins. This idea can be tested when the function of X is previously known. Applying this aggregated approach to yeast, it was found that of the roughly 2,500 proteins whose functions were previously uncategorized, 15% of them could be assigned a general function with a very high confidence and a total of 62% can be assigned a general function overall. It was stated that this approach should substantially increase the rate at which protein function is discovered [19].

2.2 Class of Approaches Based on Evolutionary Relationships

Approaches that fall within this classification rely on the general hypothesis that the physical or functional relationships between proteins can be inferred by the evolutionary relationships that the proteins have.

One such approach proposed by Pellegrini *et al.* uses phylogenetic profiling to characterize proteins as either likely or unlikely to interact. A phylogenetic profile of a protein simply represents which organisms, out of a given set of organisms, contain a homolog (or a very similar protein) to that protein. The profile for a given protein is simply a bit-string of length n (where n is the number of considered organisms) and each bit corresponds to an organism: if a given bit is 1 then the protein has a homolog in that organism, if it is 0 then it does not. The authors then went on to show that if two proteins have homologs in the same subset of organisms (had identical or very similar phylogenetic profiles) then they were likely to interact. They also showed supporting evidence to this hypothesis in that proteins that are thought not to interact do not have similar phylogenetic profiles. The authors however noted that not all interacting proteins have similar profiles, and so this method is not perfect [28]. The general idea behind this approach can be seen in Figure 2.1.

Building off of the work done by Pellegrini *et al.*, a more sophisticated way to determine how to measure the similarity between two phylogenetic profiles was created. To

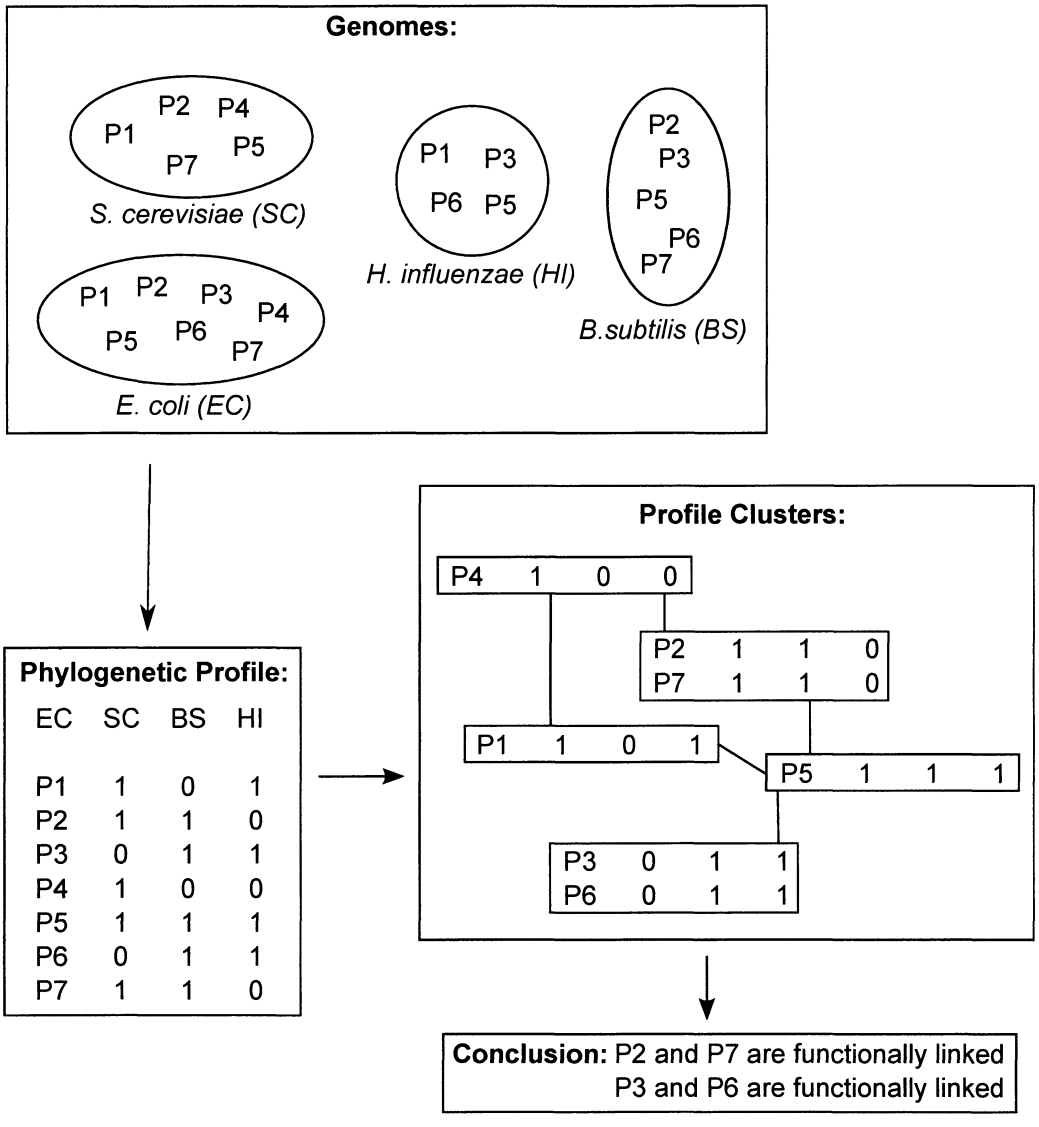

Figure 2.1: An overview of the algorithm developed by Pellegrini *et al.* to determine functional linkage between proteins through the use of phylogenetic profiles. Reproduced from [28].

do this, the profiles were mapped into a high-dimensional vector space and then an algorithm was developed to calculate the inner product of two profiles represented in this space. This inner product was termed the *tree kernel*, which belongs to a larger set of functions called kernels. Kernels are functions that calculate the inner product of two objects mapped into a vector space. With this *tree kernel* it is possible to define a distance between profiles in this vector space and, in this case, two profiles

are near each other if they are likely to have shared common patterns of evolutions. Support Vector Machines (a kernel-based supervised learning algorithm) were then used on phylogenetic sequences using the *tree kernel* as their kernel. It was shown that SVMs using the *tree kernel* performed much better on average than SVMs using a naive kernel at predicting protein-protein interactions [40].

The use of the binary phylogenetic profiles was extended in another work to compare the phylogenetic trees of proteins and use this as a method to predict how likely the given proteins were to interact. In comparison to a phylogenetic profile, a phylogenetic tree shows the evolutionary relationship between a group of organisms that have a common ancestor. Each node in the tree represents an ancestor to all of that node's children, and the further away two nodes are from one another in the tree, the less similar the organisms are. Pazos *et al.* thought that the similarity between the phylogenetic trees of interacting proteins implied that they had a similar evolutionary history. They devised a method in which the phylogenetic trees of proteins could be compared and then a prediction on whether the proteins interact or not could be made. This approach proved to be promising when compared to other evolutionary and genomic approaches [27].

2.3 Class of Approaches Based on 3D Protein Structure

Approaches to protein-protein interaction prediction that fall within this category rely heavily on the 3D structure of proteins that are known to interact. The underlying hypothesis that these approaches have in common is that structural information of interacting proteins can be extracted and applied to homologs to then predict further interactions.

A method outlined by Aloy *et al.* works along these lines. Their method predicts how well two potentially interacting proteins fit together based on a database of known 3D structures and then infers how likely the proteins are to interact based on the molecular details of the interaction sites, compared to those in the database. Their

algorithm works in the following way. First, the protein sequences are searched for specific, predetermined domains (shorter protein segments that have a known function). They then check their database of known interacting domains, for which the 3D structures are known, for any instances of the domains found within the query proteins. If each protein contains a domain that matches a pair of interacting domains in this database, then the proteins are aligned on these segments. Using this alignment, a protein interaction prediction can be made. To do this, they check whether the query proteins contain the same residues that make the atomic contacts of the interacting proteins in the database. From this they can estimate a statistical significnce for the potential interaction [1]. An overview of this method can be seen in Figure 2.2.

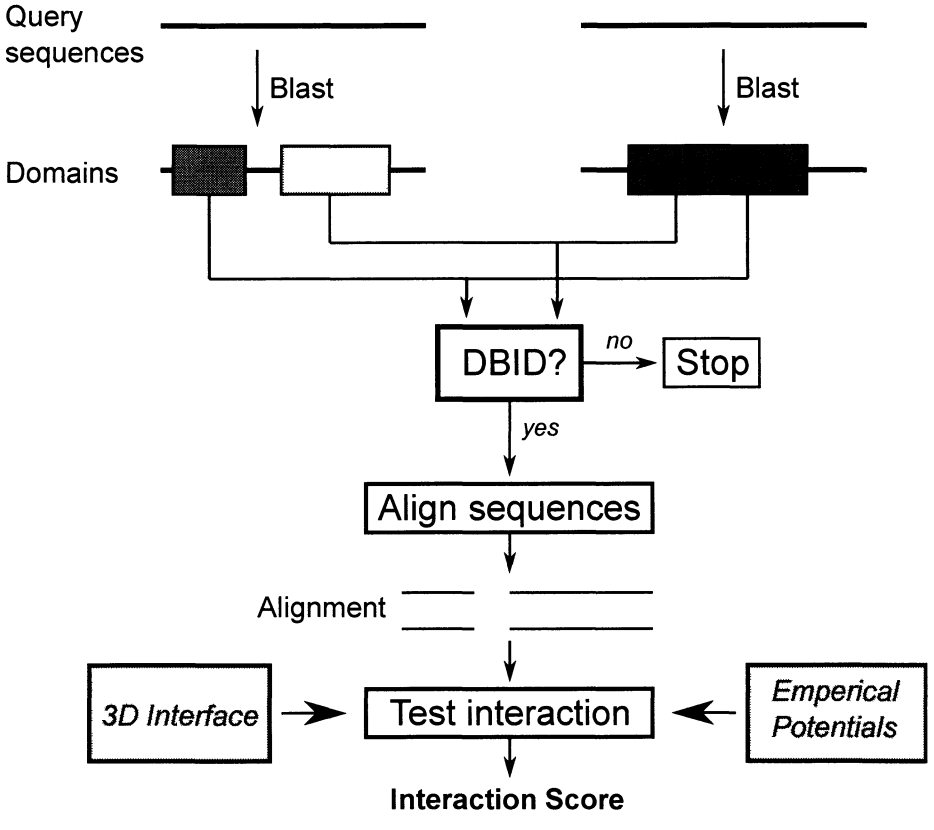

Figure 2.2: An overview of the algorithm developed by Aloy *et al.*, which predicts protein interaction based on known 3D structures. Reproduced from [1].

In this same vein, Stein *et al.* created 3did, a database containing a collection of domain-domain interactions for which the 3D structures are known in high detail. These 3D structures are exploited to provide a detailed analysis for interactions between proteins. This database was built on protein domains rather than full protein sequences and new 3D structures are incorporated regularly. The main way to retrieve information from this database is to present it with a particular query domain. 3did will then search for this domain and present all other domains that it interacts with and this will then be displayed as an interaction network (see Figure 2.3 for an example). From here, the user can select a particular interaction to receive more details. Finally, the user can also get a 3D visualization of the given interaction if they so desire. The database can also be queried with a protein sequence itself. From this sequence, 3did will identify specific domains and then display interaction links from all of these doamins to domains stored in the database [38].

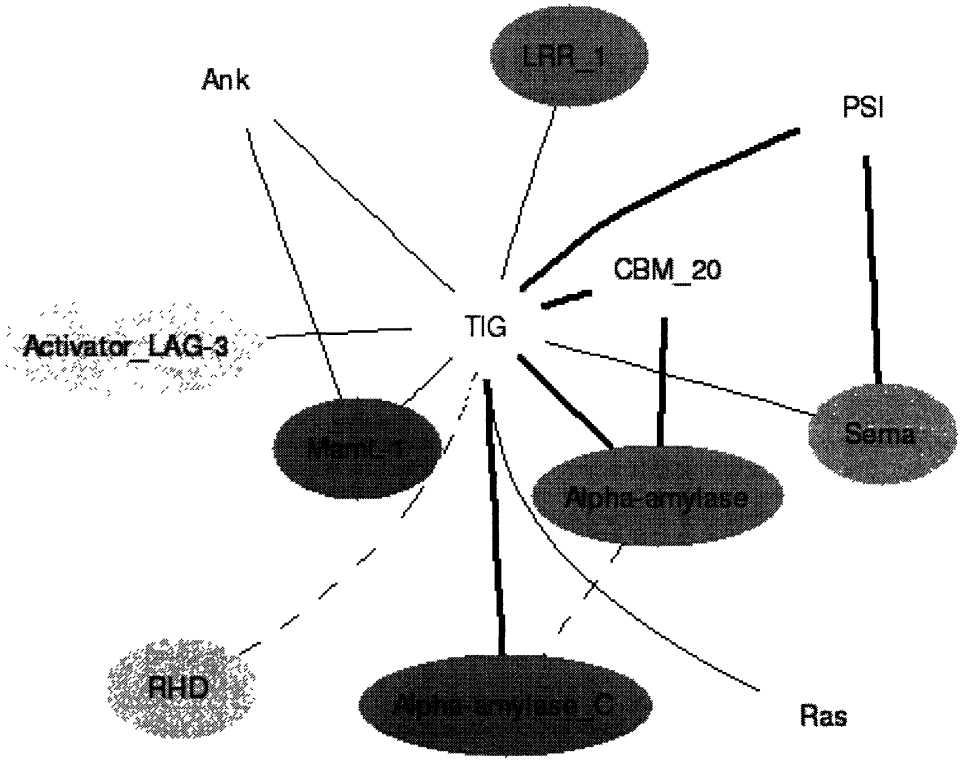

Figure 2.3: An example of an interaction network retrieved from the 3did database.

Espadaler *et al.* developed a two-pronged approach to protein-protein interaction prediction. First, they took a set of 3D structures for interacting proteins and extracted the interface sequences of the proteins along the sections where they interact. With these interface sequences, they searched through a set of proteins, looking for proteins which contained these sequences. If two proteins could be found that each contained a segment that is very similar to an interface on two interacting proteins, then these proteins were considered a possible interacting pair. The second part of their approach involved searching for homologs of the proteins known to interact. In this part, if two proteins are homologs to two proteins that are known to interact, then they are considered a potential interacting pair. After this was done, the intersection of the two sets of predicted interacting pairs were taken. This then meant that each pair in this set of pairs of proteins had the following properties:

- the pair of proteins each contain a segment that is very similar to the interaction interface of a pair of proteins that are known to interact
- the pair of proteins have a similar structure to a pair of proteins that are known to interact

This set of potentially interacting protein pairs was quite large. A portion of this set could be confirmed or at least supported with experiment evidence, leaving a large portion of this set as newly reported, potentially interacting pairs [8].

Another approach in this class is PRISM (protein interactions by structural matching), developed by Ogmen *et al.*. Similarly, they relied on the hypothesis that if two proteins contain particular regions that are similar to regions of proteins that are known to interact, then it is thought that the initial proteins possibly interact through these regions. Their main algorithm takes two sets as input: a "template" dataset (a dataset for which the interactions and 3D structurs are known) and a "target" dataset of which one would like to map the potential interactions. Then, the target proteins are compared to the template proteins to in order to find template proteins that are "structurally similar" to the target proteins. In doing this, a similarity score is generated. If two proteins are "similar" enough to a pair of interacting proteins in

the template set, then they are predicted to interact (the similarity score contributes to the overall prediction score). This process is outlined in Figure 2.4. The web interface of PRISM also allows the user to put in their own query proteins that were not included in their target set to predict possible interaction partners [24].

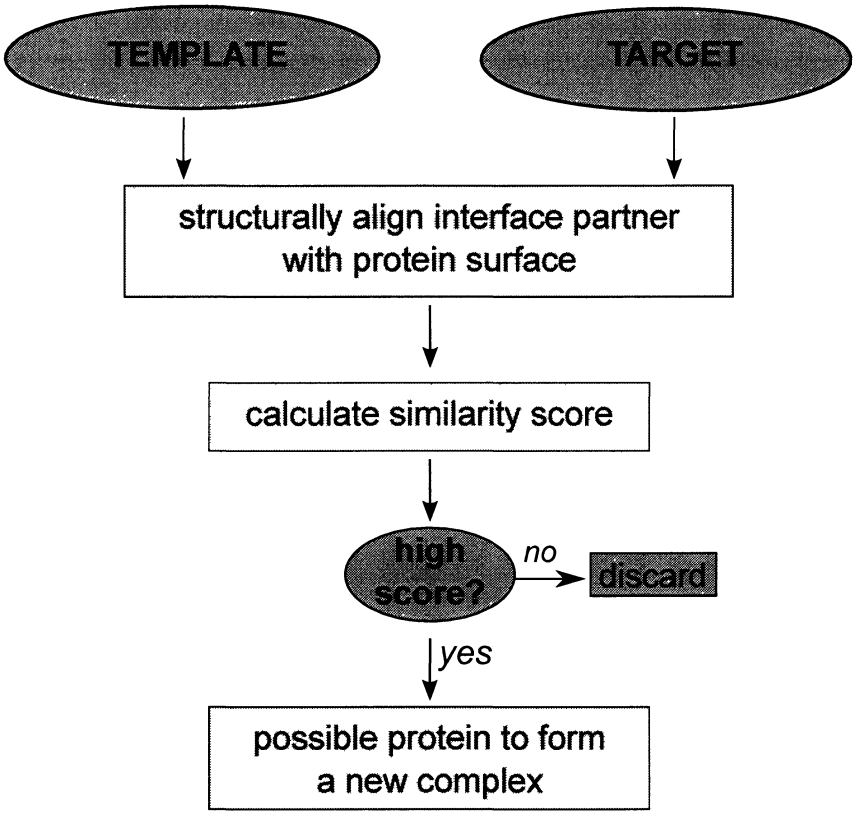

Figure 2.4: An overview of the PRISM approach to predict protein-protein interactions based on 3D structures. Reproduced from [24].

2.4 Class of Approaches Based on Protein Domains

Approaches that fall into this class are approaches that try to identify previously defined protein domains (or simply protein segments which have been previously determined to have some noteworthy properties) that participate in protein interaction. The identification of these domains could then be used to predict interactions in other proteins.

Sprinzak *et al.* developed an approach that looks to identify pairs of sequence-signatures from a database of experimentally determined interacting proteins and then use the pairs of sequence-signatures to predict further protein-protein interactions. To do this, a large database of yeast protein pairs known to interact was used. Each protein in this database was then assigned signatures (domains) that are defined in another yeast database known as InterPro. The signatures in this database are in the form of regular expressions, profiles, fingerprints and hidden Markov models, all compiled from various other databases. They then counted the combination of these signatures in the database of known interacting protein pairs to identify sequence-signature pairs that appeared in many pairs of interacting proteins (where one protein contained one sequence and the other protein contained the other sequence). The authors then concentrated on the sequence-signatures that occurred more often than expected at random to be sure that these signatures were relevant. Using leave-one-out cross validation on subsets of known interacting protein pairs, they were able to reach sensitivities as high as 97%, however the authors note that these test groups were quite small. The authors also mention the fact that they still must rely upon the InterPro database to define the sequence-signatures, and therefore are limited by that [37].

Another approach similar to this was developed by Kim *et al.* In their approach, they took advantage of two separate databases. The first was a database containing experimentally determined interacting protein pairs called the database of interacting proteins (DIP) and the second database was the InterPro database discussed in the previous example. Their stated assumption was that if the domain pair (a, b) is found in the interacting protein pair (A, B) then it is considered a potentially interacting domain pair, and the more often this domain pair was found within interacting proteins then the more likely it is a truly interacting domain pair. Their method was defined as follows. First, 10,435 interacting protein pairs from DIP were extracted. Within these interacting pairs, the 5,849 unique protein sequences were scanned for defined domains from InterPro. They then went on to build a potentially interacting

domain (PID) matrix, where every combination of potentially interacting domains was considered and a score representing their prevalence was stored. You could then use this matrix to do protein interaction predictions for two query proteins by simply searching the proteins for domains, and then seeing how likely these domains are to interact by checking their PID score. Using cross validation it was found that the average sensitivity was 50.1% of the domain containing pairs of proteins. This dropped to 27% however when all proteins in DIP, not just proteins containing domains, were considered. On the other hand, their test results showed a specificity of 98.4%, which is quite promising [16].

Another domain based approach was presented by Deng *et al.* Their approach is similar to the previous two in that they relied on a set of known interacting protein pairs and a set of predefined protein domains, however they used the domains defined in the PFAM (Protein Family) database. Their algorithmic approach was also different. They presented two separate approaches, the better of the two relied heavily on probability theory. The approach was to apply maximum likelihood estimation to infer interacting domains from the known interacting pairs and then apply this to a set of proteins in order to predict some novel protein-protein interactions. To test their theory, they applied it to a set of proteins in which the interactions are already known and they found that their prediction rate is about 100 times better than that of random assignment [6].

A more complex system titled PreSPI was introduced by Han *et al.* which tried to overcome the earlier approaches' tendencies to assume that protein-protein interactions were mediated through a single pair of domains. With this approach they wanted to consider more possibilities by considering the mediating factor of a protein-protein interaction to be groups of domains instead of just a single pair. In practice, this means that every subset of domains within a pair of interacting proteins are examined and are considered potentially responsible for the interaction. So if protein A contains n domains and protein B contains m domains then instead of simply considering the $m \times n$ potentially interacting domain pairs, the $(2^n - 1) \times (2^m - 1)$

different potentially interacting domain combination pairs from both proteins need to be considered. This notion is illustrated in Figure 2.5.

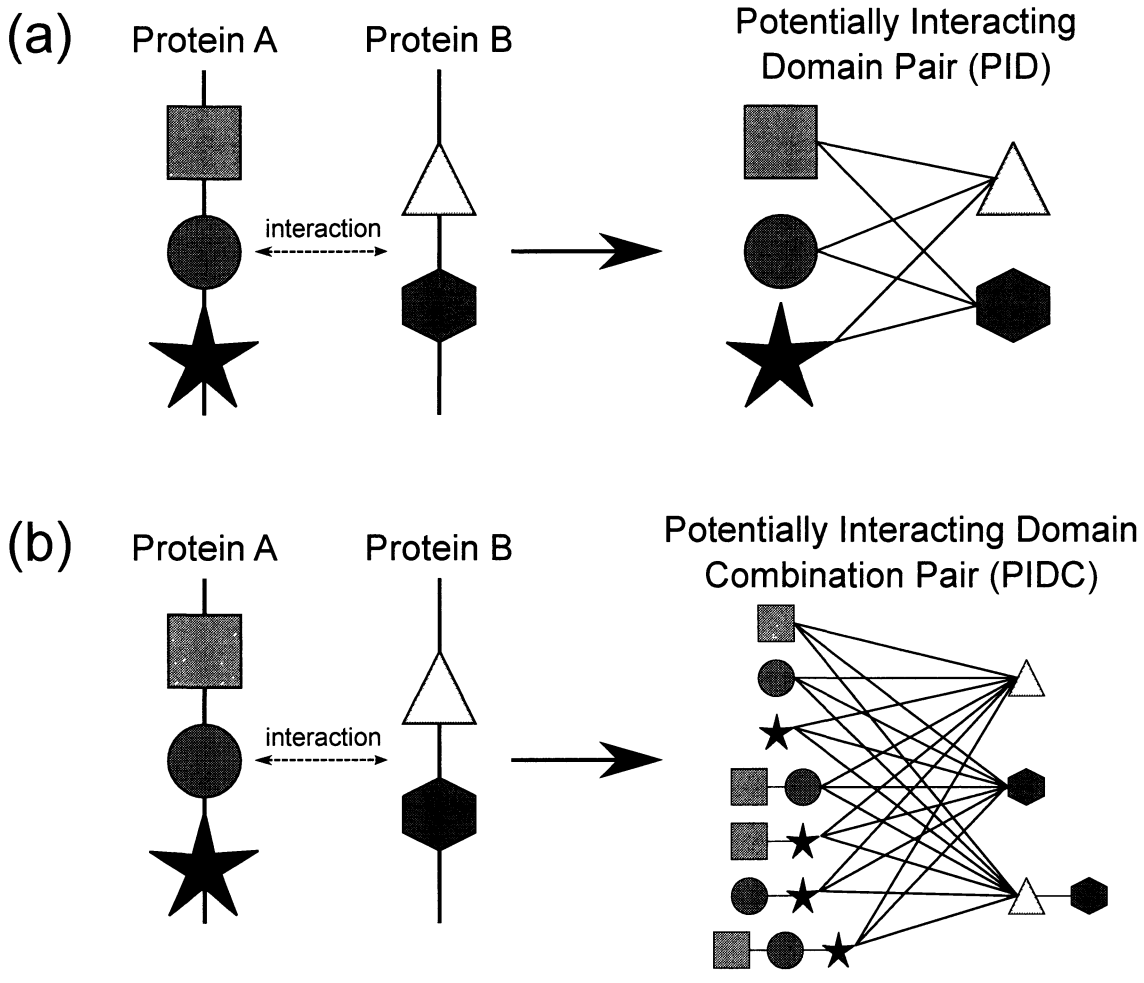

Figure 2.5: An illustration of the difference between a) the conventional domain pair based prediction model and b) the domain combination pair model proposed by Han *et al.* Reproduced from [12].

The authors then went on to build a matrix to be used for interaction prediction, similar to the the matrix used by Kim *et al.* However, this matrix was an $n \times n$ matrix, where n is the total number of domain combination groups identified in the training set of interacting protein pairs. This then allowed for a cell for each domain combination pair which contained a value representative of the pair's prevalence in interacting proteins. Using matrices built like this, a prediction on two query proteins

could be made. Using this method they could also rank the quality of their predictions as well by assigning their prediction as a probability. In testing their approach, they were able to reach a sensitivity of 77% and a specificity of 95%. It was also noted that, in general, their ranking method seemed to work as well since a large portion of the true positives that were predicted to interact had a high interacting probability and a large portion of the true negatives that were predicted not to interact had a low interaction probability. The authors note however that this ranking method should be applied with caution, especially when an interaction probability between 0.4 and 0.6 is received because this is where some problems could be found [12].

2.5 Class of Approaches Based on Primary Protein Structure

Finally, the last class of approaches to protein-protein interaction prediction is based on primary protein structure. In these approaches, the general hypothesis is that the interactions between proteins are mediated through short polypeptide sequences, or simply a sequence of amino acids that define the primary structure of the protein. It is thought that these sequences can be extracted from sets of interacting protein pairs and then applied to proteins in order to predict possible interactions.

An example of one such approach was introduced by Bock *et al.* In their approach, they would take two similar organisms, one of which a set of interacting protein pairs is known and the other of which nothing is known. The main idea behind their work is that since the organisms are similar, then they will contain very similar proteins (homologs) and the proteins in the second organism will most likely function in a very similar manner. Based on this, protein-protein interactions can be predicted in the second organism based on the proteins that interact in the first organism. To do this, SVM learning algorithms were used to "learn" the protein-protein interactions in the first organism. After testing and tuning the parameters based on this organism, the proteins from the second organism could be fed into the SVM. The SVM would then construct an interaction network on these proteins, based on what it had learned from the interaction in the first organism. The authors noted that, from here, it would be possible to experimentally validate the results and then if this interaction network is

deemed to be relatively correct then it could be used to predict interactions between proteins in organisms similar to it. Not only was the precision and sensitivity pretty impressive for their tests (80% and 69% respectively), the more impressive results were more general. It was shown that the interaction networks produced using this method were strikingly similar to experimentally derived networks in terms of average connectivity and cluster size distribution within the interaction networks. These facts add more biological credibility to their results [3].

Another approach developed by Martin *et al.* builds upon the previously mentioned approach and the approach developed by Sprinzak *et al.* described in the previous section. They basically combined the use of a support vector machine from Bock *et al.* with the sequence-signatures idea from Sprinzak *et al.*, while not having to rely on predetermined domains specifically. To facilitate this, Martin *et al.* came up with a way to represent short sequences of amino acids (the sequence-signatures) as vectors, which they termed a signature. They then went on to define a *product signature*, which is a signature for a pair of amino acid sequences. From this, a kernel function was developed to determine the inner product between two product signatures. Now that a kernel function was defined, the authors could take advantage of the SVM. Since this method is much more general than the previous approaches, they were able to reproduce the experiments done with the previous approaches with either comparable or better results. The authors also noted that this system could possibly be used to predict protein domains. However, the main issue with this approach (as well as the approach presented by Bock *et al.*) is that since the interaction networks of proteins are so sparse, they would be likely to produce a large number of false positives since they rely on classifiers (SVMs), of which this is a known issue [20].

Roy *et al.* have recently developed another approach that use basic amino acid composition (AAC) features to predict protein-protein interactions. These features are strictly based on normalized counts of single or pairs of amino acids within the query proteins and proteins for which interactions have been previously determined. Through the use of various different classifiers, they found that their interactions

predicted in yeast were of comparable accuracy to those predicted using previously defined protein domains. An interesting aside that supports their hypothesis was that AAC features that were deemed to be important to protein-protein interaction were statistically over-represented in domains known to be involved with protein-protein interactions. This suggests that AAC alone can capture a significant amount of information needed to predict interactions [33].

Chapter 3

Review: Protein-protein Interaction Prediction Engine

The Protein-Protein Interaction Prediction Engine, or PIPE for short, is a computational tool that can be used to identify and predict protein-protein interactions. The first two versions of PIPE worked exclusively with *S. Cerevisiae*, more commonly known as baker's yeast. The main hypothesis that the engine is built upon is that protein-protein interactions are mediated by short sequences of amino acids which are common across many protein interaction sites within a given organism, placing it in the class of approaches based on primary protein structure. This means that, given two proteins, a prediction can be made on whether they interact or not based solely on their primary structure. To do this, a search through a set of protein pairs known to interact for short sequences that exist in the query proteins is done. If there exists short sequences of amino acids within each of the query proteins that also co-exist in a number of known interacting proteins then the query pair is predicted to interact. The fact that the PIPE approach relies solely on the primary structures of the proteins and a list of known interacting proteins to do its predictions is a considerable advantage compared to other approaches previously mentioned.

3.1 PIPE Version 1

The first version of PIPE was published in July of 2006 [31]. In this section various parts of that publication will be examined and discussed, mainly focusing on the PIPE algorithm itself.

3.1.1 Main Algorithm

The inputs to the PIPE algorithm are a list of pairs of proteins that are known to interact and a pair of query proteins (henceforth known as protein A and protein B). At the time of publication, the input protein pair list was composed of 15,118 pairs of

protein-protein interactions from a total of 6,304 unique proteins compiled from yeast interactions reported in multiple databases. The goal of the algorithm is to predict whether or not A and B interact based on their primary structure and the primary structures of the protein pairs that are known to interact. The algorithm works in four steps:

- **Step 1:** As previously mentioned, the PIPE algorithm relies on previously determined protein-protein interactions. The first step of the algorithm takes the set of known protein-protein interactions and creates a graph G as follows:
  - Every unique protein in the set of known interacting protein pairs is represented by a node in G
  - Every interaction between two proteins X and Y is represented as an edge between X and Y in G
- **Step 2:** In the second step of the algorithm, the first query protein (protein A) is investigated to find similar proteins in our graph of known interactions. To do this, A is split up into overlapping fragments of size w. This can also be thought of as using a sliding window of size w on A, where sliding the window by one amino acid creates a new fragment. For each fragment a_i of A, where $0 \leq i \leq (|A| - w + 1)$, we:
  1. Search for fragment a_i in every protein in the graph G. To do this, a sliding window of size w is used on each protein in G, and each one of these protein fragments is compared to a_i.
  2. For each protein that contains a sequence that matches a_i, all of that protein's neighbours in G are added to a list R

  To determine whether two protein fragments match, a score is generated with the use of a substitution matrix. A specific type of substitution matrix exists for comparing the similarity of sequences of amino acids. In this matrix, there is a row and column for each amino acid and the value stored at cell (i, j) is the probability that the amino acid j is replaced (through some evolutionary process) by amino acid i after a given evolutionary period. The PAM1 matrix

is normalized so that it stores the probabilities of amino acids changing into other amino acids where only a single mutation occurs per 100 amino acids. The PAM1 matrix is useful for comparing sequences of amino acids that are very similar [5]. However, with this particular implementation, a broader range of sequences should be considered similar. To accomplish this, the PAM120 matrix was used. This matrix is simply the PAM1 matrix multiplied by itself 120 times. To judge whether two sequences of amino acids are similar, the sum of the PAM120 matrix values of the corresponding amino acids are added up. If this sum is greater than some threshold (denoted as S_{PAM}) then the sequences are deemed to be similar.

- **Step 3:** The third step of the algorithm involves searching for sequences found in the second query protein, protein B, in the list R produced in the previous step. To do this, a sliding window of size w is used again to create fragments b_j, where $0 \leq j \leq (|B| - w + 1)$, of B. A search for every fragment b_j is then performed on all of the protein in the list R in the same manner as the previous step. We now create an $n \times m$ matrix, where $n = |A|$ and $m = |B|$, that is initialized to contain only zeroes. For every time a protein fragment b_j of B matches a fragment of a protein Y in R (who was inserted into R for being the neighbour of some protein X that contained a fragment that matched some fragment a_i from A) the cell at position (i, j) will be incremented. Row i of of the matrix represents fragment a_i and column j represents fragment b_j. The resulting matrix will show how many times each pair of fragments of size w from A and B co-occur in other proteins that are known to interact.

- **Step 4:** The last step of the algorithm is to present the matrix as a 3D surface where each row represents a fragment a_i from query protein A and each column represents a fragment b_j from query protein B and the elevation represents the number of matches found for the particular fragments a_i and b_j. If there is a cell in this matrix that has a value greater than some threshold M then A and B are predicted to interact.

An overview of this algorithm is given in Figure 3.1.

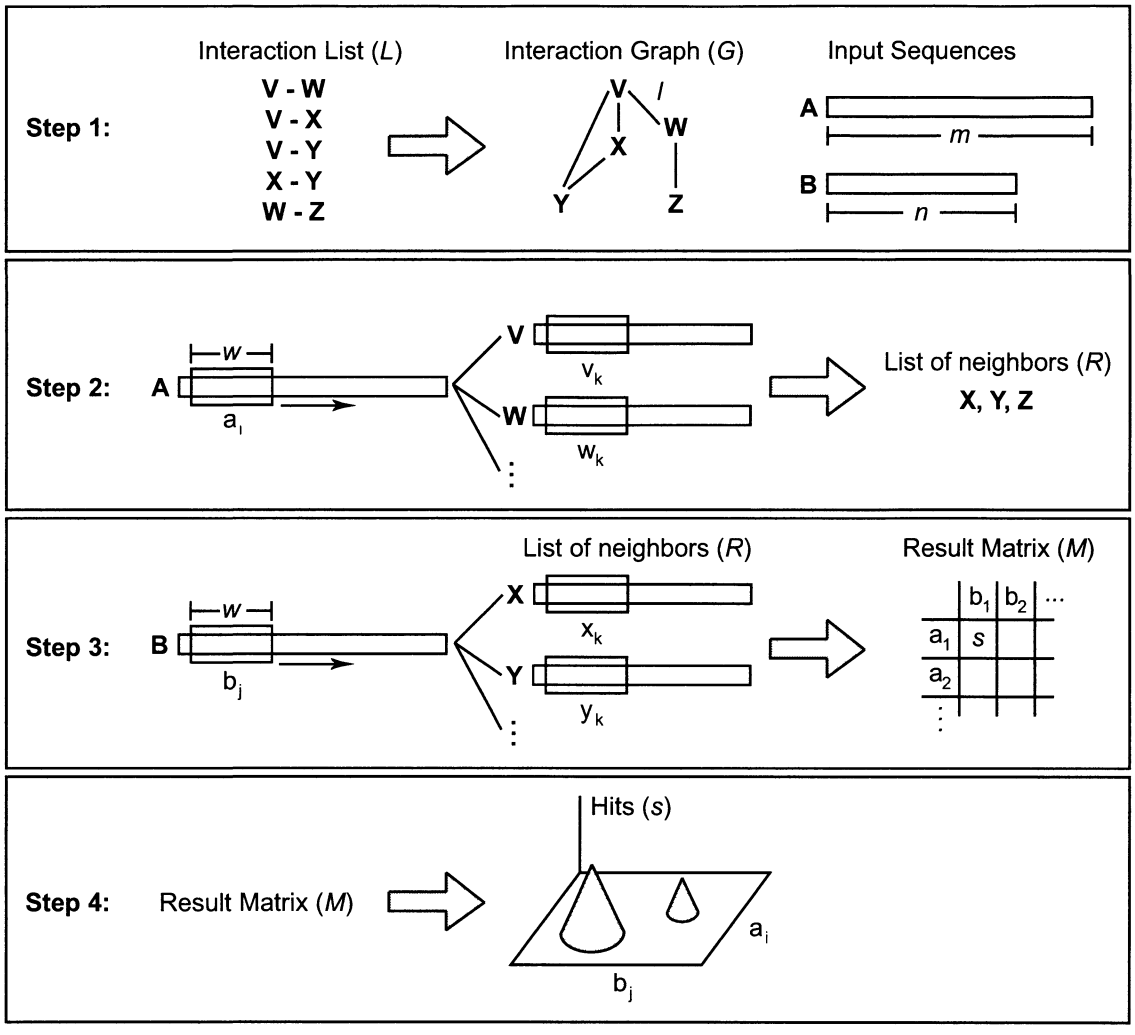

Figure 3.1: Visualization of the PIPE algorithm. Taken from [31].

3.1.2 PIPE Parameters

There are three main parameters involved with the PIPE algorithm. They are:

1. The window size, referred to as w.

2. The threshold S_{PAM} that determines a match between two protein fragments.

3. The threshold M that determines, based on the output matrix, whether or not PIPE predicts that proteins A and B interact.

In theory, a desirable w would be as small as possible so to identify interaction sites with as much precision as possible. However, when w gets too small then too many random fragments would match. A window size of $w = 20$ was found to be reasonable, and from that the values $S_{PAM} = 35$ and $M = 10$ were used. To show that the threshold $S_{PAM} = 35$ with a window size of 20 was reasonable, 1,000,000 randomly built fragment pairs were created, using the same amino acid distribution that existed in the known interacting protein pairs. Figure 3.2 shows the probability of two random fragments matching with a given S_{PAM} value. The probability of two random fragments matching with $S_{PAM} = 35$ is less than 10^{-6}.

Similarly, 1,000 random protein pairs were generated whose amino acid distribution was the same amino acid distribution that existed in the known interacting protein pairs. PIPE was run with each pair and the maximum score in the resulting matrix was determined. Figure 3.3 shows the probability of a pair of randomly generated proteins to have a given PIPE score. The probability of two random proteins receiving a score greater than 10 is less than 10^{-6}.

3.1.3 Running PIPE and Interpreting the Output

Typical examples of PIPE output are shown in Figure 3.4. The x and y axis represent the amino acid fragments from the query proteins where a given position represents the protein fragment of size 20 starting from that position in the overall protein. The values on the z axis represent the number of times a given pair of protein fragments co-occur in known pairs of interacting proteins. Figure 3.4a shows an example output for a pair of proteins that PIPE would predict not to interact. On the other hand, Figure 3.4b shows an example output where PIPE would predict that the two query proteins would interact.

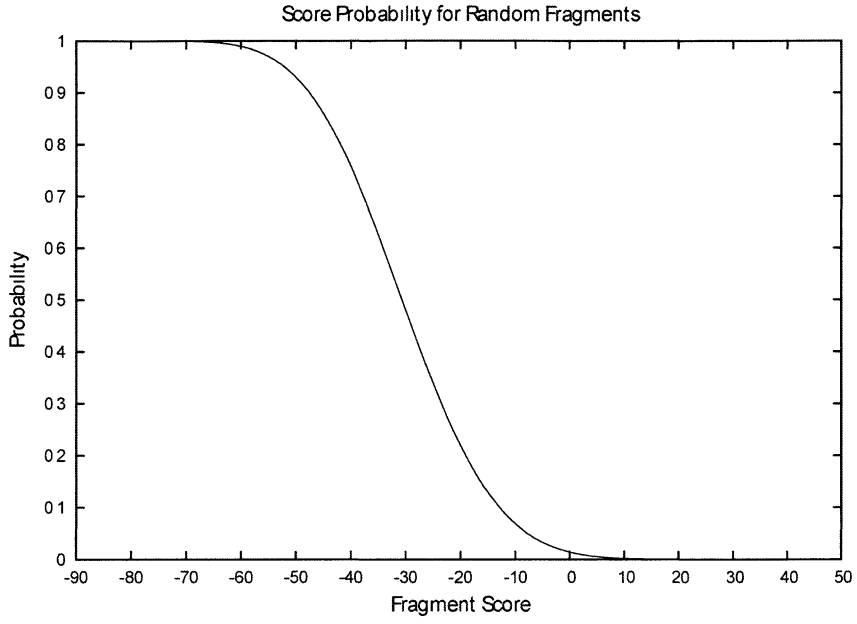

Figure 3.2: PIPE Parameter Tuning: The probability of two random protein fragments "matching" for a given S_{PAM} value. Taken from [31].

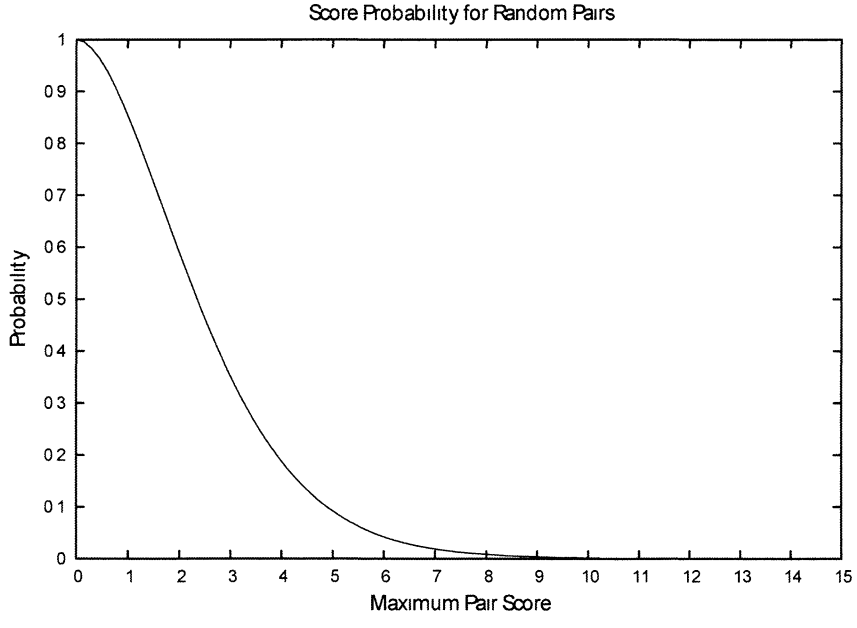

Figure 3.3: PIPE Parameter Tuning: The probability of two random proteins being predicted to interact for a given M value. Taken from [31].

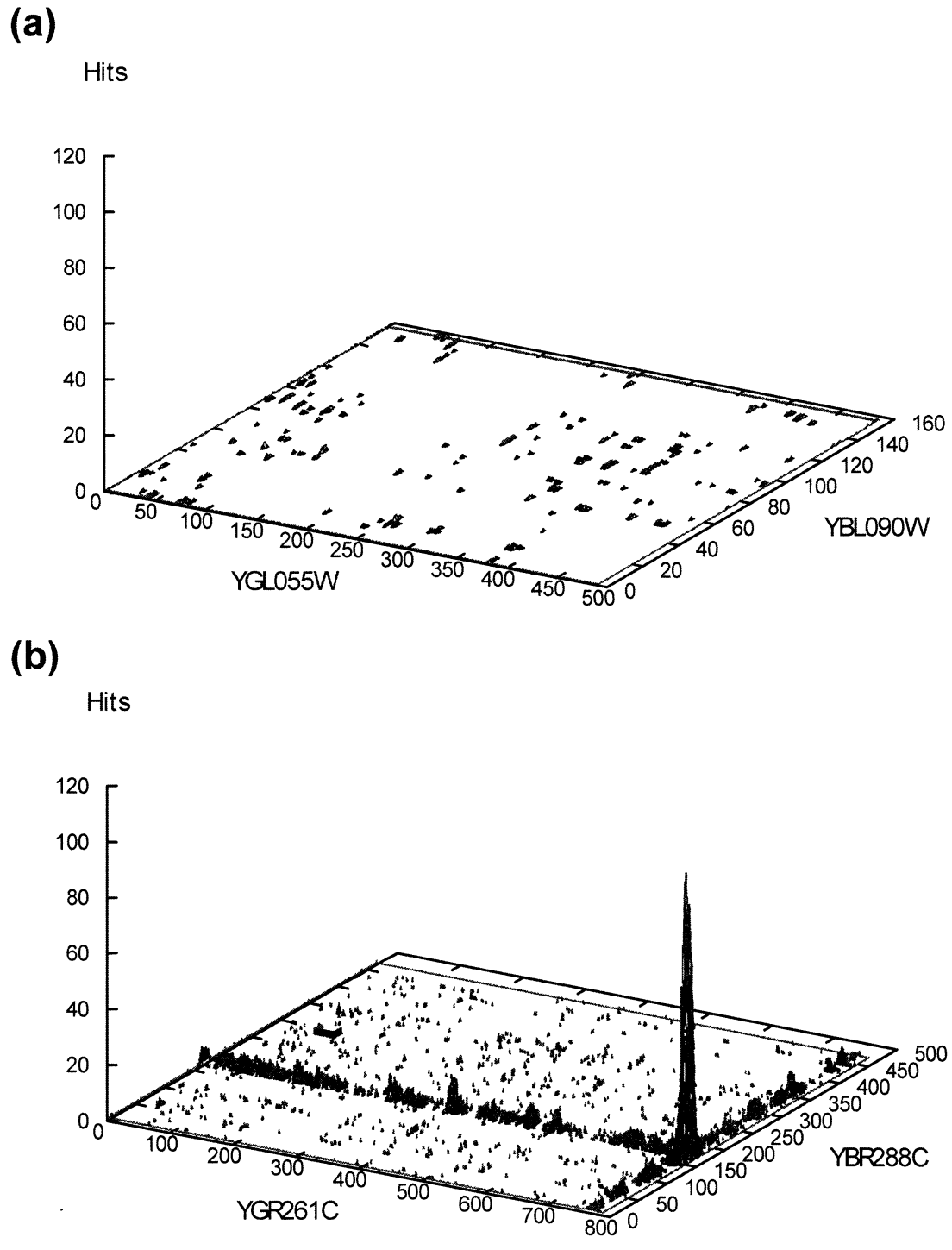

Figure 3.4: Example PIPE output landscapes. Taken from [31].

3.1.4 PIPE Version 1 Overall Results

In this section the overall results and abilities of the first version of PIPE will be discussed.

- **Ability of PIPE to Detect Interacting Proteins**

  To determine the accuracy of PIPE two sets of protein pairs were used. The first set was 100 pairs of proteins that are known to interact based on at least three different lines of experimental evidence. These protein pairs also don't exist within the data set used by PIPE to build its known interaction graph. Of these 100 pairs of proteins, PIPE successfully predicted 61 of them. This suggests that this version of PIPE has a sensitivity of 61% and a false negative rate of 39%. These results are comparable to experimental results performed in the lab. To evaluate the specificity of PIPE, a set of 100 pairs of proteins that are expected not to interact based on protein localization data, co-expression profiling, known direct or indirect functional or genetic relationships and the information gathered from the complete set of protein interaction data sets [31]. PIPE successfully classified 89 of these 100 protein pairs suggesting a specificity of 89% and a false positive rate of 11%. These figures together give PIPE an overall accuracy of 75%.

- **Ability of PIPE to Detect Interaction Sites Between Interacting Protein Pairs**

  To determine if PIPE could detect the actual interaction sites of a given pair of proteins who are known to interact, 10 pairs for which the interaction sites were known were gathered. PIPE classified seven of these ten pairs as likely interactors. Of the 7 pairs that PIPE correctly classified, PIPE successfully detected interaction sites, based on the output landscape, of four of these pairs. This suggests a 40% success rate for PIPE to identify previously reported interaction sites. However, the data sets used for these tests were very small since there was not much data of this kind available at the time.

- **Ability of PIPE to Detect Novel Protein-Protein Interactions**

  To test whether or not PIPE was able to predict novel protein-protein interactions, two proteins (namely YGL227W and YMR135C) were investigated. This protein pair was not reported in the literature and no experimental interaction data was available. With a peak score of nearly 140, PIPE predicted that these two proteins were a novel interacting pair. The PIPE output for these two proteins can be seen in Figure 3.5. To confirm this prediction, laboratory experiments were performed and it was demonstrated that these two proteins do in fact interact. This shows PIPE's ability to detect novel protein-protein interaction, which presents a significant advantage over the experimental techniques used. Not only is it much faster, easier and cheaper to use but roughly only 31% of yeast proteins can be successfully investigated using the experiment techniques used here. Based on this information, PIPE may be able to contribute in areas where laboratory approaches cannot.

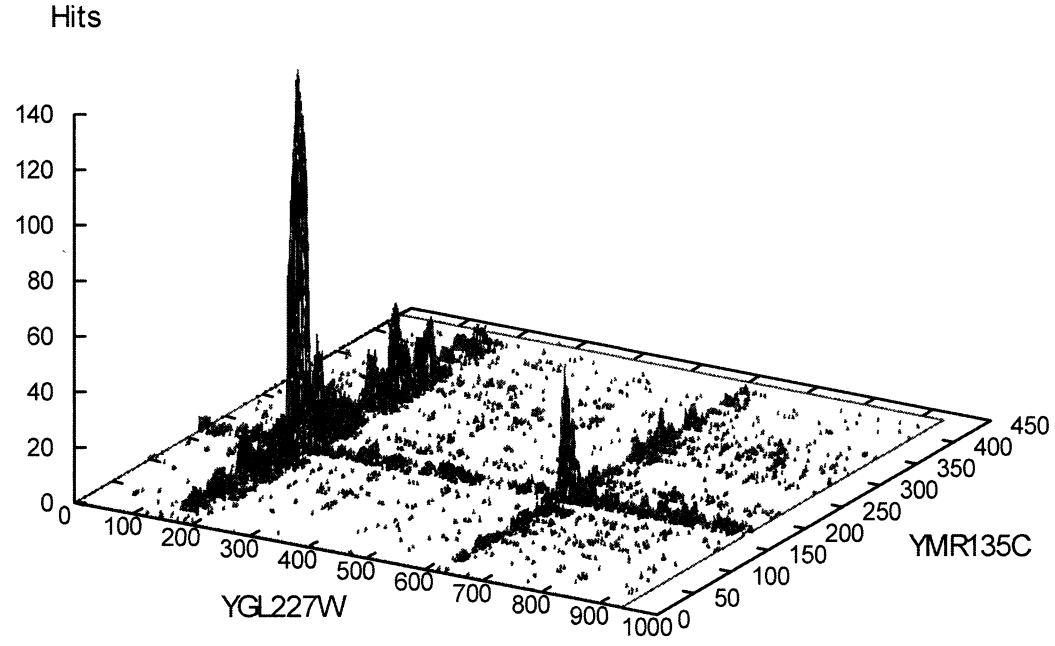

Figure 3.5: PIPE output for proteins yeast YGL227W and YMR135C. This pair of proteins was predicted to interact and this prediction was confirmed through laboratory experiments. Taken from [31].

- **Ability of PIPE to Elucidate the Internal Architecture of Protein Complexes**

  Protein complexes, which are simply a group of proteins bound together through interactions, are another structure that are of interest. While performing experimental tests on the YGL227W protein previously mentioned, a novel protein complex termed vid30c was identified. However, the internal structure of this protein complex could not be examined through experimental methods. To build a model of the protein complex, PIPE was used to analyze protein pairs involved in vid30c. The data gathered was then used to build the hypothetical model shown in Figure 3.6. Through the use of other laboratory experiments, the overall structure of vid30c predicted by PIPE is consistent with pair-wise analysis of the proteins. To further test PIPE's abilities in predicting protein complexes' internal structures, 10 known protein complexes were tested on PIPE. Of the 30 potential protein pair interactions that came out of these 10 complexes, 20 were shown to to interact and 10 were shown not to. PIPE successfully predicted 13 of the 20 true interactions and 4 of the 10 non-interacting protein pairs. Overall, PIPE successfully predicted 3 internal structures perfectly out of the 10 protein complexes.

3.1.5 Discussion

Given the results discussed above, the first version of PIPE was shown to be an effective tool to predict protein-protein interactions. With a sensitivity of 61% and a specificity of 89%, this version of PIPE was shown to have an overall accuracy of 75%. Although these numbers are quite promising compared to other approaches, this version of PIPE would be weak for detecting protein-protein interactions for proteome-wide data sets. An example of running PIPE on all pairs of yeast proteins was given. There are approximately 20,000,000 pairs of yeast proteins, of which roughly 50,000 are assumed to interact. Knowing this, if PIPE were to be run on all of these pairs, it would report roughly 30,000 of the 50,000 interacting pairs and identify 17,750,000 true negatives. However, PIPE would also report 2,200,000 false positives and a less impactful 20,000 false negatives. Due to the huge number of

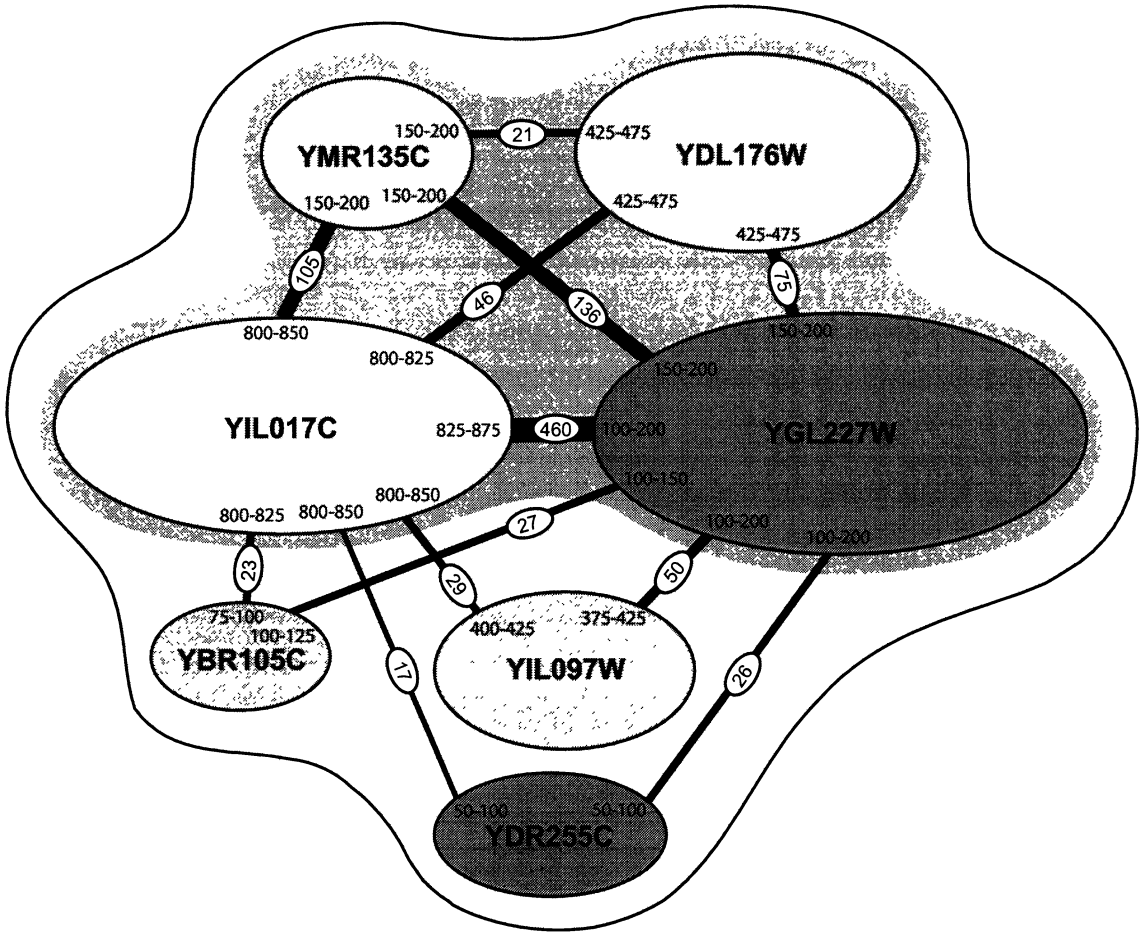

Figure 3.6: Vid30c internal architecture as predicted by PIPE. Taken from [31].

false positives that would be reported, this version of PIPE would be ineffective for analyzing such large data sets. Another drawback of this version of PIPE is the running times. Looking back at the outline of the algorithm, it is obvious that the amount of computation involved in running PIPE for a given protein pair is substantial. At this point in PIPE's history it would take several hours on a standard desktop machine to run PIPE for a given protein pair. It was mentioned that more sophisticated data structures were being implemented as well as a move towards parallel processing was underway, however they were not complete for this version. These running times would make it impossible to do a proteome-wide all-to-all protein interaction search.

3.2 PIPE Version 2

The second version of PIPE was published in June of 2008 [32]. The main goal of the second version of PIPE was to perform a proteome-wide, all-to-all screen of the proteins in yeast. As mentioned in the previous sections, to be able to do this two main improvements would have to be made. First of all, the running time would have to be improved. Trying to run the original PIPE program on the approximately 20,000,000 possible protein pairs that exist in yeast would have taken decades, which is obviously unacceptable. The second thing that had to be dealt with was the false-positive rate. As previously mentioned, with a specificity of 89% the original PIPE algorithm would generate 2,200,000 false positives which is significantly more than the number of actual interacting pairs of proteins. The following section will describe how these two obstacles were overcome. Following that, a look at the results and a short analysis of their implications will come.

3.2.1 Algorithm Improvements

PIPE was initially designed to predict the likelihood that two given query proteins interact. To allow for an all-to-all protein pair scan, PIPE was parallelized to reduce the overall running time. However, even parallelized, it was estimated that PIPE would still need at least 50 years to predict all possible pairs of yeast proteins. As previously noted, significant improvements to PIPE's core algorithm had to be made to allow a realistic running time for an all-to-all protein pair scan. There were three specific improvements introduced that significantly improved PIPE's running time.

1. The character based amino acid representation of the proteins was converted into binary, significantly decreasing the time needed to compare protein fragments. This rather simple change removed the need for a character-to-index mapping function when adding up the PAM120 scores. This improvement produces a speedup of 18 times on average for a set of 1000 randomly chosen protein pairs.

2. The process of "sliding the window" across a protein was the second aspect that was improved. It was identified that, to move from one fragment to its

immediate neighbour, only a single character deletion (the first character) and a single character addition (to the end of the sequence) was needed instead of completely recreating the string within the window at each stage of the algorithm. The similarity of these neighbouring fragments meant that a significant number of PAM120 lookups were no longer required. Implementing this change produced a speedup of 43 times on average for a set of 1000 randomly chosen protein pairs.

3. The fact that a lot of identical protein fragment comparisons were repeated during the computation of the PIPE algorithm was the third item addressed. Two steps were needed to remedy this situation. First, all possible protein fragment comparisons are pre-computed and the pairs that are deemed to match are recorded to disk. When a set of query proteins are presented to the system, all relevant fragment comparison results are loaded into main memory where PIPE can simply query them instead of computing the similarity between two given protein fragments. This step removed an enormous amount of redundant work being performed by PIPE. Performing the lookup as opposed to computing how similar two fragments were produced a speedup of 16,150 times on average for a set of 1000 randomly chosen protein pairs.

These speedups are summarized in Table 3.1.

Version	Average Runtime (s)	Speedup
Original PIPE implementation	6944.40	1x
+Digital alphabet optimization	389.65	18x
+Sliding window optimization	160.53	43x
+Pre-computation/Query approach	0.43	16,150x

Table 3.1: Performance improvements from PIPE1 to PIPE2 [32].

It should be noted that these speedup times do not take into consideration the pre-computation time. This is a relatively insignificant amount of time (roughly 30 minutes, or 1%) of the time needed to run PIPE2 on all of the possible protein pairs (roughly 48 hours on 76 processors). When this pre-computation is considered, PIPE2 has an approximate overall speedup of 14,775.

The next major task in improving the PIPE algorithm was to address the amount of false-positives that it produced. A median filter was applied to the output matrix which effectively eliminated very thin peaks (which are viewed as random anomalies and could produce false positives) but allowed the wider hill regions to survive. However, applying a median filter is computationally expensive (it was actually shown that an implementation took longer to compute than the PIPE2 algorithm itself). To combat this, the filter was modified in the following way. For a given cell c,

- If its neighbours consisted of more zeros than non-zeroes then c would be set to zero,
- Otherwise c would be set to 1.

After this modified median filter was applied, the average value of the cells of the output matrix was calculated. If this average was above a set cutoff threshold then the proteins were predicted to interact. The way in which this approach allows true positives to be detected and false positives to be weeded out can be seen in Figure 3.7 .

To improve the thresholds and parameters of PIPE, they were tuned using a true positive set and true negative set of 1,274 pairs each, using leave-one-out cross-validation. Using LOOCV, each one of the true positive pairs of proteins are removed individually from the true positive set before running that pair through PIPE2. A combination of experiments were done. First PIPE2 was tested with no median filter whatsoever and the cutoff threshold was varied 0.0 to 1.5 in 0.001 increments. The second set of tests used the median filter as described above. The filter size was varied from 3x3

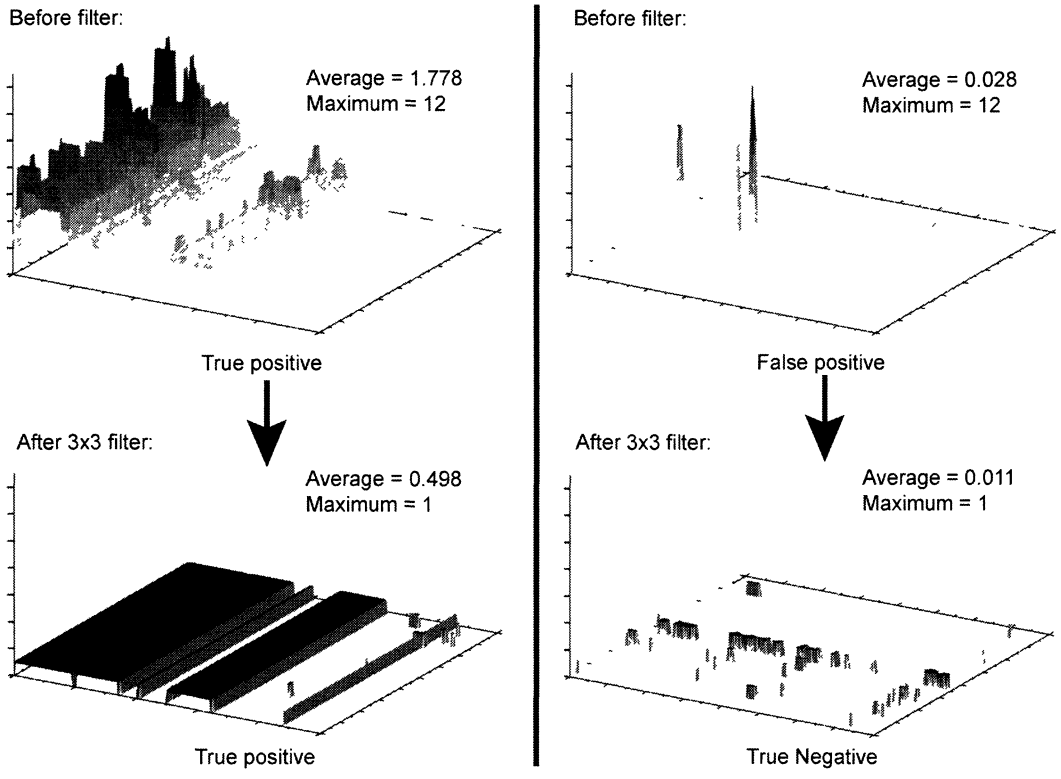

Figure 3.7: The effect of the modified median filter on true and false positive PIPE outputs. Taken from [32].

to 11x11 and the cutoff threshold was varied from 0 to 1 in 0.001 increments. The results of these experiments were that a filter size of 3x3 and a cutoff threshold of 0.45 were reasonable.

To test the specificity of PIPE2, a presumed negative set of 100,000 protein pairs was constructed. Of these 100,000, 99,946 were correctly classified as non-interacting protein pairs with 54 false positives. For the 1,274 true positives, PIPE2 correctly classified 186 pairs as interacting pairs with 1088 false negatives. Overall this suggests a specificity of 99.95% and a sensitivity of 14.6%. It should be noted that a higher sensitivity could be obtained at the cost of a lower specificity by simply adjusting the parameters. However, in this case, the higher the specificity obtainable the better.

3.2.2 Results and Analysis

When PIPE2 was run on the yeast proteome, the following stats represent the data that was available at the time:

- total number of yeast proteins: 6,304
- total number of proteins with at least one known interacting partner: 4,716
- total number of proteins with no known interacting partners: 1,588
- total number of possible interactions: 19,867,056
- largest number of known interactions partners for a single protein: 282
- smallest number of known interactions partners for a single protein: 0
- total number of known interactions: 15,118
- average number of known interactions per protein: 4.80
- average number of known interactions per protein with at least one interaction: 6.41

PIPE2 was run on all 19,867,056 possible pairs of yeast proteins. From these tests, PIPE identified 29,589 positive interactions. Of these positive results 15,151, or roughly 51.2%, had been perviously reported as interacting proteins. This left 14,438 pairs as novel interactions. When comparing the total number of unique participating proteins and interactions predicted by PIPE2 to the number of unique proteins and total interactions reported by experimental methods, it became obvious that PIPE2 can investigate a significant number of possible protein-protein interactions that cannot be processed by experimental methods.

To investigate the validity of the interactions predicted by PIPE2, three sets of 100 random protein pairs from the identified novel interactions were gathered. These pairs were thoroughly investigated to determine the common functional information

for each pair. Of these 300 pairs, 59 of the predictions can also be supported by a functional relationship. A second line of validation used was to check each pair for common interactors. 140 of these 300 pairs can be supported by a previously reported common interaction. Overall, 39 pairs were supported by both of these lines of validation and 199 of the 300 were supported by at least one. These findings are summarized in Table 3.2.2.

Set Number:	1	2	3	Total
Evidence Based On Function	20	22	17	59
Evidence Based On Third Interaction	49	45	46	140
Both Lines of Evidence	17	13	9	39
Either Line of Evidence	69	67	63	199

Table 3.2: Analysis of PIPE2 data using functional relationship and the presence of a common interactor to support the predictions made [32].

A further investigation into PIPE2's novel predictions allowed researchers to identify potential pairs of proteins that are involved in non-homologous end-joining (a DNA repair mechanism). A specific novel interacting pair was identified that could potentially be involved in NHEJ and was then confirmed in laboratory experiments. This indicates that meaningful novel biological information can be extracted from the interactions that PIPE2 predicted.

3.2.3 Discussion

Even though the PIPE2 software was a vast improvement over the original PIPE code, it still had some major downfalls. It became clear that although PIPE2 could successfully do proteome-wide predictions for organisms like yeast it could not attempt to tackle more complex organisms such as *C.Elegans* and certainly not *Homo Sapiens*. The main reason for this is because PIPE2 was embarrassingly parallel and no attempt was made to improve or optimize how multiple instances of PIPE2 ran at the same

time. The scheme was simply to run PIPE2 on a static list of proteins on each available node on a given cluster. This scheme has two inherent flaws. The first is that there simply was no load balancing whatsoever. The reason for this is that the time to make a prediction on a pair of proteins can vary significantly. While PIPE2 can compute a prediction on most protein pairs in a fraction of a second, there are a small percentage of "hard pairs" which take significantly longer (sometimes several orders of magnitude longer). This is because some proteins are very "popular" in the known interaction list, meaning that it has a lot of adjacent neighbours in the known interaction graph. If the PIPE algorithm determines that a query protein "matches" a significant number of "popular" proteins then the PIPE algorithm must visit each one of its neighbours to check for similarities to the other query protein. This is the main contributor in making a given query protein pair a "hard pair". If a given instance of PIPE2 was given a relatively easy list of protein pairs to compute predictions for and happened to finish well before all of the other processes currently running on other cores it would simply stop and not help out which results in wasted resources. Similarly, if another PIPE2 process was given an unusually hard list of pairs to compute, all of the other process would finish well before it and would not be able to help out. In an ideal situation, there would be a single process in charge of distributing work to the other process, ensuring that no process stopped working until there was no more work to be done.

The second flaw of the PIPE2 parallelism approach is the most damning; for more complex organisms PIPE2 would simply crash due to a lack of memory. Since there was no resource sharing or communication between any PIPE2 processes, each process would have to load its own instance of the known interactions graph on top of the needed database files, the intermediate and result matrices and all other PIPE2 related data needed to perform its predictions. On top of this, PIPE2 would load all of the pre-computed database files into memory that it would need for its entire run. Storing all of this data in memory was simply too much for most cluster nodes and would cause PIPE2 to crash, especially when trying to make predictions on more complex organisms. This behaviour was specifically observed when attempting to

make predictions on both *C.Elegans* and *Homo Sapiens* protein pairs.

Chapter 4

PIPE Version 3: Massively Parallel PIPE

A new version of PIPE, referred to henceforth as PIPE3, was designed and implemented to address the major limitations of the PIPE2 implementation. As stated above, the two main areas of concern were load unbalancing and inefficient memory usage. A new parallel architecture was designed to address these issues with the added benefit of being flexible enough to scale to any given physical cluster architecture, allowing PIPE3 to make full use of any and all resources made available.

4.1 Parallel Architecture

There are two main components to the new parallel architecture of PIPE3. To address the issue of load imbalances a distributed dynamic load balancing scheme was used. As opposed to PIPE2's crude static load balancing scheme (which would simply divide the total amount of work between the total number of PIPE2 processes), a dynamic approach to this problem would allow PIPE3 to ensure that all available computational resources would always be busy (as long as there is still work to be done). To ensure that a relative amount of control could still be had over the overall flow of the program, the classic master/slave parallel model was used as opposed to a decentralized approach. Using this model would also allow PIPE3 to recover after a hardware crash a lot easier since it would keep all of the partial results and remaining work in a centralized location. To address the issue of inefficient memory usage, a parallel, shared memory approach was taken. The idea here would be to allow multiple PIPE processes to access data in shared memory instead of each of them duplicating all of the data themselves. This would be accomplished through the classic all-slaves parallel model.

4.1.1 Master/Slave Model

In a typical master/slave parallel model there is one master process which controls the overall flow of the program and a series of slave processes that actually carry out the intended work. This model was introduced to the overall parallel architecture to reduce the load imbalances present in the old PIPE2 parallel scheme. It would work in the following way:

- The single master process would be in charge of loading the protein pairs to be computed as well as collecting the results. It would also distribute work to the slave processes in relatively small (to the overall amount of work to be done) packets.

- The slave processes would be responsible for actually running the PIPE algorithm on the packets of work delivered by the master process. Once it completes its work, it requests another packet of work and returns its results from the previous packet.

This idea is illustrated in Figure 4.1.

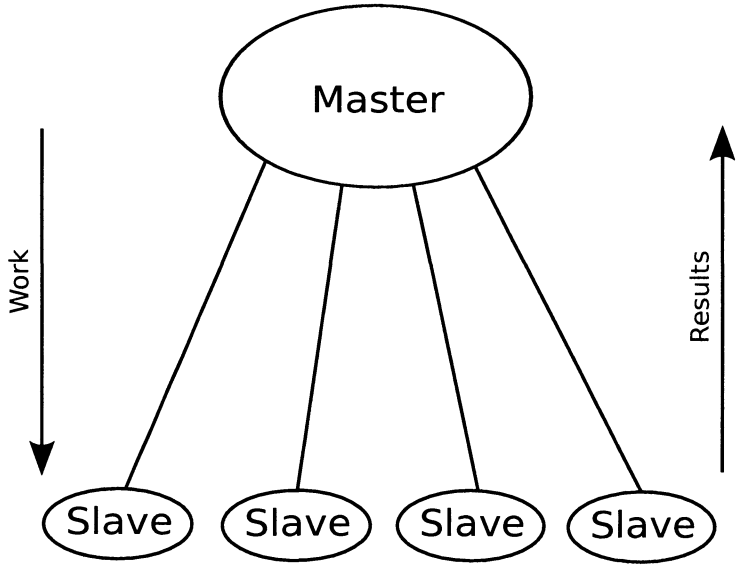

Figure 4.1: The general master/slave parallel model. The master process distributes work and collects results and the slave processes are responsible for actually carrying out the work and reporting back to the master.

This relatively simple scheme almost completely removes the load imbalances present in the PIPE2 parallel scheme. By only giving a slave processes a relatively small amount of work at a time, this ensures that no single process will get bogged down for an overwhelming amount of time while the other processes sit idle. If a slave does get stuck with an abnormally hard packet's worth of work, the other slaves will continue to work on their packets and, if they finish, they will request more work from the master process and continue to work ahead. The only time that a given slave will not be working while other slaves are is when the master process has no more work to distribute. However, when the amount of work increases and the packet size is small enough, the amount of wasted resources goes to zero and is certainly less than the amount of wasted resources that PIPE2 endured. It should be noted however that if the packet size is too small then the amount of communication between the master and slave processes will negatively impact the running time of the overall program. It is important to balance the packet size between being too small (increasing communication overhead) and too big (increasing the wasted resources at the end of a given run). The main differences in the running times between a PIPE2 run and a PIPE3 run are illustrated in Figure 4.2. In this figure, each bar represents a set of protein pairs to run predictions on, and the length of the bar represents how long it takes to do the computation. In the PIPE2 scheme, the total amount of work was simply split up between the number of processes that would be run. Since the computation time can vary so much, the inevitable load imbalances become apparent. This can then be compared to the way in which PIPE3 would tackle the same data set. Instead of coarsely dividing the work up between the processes, the work would be divided up into much smaller packets and distributed appropriately. The improvement in resource utilization and reduction in the overall running time then becomes apparent.

Since the user has control over the packet size, they can ensure that the master process isn't overwhelmed with work requests from the slave processes by ensuring that the packets are sufficiently large. It was for this reason that a centralized master/slave model was used as opposed to a semi or fully distributed system. If it were the

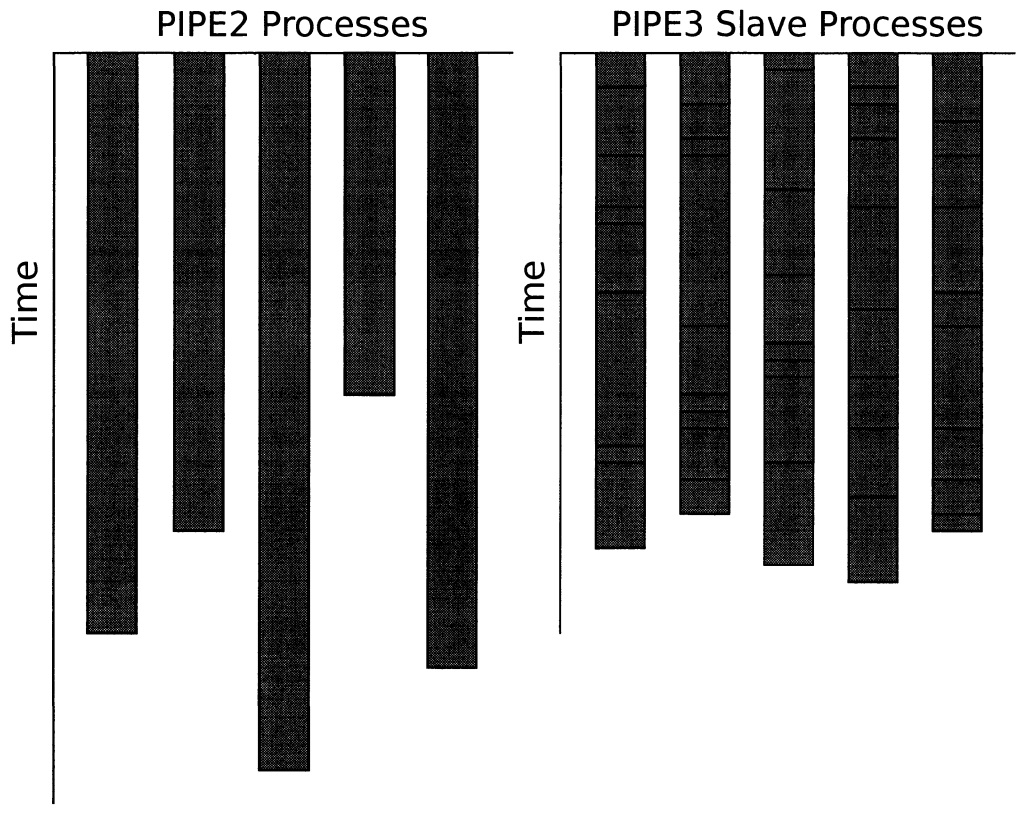

Figure 4.2: Overall running time of the PIPE2 parallel scheme compared to the master/slave approach adopted by PIPE3.

case that the master process was constantly being overwhelmed by work requests, a tiered master/slave model could have been employed. In this scenario there would be intermediate master processes between the slave processes and the main master process who would hold several packets of work and who would each be in charge of distributing work to a smaller number of slave processes. These intermediate master processes would then request work from the main master process when they needed it, significantly reducing the amount of communication that the main master process participates in. If the ratio of PIPE3 slave processes to the one master process grows significantly in the future due to the physical architecture of a given cluster then an approach like this might be necessary, but having control over the size of the packet will allow users to avoid this kind of situation for almost any typical compute cluster.

4.1.2 All-Slaves Model

In a typical all-slaves model, a single process works along until it hits a point at which parallel processing is possible. At this point it splits into a series of threads where each thread is working independently towards a common goal, at which point all of the threads collapse back into one process that moves forward. This idea is presented in figure 4.3.

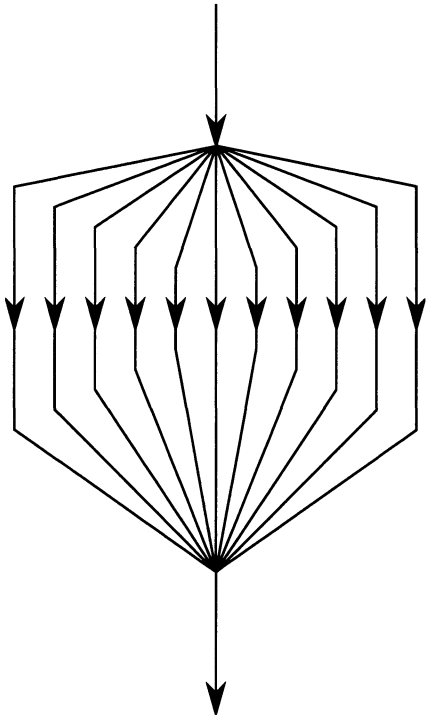

Figure 4.3: The general all-slaves parallel model. Each line represents a single computational process. At the start of computation, there is a single process. At a given point, this process splits up into multiple threads who all work independently. When they are all finished, they join back together into a single computational process.

This approach was implemented to reduce the memory inefficiencies of PIPE2. The main idea is that, in a slave process, all of the information that can be shared between independent PIPE processes would be shared instead of having multiple instances of the same information in memory. The main contributor to memory inefficiencies in PIPE2 was the fact that the relatively large interaction graph was loaded by each

PIPE2 process, even if they were on the same node. Now, a single interaction graph will be loaded per slave process and then the slave would split up into a number of threads (ideally using all of the available computational resources) who would work independently of one another while all accessing a single instance of the interaction graph stored in shared memory. On top of this, the pre-computed database files were not all loaded at once at the start of the computation, they were loaded only when they were needed. This allowed more threads to run simultaneously on a given cluster node by reducing the overall memory usage.

4.1.3 Putting It All Together

The master/slave aspect of PIPE3 was implemented using MPI and the all-slaves aspect was implemented using openMP. For the master process, the general program flow can be seen in Algorithm 1.

Algorithm 1: PIPE3 master process.

load protein pairs into packets
while *packets remain* **do**
    receive work request from slave x
    receive previous results from slave x
    send packet to slave x
    write results to output file
foreach *slave process* **do**
    receive work request from slave x
    receive previous results from slave x
    send KILL_SIGNAL to slave x
    write results to output file

The PIPE3 master process is relatively simple. After loading the protein pairs to be computed from a file into packets, it simply waits for work requests from the slave processes. While it has work to be computed it will distribute it to the slaves as they request it as well as receiving previous results and writing these results to file. Once

the master process has run out of work, it answers any further requests with a signal to stop and exits itself once all-slaves have been told to stop.

The general program flow of the slave process(es) can be seen in Algorithm 2.

Algorithm 2: PIPE3 slave process.

```
load protein interaction graph
current_packet ← ∅
current_results ← ∅
work_available ← TRUE
foreach thread in parallel do
    while work_available do
        if current_packet = ∅ then
            request work from master
            send current_results to master
            receive message from master
            if message = KILL_SIGNAL then
                work_available ← FALSE
                BREAK
            else
                current_packet ← message
        retrieve pair from current_packet
        run PIPE algorithm on pair
        add results to current_results
```

Although slightly more complex than the master process, the PIPE3 slave process is still relatively simple. The first four lines represent the data that the independent threads will share: the protein interaction graph needed by the PIPE algorithm, the packet of work that the slave is currently working on, the recently produced results and a flag representing whether or not there is still work to do. Once the interaction graph is loaded and the flag is set appropriately, the slave process splits into a user

defined number of threads. From here, each thread (while there is work available) first checks if the packet they are working on still has pairs to compute. If not, the thread requests more work from the master process. When the master process responds, the thread checks if the message is a signal to stop. If it is, it sets the *work_available* flag appropriately and exits. If not, it sets the incoming packet up to be worked on by itself and the other threads and continues. When there is work available in the current packet, each thread simply takes a pair from the packet, runs the PIPE algorithm on that pair (using the shared protein interaction graph in the process) and then adds the result to the results array. It should be mentioned that the first thread to notice that the current packet is empty will request more work from the master. In other words, it will not wait for the other threads to finish what they are working on before requesting more work, which was an initial design idea. However, it was soon discovered that this would lead to a lot of threads wasting a significant amount of time. The decision was made to make the first thread to realize that there was no work in the current packet request another packet, even though this was more difficult to implement and increased the amount of code within critical regions (code that can only be executed by a single thread at a time). The difference between these two approaches can be seen in Figure 4.4, with the rejected design option on the left and the accepted design option on the right.

For an example overview of the program flow of PIPE3 with one master process and three slave processes, refer to Figure 4.5.

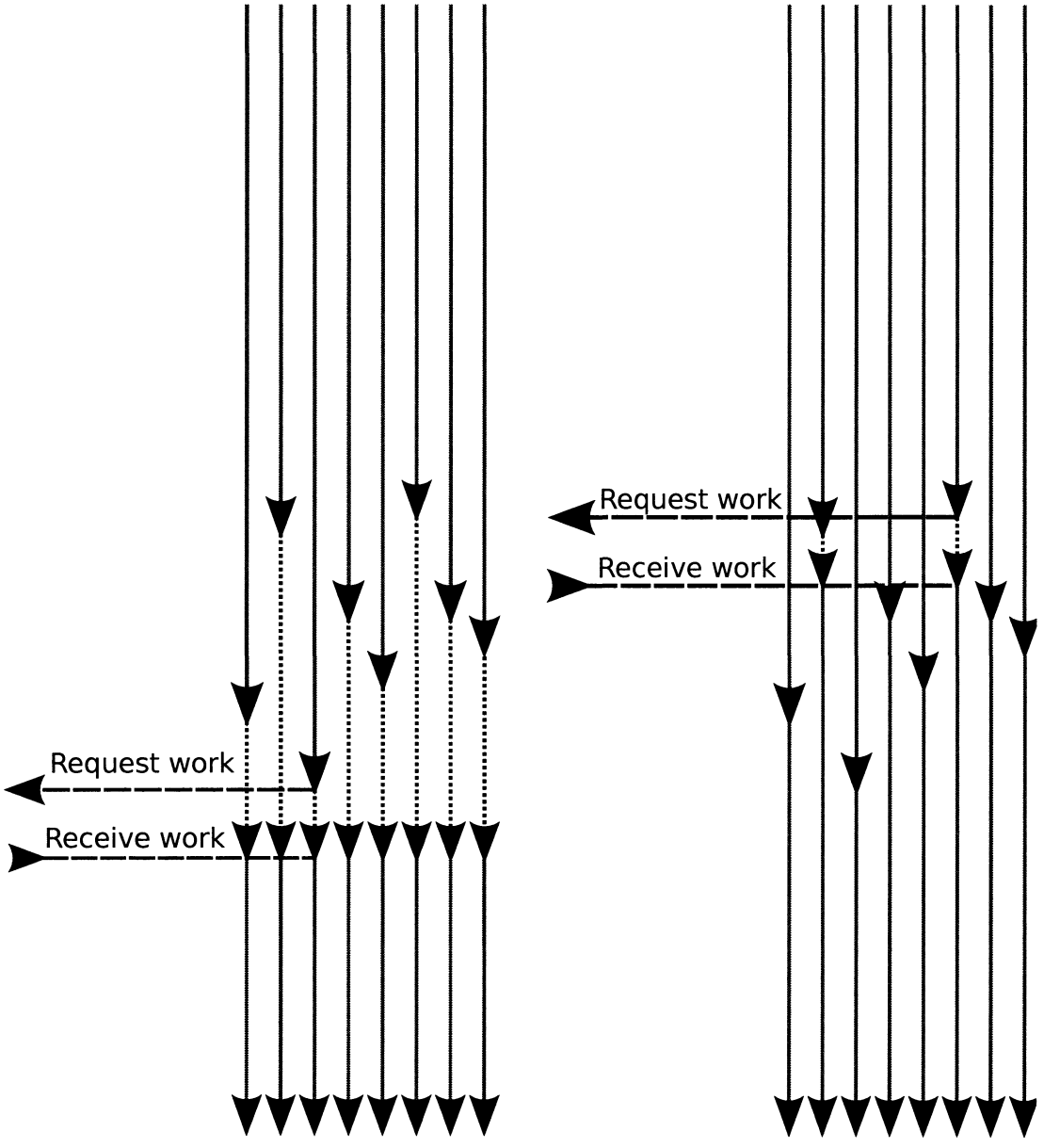

Figure 4.4: PIPE3 slave process work request design options. The arrows represent PIPE3 slave process threads with the red arrows (above the dashed arrows) representing jobs being processed from the packet A, green arrows (below the dashed arrows) representing jobs being processed from the packet B and the dashed arrows represent idle threads.

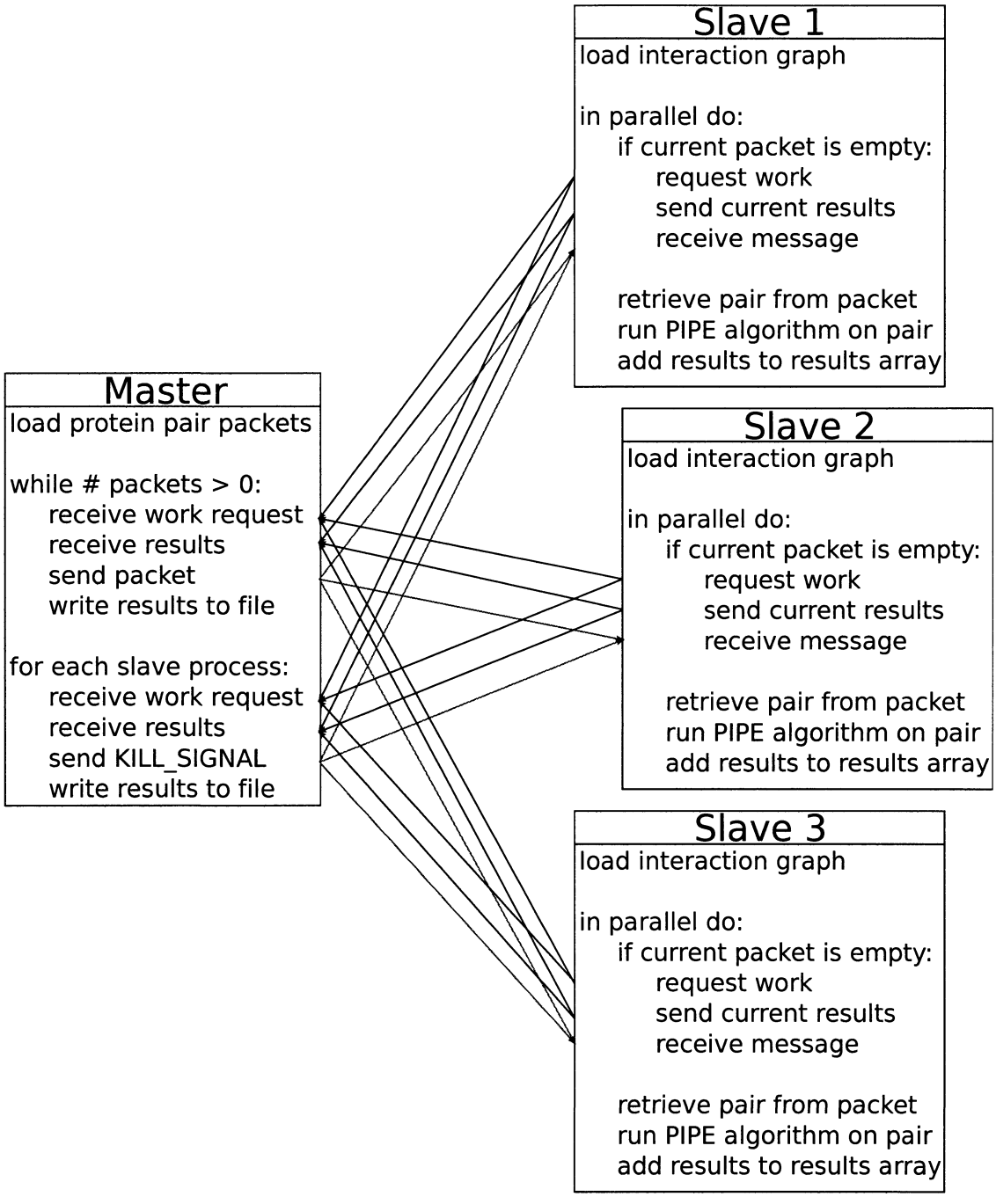

Figure 4.5: An example of the overall communications as well as an abbreviated process flow of PIPE3 running with one master process and three slave processes

4.2 Parametrization

To enhance the overall flexibility of PIPE3, every possible runtime configuration was paramaterized, as opposed to them being hardcoded in previous versions. With PIPE3, the following parameters can be specified at runtime:

- **Number of Processes** - The total number of processes used for a given run. If n processes are run, there will be one master process and $n-1$ slave processes.
- **Input File** - A file containing a list of query protein pairs.
- **Output File** - A file to which the results will be printed.
- **Interaction Graph** - The location of the interaction graph file for a given organism.
- **PIPE Database** - The location of the PIPE database files for a given organism.
- **Window Size** - The size of the sliding window used to compare protein segments. The typical value is 20.
- **Filter Size** - The size of the filter used on the result matrix. The typical value is 3.
- **Accept Threshold** - The minimum average cell value in the result matrix accepted as a "positive" result. To return all results the threshold is set to 0.
- **Packet Size** - The number of query protein pairs to be included in a packet sent from the master process to a slave process.
- **Threads** - The number of threads that each slave process will use to compute prediction on query protein pairs in parallel.
- **Big Endian Flag** - A flag to signal wether the current architecture uses big endian or little endian representation. Can be set to 0 or 1.
- **Time Limit** - A limit on the amount of time a given process can work on a pair of proteins.

In the past, if one wanted to run the PIPE code on a different organism than the one previously ran, the code would have had to be changed which would have required a recompilation. Now the user will be free to run PIPE3 on any organism for which they have the appropriate data (interaction graph and database files), on any scale they wish (by setting the number of processes to be run and the number of threads to be used) as well as have complete control over the PIPE parameters (window size, filter size, accept threshold) if they wish. This code lends itself to being a part of an overall automated system much better than past versions of PIPE did.

4.3 Running PIPE3

The main difference between running PIPE2 and PIPE3 is that the user must decide on what scale they wish (and are able) to run the code, all other parameters can be taken from the previous PIPE2 configuration. PIPE3 needs at least 2 processes to run (a master and at least one slave), which would be comparable to running PIPE2. The number of processes run is generally dependent on the number of interconnected shared-memory nodes available. These nodes can be anything from a series of desktop computers networked together to a large scale cluster node. The optimal master/slave configuration would be to have one slave run on each available node. Having more slaves than nodes would cause at least one node to have at least two slaves which would lead to memory inefficiencies. Running less slaves than nodes available would result in unused nodes. So, if there are n nodes available, PIPE3 should be run with $n+1$ processes (n slave processes and one master process). The master process will sit along with a slave process on one of the nodes. Since the master process doesn't perform any computationally intensive operations, the performance of the slave process it is matched with will not suffer significantly.

The number of threads that should be used per slave process depends on the computational capabilities of the nodes. Generally the number of threads should be on the order of the number of compute cores (or, in the case of cores with hardware supported mulit-threading, the total number of hardware supported threads) on each

node. One should expect that the performance to increase as the number of threads used approaches the total number of cores per node and, in most cases, will continue to increase as the number of threads used surpasses the total number of cores per node. This "overloading" can cause a performance increase by swapping out threads currently waiting on I/O or other slow system operations to threads that are ready to compute something. The time that would have been wasted waiting was used to further the progress of the overall computation, which leads to a performance increase. This behaviour peaks at some point, so care needs to be taken when selecting the number of threads to be used for a given run.

Chapter 5

Benchmarking Results

Benchmark testing was performed on three clusters: Carleton's Bioinformatics Lab cluster, HPCVL's Beowulf cluster and HPCVL's Victoria Falls cluster. Each set of benchmarks contains two separate tests. The first test is designed to show how well the code scales as more threads are used by a slave process. This test is done by using a single slave on a single cluster node, first starting with one thread and then increasing the threads until a visible peak is found. This is the manner in which the optimal number of threads per slave process is found. The second test is to show how well the code scales when more nodes (and, by extension, more slave processes) are added. Each slave will use the optimal number of threads as discovered in the first test. Most of these tests are done on a set of 5,000 random yeast protein pairs except for some of the later tests on the Victoria Falls cluster which required larger data sets. For these tests sets of 50,000 and 500,000 random yeast protein pairs were used. All reported running times (and, by extension, the resulting speed ups) are averaged over 100 runs.

5.1 Carleton Bioinformatics Lab Cluster Benchmarking

The first benchmarking tests done with the PIPE3 software was done at Carleton's local Bioinformatics lab cluster. The cluster is made up of 8 nodes interconnected by a gigabit switch. Each node has:

- an Intel Quad Core processor (1.6GHz)
- 8 GB DDR2 RAM
- 320 GB hard drive used by the OS and for temp files

At the time of these benchmarks 6 nodes were available.

The first set of tests were done to determine the optimal number of threads to run for each slave process, as described above. The overall results are displayed in Table 5.1, with the running time and speed up trends plotted in Figures 5.1 and 5.2 respectively.

Number of Threads	Average Running Time (s)	Speedup
1	2318.547131	1
2	1213.482118	1.911
3	835.571807	2.775
4	777.797144	2.981
5	742.669497	3.122
6	746.089079	3.108
7	745.651284	3.109
8	775.769684	2.989
9	790.394030	2.933
10	804.553819	2.882

Table 5.1: Carleton Bioinformatics Lab Cluster Benchmarking Test 1: The running time of PIPE3 on 5,000 random yeast pairs vs. the number of threads used by a single slave process.

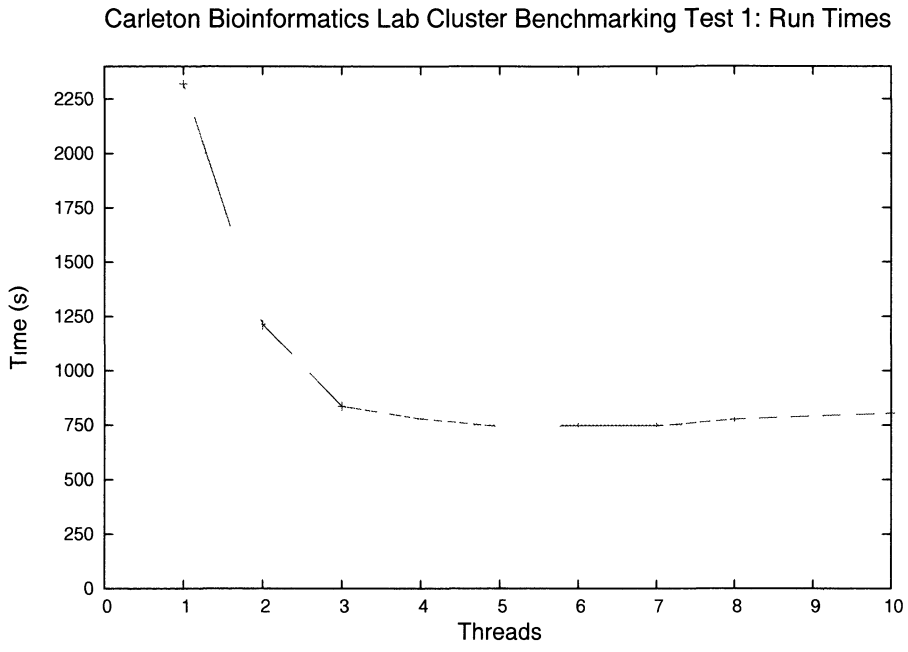

Figure 5.1: Runtime trend from benchmarking test 1 on the Carleton Bioinformatics Lab Cluster.

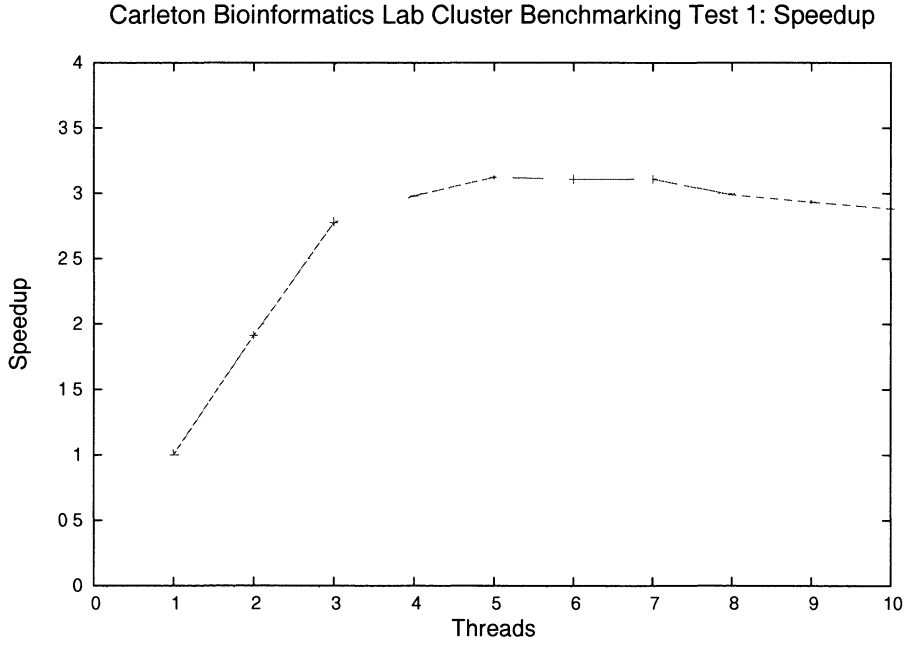

Figure 5.2: Speedup trend from benchmarking test 1 on the Carleton Bioinformatics Lab Cluster.

As can be seen in the above results, 5 threads per slave process was determined to be optimal with a speedup of 3.122. The next test was done to examine how the code scaled when more nodes are added in. The same random 5,000 yeast pairs was used as the data set and, after each set of runs, another node was added in. Using the information from the previous experiment, each slave was run using 5 threads. A summary of the results can be seen in Table 5.2, with the running time and speed up trends plotted in Figures 5.3 and 5.4 respectively.

Number of Slave Processes	Average Running Time (s)	Speedup
1	742.669497	1
2	331.018381	2.244
3	232.867967	3.189
4	168.935589	4.396
5	131.902550	5.630
6	107.919543	6.882

Table 5.2: Carleton Bioinformatics Lab Cluster Benchmarking Test 2: The running time of PIPE3 on 5,000 random yeast pairs vs. the number of slaves used. Each slave ran with 5 threads.

Based on these experiments, we can see that a maximum speedup of 3.122 can be achieved on a single node by using 5 threads and that the code seems to scale super-linearly when more nodes are added into the mix. This behaviour can be expected to continue as the amount of work and number of nodes used increases.

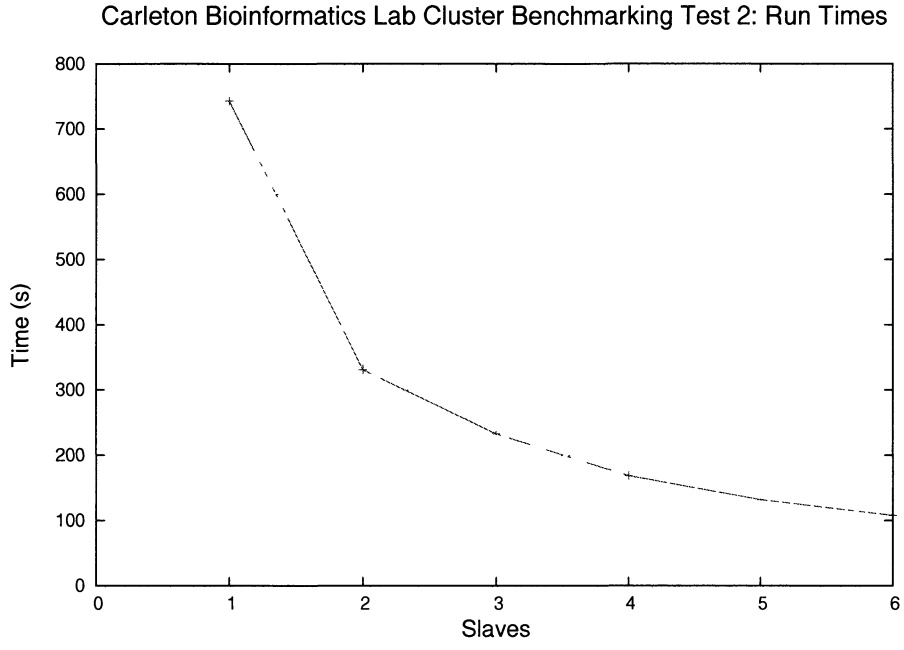

Figure 5.3: Runtime trend from benchmarking test 2 on the Carleton Bioinformatics Lab Cluster.

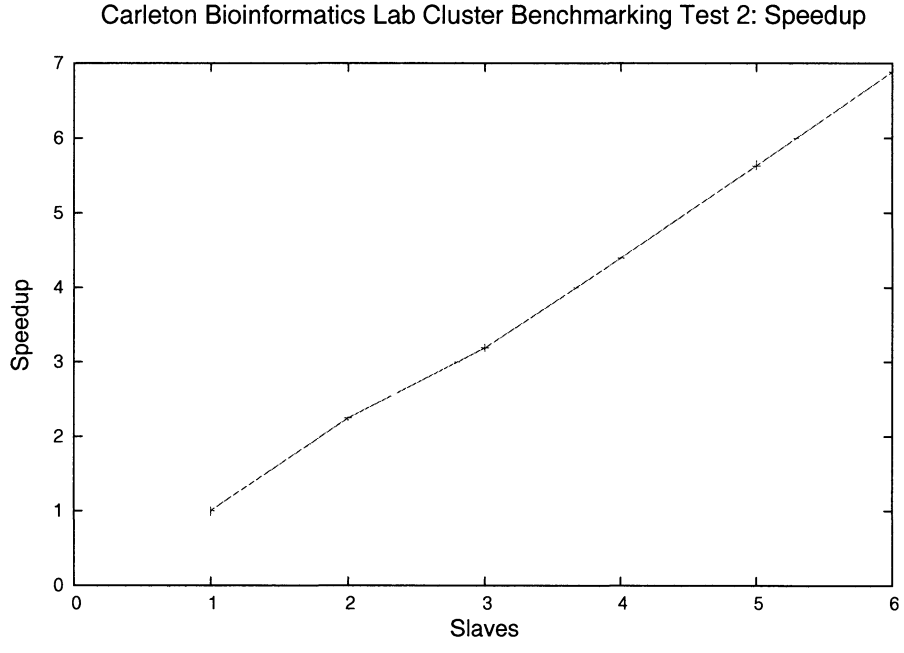

Figure 5.4: Speedup trend from benchmarking test 2 on the Carleton Bioinformatics Lab Cluster.

5.2 HPCVL's Beowulf Cluster Benchmarking

HPCVL's Beowulf cluster is located at Carleton University in Ottawa and has been heavily used by earlier versions of the PIPE software. The cluster is made up of 64 nodes interconnected by a gigabit switch. Each node has:

- 4 Opteron Cores (2.2 GHz)
- 8 GB RAM
- 3 TB of total disk storage

Again, the first set of tests were done to determine the optimal number of threads to run for each slave process, as was done on the previous benchmarks. The overall results are displayed in Table 5.3, with the running time and speed up trends plotted in Figures 5.5 and 5.6 respectively.

Number of Threads	Average Running Time (s)	Speedup
1	3750.0589016	1
2	2027.0536836	1.850
3	1423.47368383	2.634
4	1359.32627749	2.759
5	1356.51471854	2.764
6	1372.91203179	2.731
7	1368.82533481	2.740
8	1406.15620157	2.667
9	1427.07301278	2.628
10	1441.51278317	2.601

Table 5.3: HPCVL's Beowulf Cluster Benchmarking Test 1: The running time of PIPE3 on 5,000 random yeast pairs vs. the number of threads used by a single slave process.

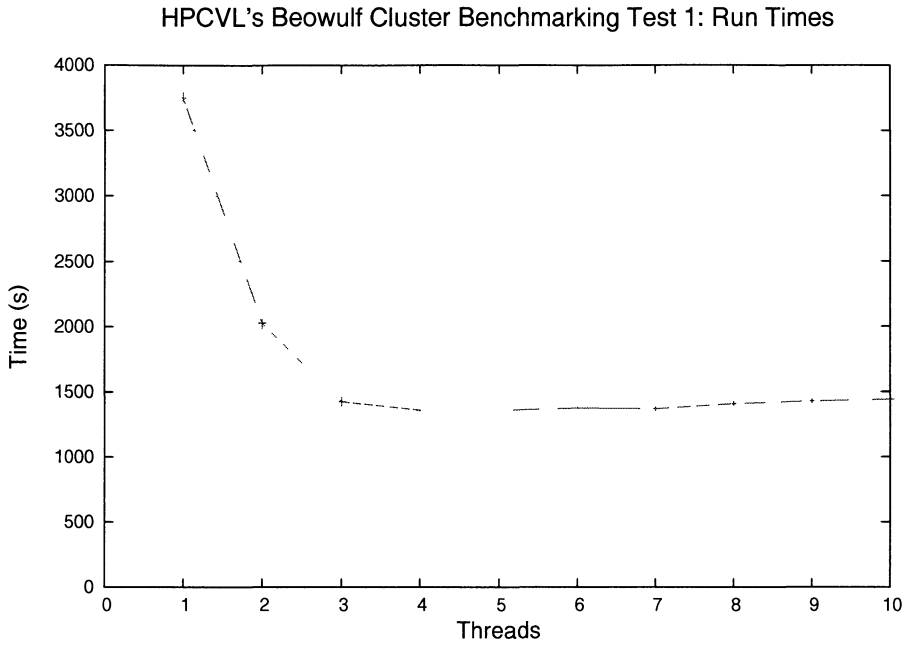

Figure 5.5: Runtime trend from benchmarking test 1 on HPCVL's Beowulf Cluster.

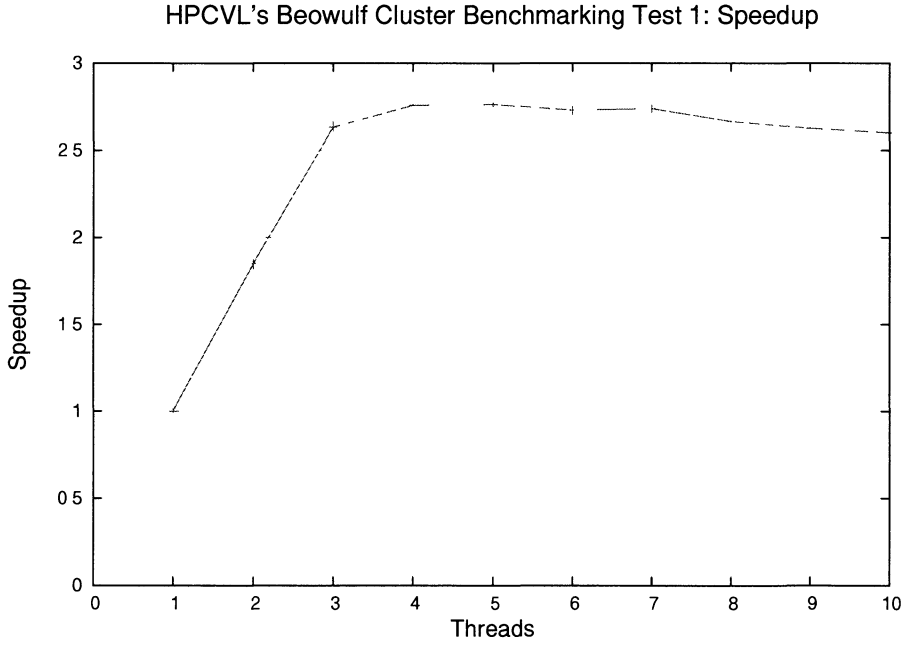

Figure 5.6: Speedup trend from benchmarking test 1 on the HPCVL's Beowulf Cluster.

As can be seen in the above results, 5 threads per slave process was determined to be optimal with a speedup of 2.764. The next test was done to examine how the code scaled when more nodes are added in. The same random 5,000 yeast pairs was used as the data set and as before, after each set of runs, another node was added in. Using the information from the previous experiment, each slave was run with 5 threads. A summary of the results can be seen in Table 5.4, with the running time and speed up trends plotted in Figures 5.7 and 5.8 respectively.

Number of Slave Processes	Average Running Time (s)	Speedup
1	1356.51471854	1
2	574.4489908	2.361
4	283.32493562	4.788
8	145.42208586	9.328
16	68.92684697	19.680
32	34.77818442	39.004

Table 5.4: HPCVL's Beowulf Cluster Benchmarking Test 2: The running time of PIPE3 on 5000 random yeast pairs vs. the number of slaves used. Each slave ran with 5 threads.

As was seen in the previous benchmarks, when adding more Beowulf nodes to the mix we witness a super-linear speedup. This trend can be expected to continue as the amount of work and number of nodes used increase.

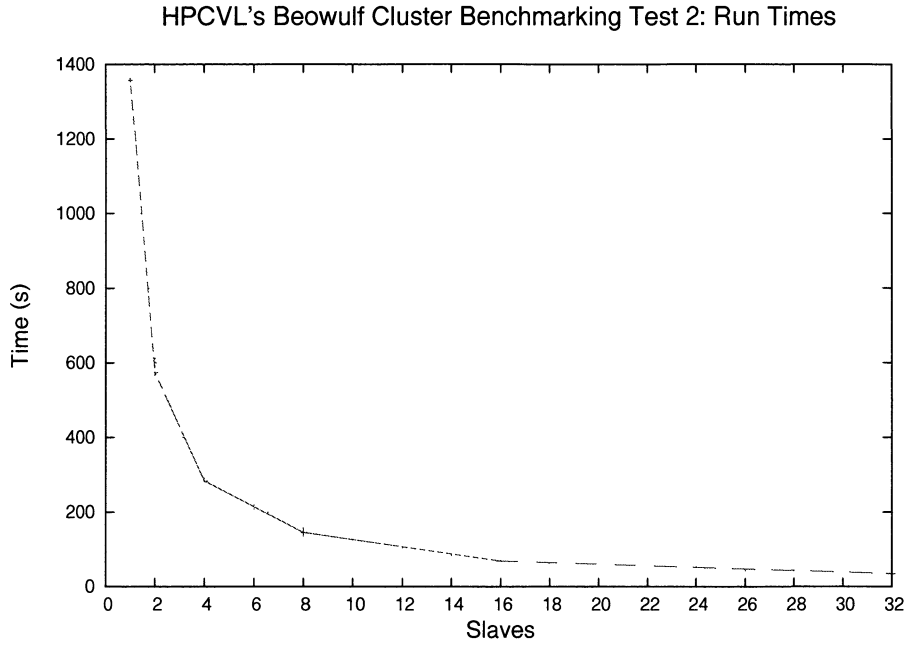

Figure 5.7: Runtime trend from benchmarking test 2 on HPCVL's Beowulf Cluster.

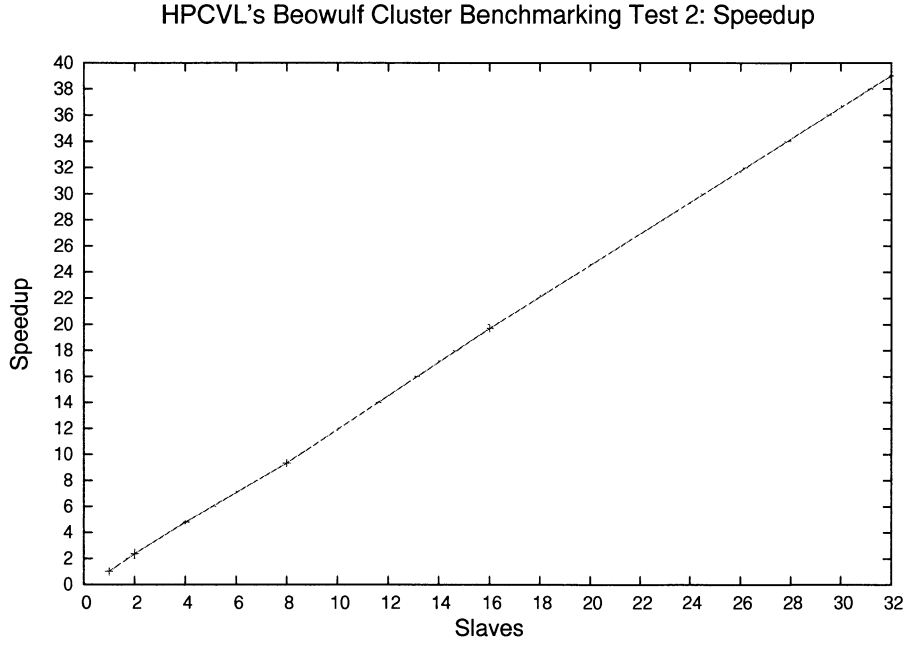

Figure 5.8: Speedup trend from benchmarking test 2 on the HPCVL's Beowulf Cluster.

5.3 HPCVL's Victoria Falls Cluster Benchmarking

HPCVL's Victoria Falls cluster is located at Queens University in Kingston and was chosen to be used because of its tremendous throughput potential. The cluster is made up of 73 nodes interconnected by a gigabit switch. Each node has:

- 2 UltraSparc T2+ chips (1.2 Ghz) with 8 compute cores per chip. Each of these compute cores has 8 hardware supported threads giving each node a total of 128 hardware supported threads per node.
- either 64 GB RAM (10 nodes) or 32 GB RAM

Again, the first set of tests were done to determine the optimal number of threads to run for each slave process, as was done on the previous benchmarks. Due to the sheer number of hardware supported threads on a Victoria Falls node, the 5,000 random yeast pairs was not sufficient to show the full scaling capabilities of a given node. For this reason a second data set comprised of 50,000 random yeast proteins was created. The first benchmarking test on the Victoria Falls cluster would be done in the following manner. The 5,000 pair data set would be used to examine the scaling of the code with a small number of threads (1 – 16 threads) and then the 50,000 pair data set would be used for tests using 16 or more threads. The 50,000 pair data set wasn't used on the smaller number of threads simply because of the amount of time it would take to process. To calculate the speedup on the higher number of threads processing the 50,000 pair data set, the running time of a single thread to processes the 50,000 pair data set was approximated by taking the speedup from the trials on the 5,000 pair data set. Using 16 threads on the 5,000 data set a speedup of 3.861 was observed. This speedup was then applied to the time it took to process the 50,000 pair data set by 16 threads (~20,825 seconds) to approximate the running time of a single thread processing the same data set (~80,409 seconds). The overall results for the 5,000 pair data set are displayed in Table 5.5 with the running time trend plotted in Figures 5.9. The overall results for the 50,000 pair data set are displayed in Table 5.6 with the running time trend plotted in Figures 5.10. The speedups observed in these two tests were then combined to show the overall speedup in Figure 5.11.

Number of Threads	Average Running Time (s)	Speedup
1	9308.75325356	1
2	6622.24860655	1.406
4	4438.87609902	2.097
8	3259.65337563	2.856
16	2410.89934953	3.861

Table 5.5: HPCVL's Victoria Falls Cluster Benchmarking Test 1, Part 1: The running time of PIPE3 on 5,000 random yeast pairs vs. the number of threads used by a single slave process.

Number of Threads	Average Running Time (s)	Speedup
1	80408.797712014086	1
16	20825.2934437	3.861
32	15325.5586534	5.247
64	11530.1009018	6.974
128	9259.13093131	8.684
256	9302.602417	8.644
512	9241.05782809	8.701

Table 5.6: HPCVL's Victoria Falls Cluster Benchmarking Test 1, Part 2: The running time of PIPE3 on 50,000 random yeast pairs vs. the number of threads used by a single slave process.

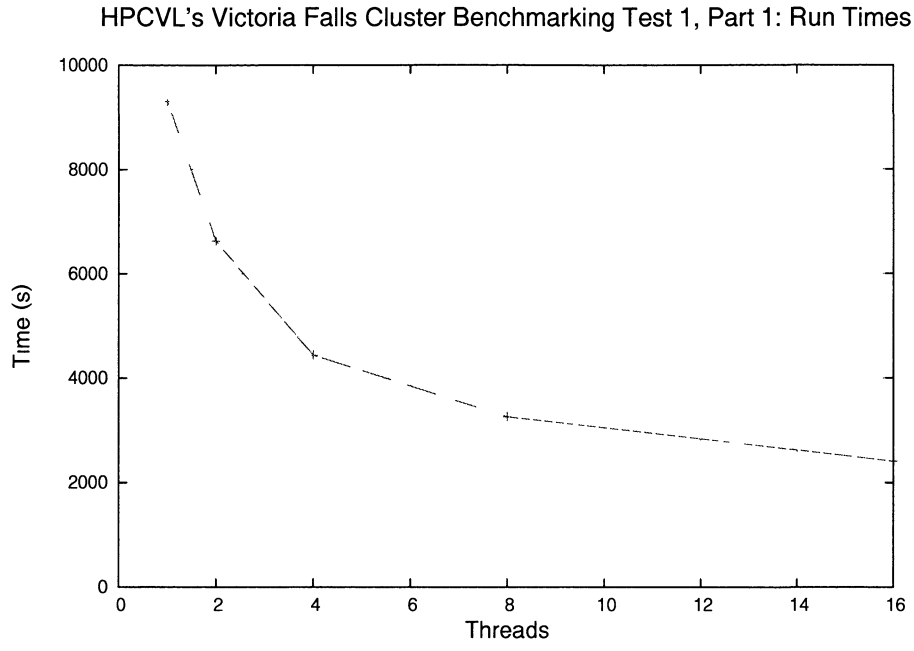

Figure 5.9: Runtime trend from benchmarking test 1, part 1 on HPCVL's Victoria Falls Cluster.

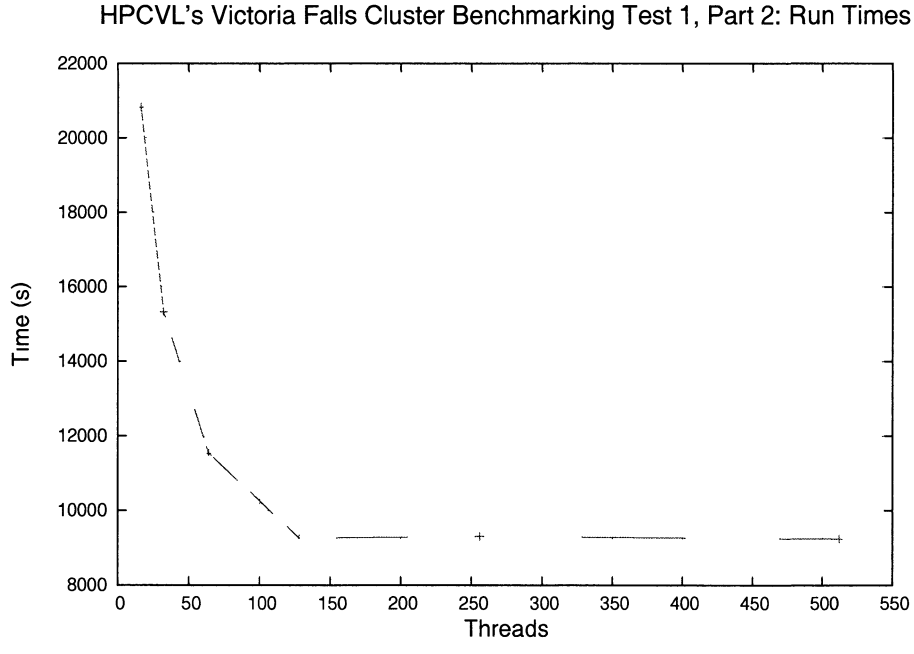

Figure 5.10: Runtime trend from benchmarking test 1, part 2 on HPCVL's Victoria Falls Cluster.

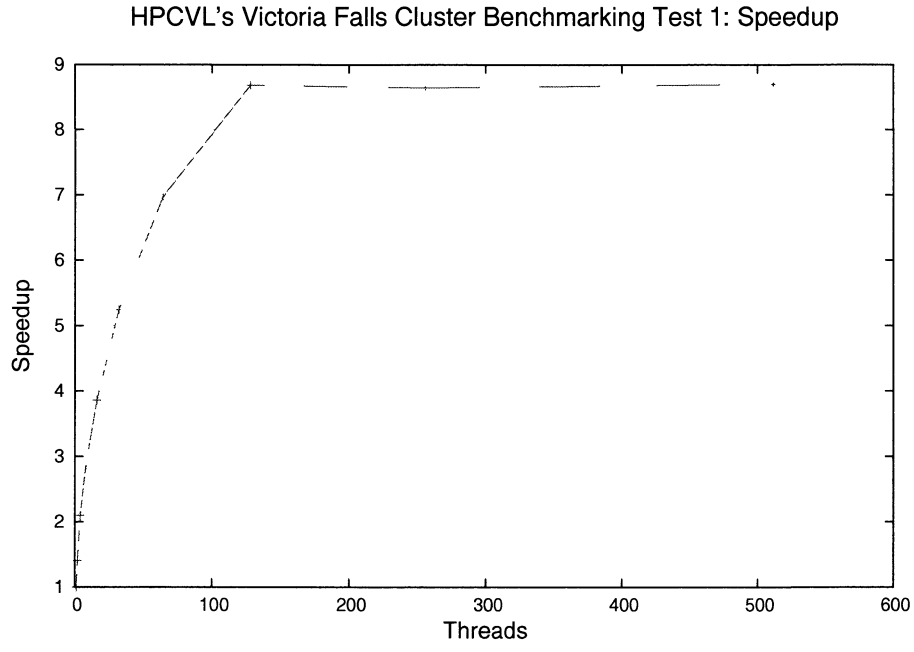

Figure 5.11: Speedup trend from benchmarking test 1 on HPCVL's Victoria Falls Cluster.

Based on the tests above, around 512 threads was found to be optimal. As was seen in previous benchmarks, once the speedup curve plateaus, the optimal number of threads is somewhere in that region. Due to the fact that virtually no change in speedup was observed from 128 to 256 threads or from 256 to 512 threads, any number of threads in this range could be expected to produce similar results. Increasing the number of threads isn't expected to produce any better results and, due to the sheer number of concurrently running threads, memory consumption could become an issue. In light of all of this, the second set of benchmarking tests on the Victoria Falls cluster was done using 512 threads. A summary of the results can be seen in Table 5.7, with the running time and speed up trends plotted in Figures 5.12 and 5.13 respectively.

Number of Slave Processes	Average Running Time (s)	Speedup
1	18244.3975384	1
2	8992.99307318	2.029
4	4571.25758775	3.991
8	2294.45482244	7.952
16	1183.31196108	15.418
32	620.47111997	29.404

Table 5.7: HPCVL's Victoria Falls Cluster Benchmarking Test 2: The running time of PIPE3 on 500,000 random yeast pairs vs. the number of slaves used. Each slave ran with 512 threads.

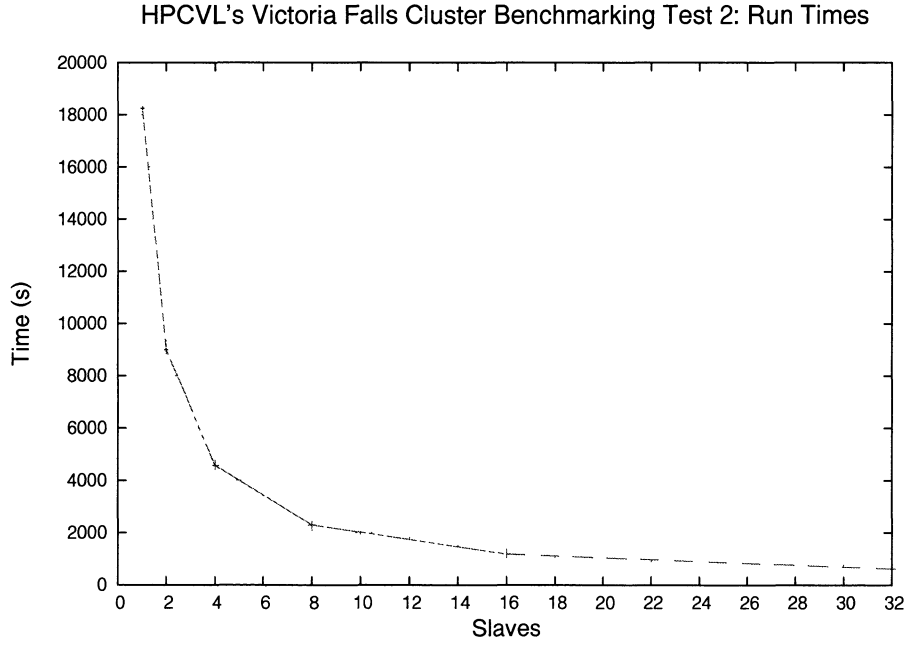

Figure 5.12: Runtime trend from benchmarking test 2 on HPCVL's Victoria Falls Cluster.

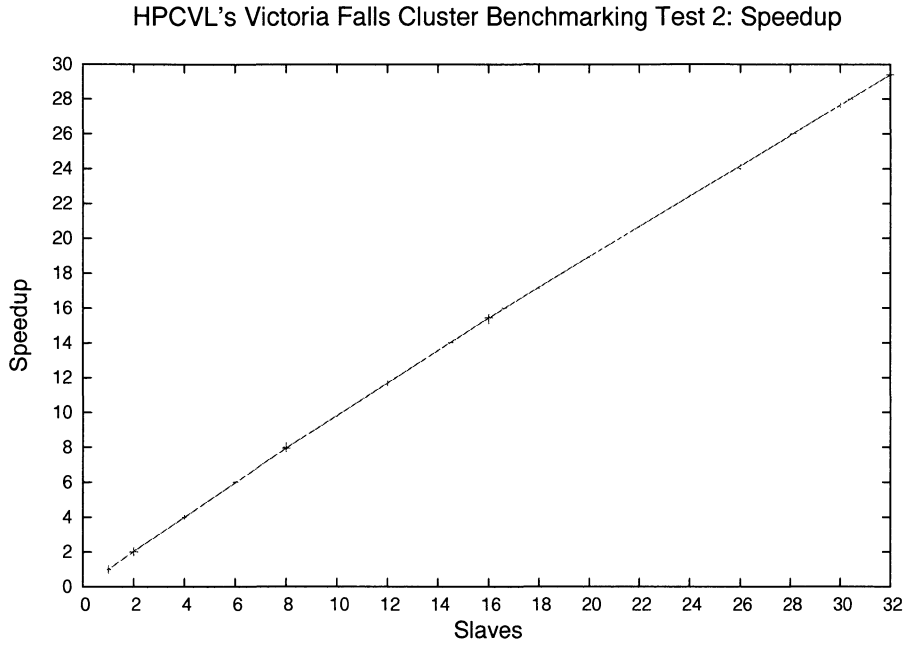

Figure 5.13: Speedup trend from benchmarking test 2 on HPCVL's Victoria Falls Cluster.

Looking at these results we see that the code scales almost linearly as the number of nodes used increases. Although the scaling isn't quite as dramatic as in the previous benchmarking tests, it is still excellent considering the vast number of threads involved. The lack of superlinear scaling with a large number of nodes can most likely be explained by the increased number of threads spending more time waiting for new packets. On the Bioinformatics cluster and on Beowulf, each node only had 5 concurrent threads which means that the chance of a thread waiting on another thread to communicate with the master process is very small. On the Victoria Falls server however, each node was run using 512 threads which drastically increases the chance that threads have to wait for communication. Regardless though, the scaling displayed by PIPE3 on the Victoria Falls cluster is still good and, as the amount of data and the size of the individual packets increase, the scaling is expected to stay relatively linear.

5.4 Overall Benchmarking Results

The overall performance of the parallel architecture of PIPE3 performed as expected. In general, the ideal number of threads to run on each slave processes is a little bit above the total number of processors or hardware supported threads on each node. The reason for this is that having slightly more threads than processors will keep all of the processors busy. The fact that we see an additional performance increase when the nodes are overloaded is most likely due to latency hiding. Since the PIPE algorithm has to make a considerable amount of I/O calls (opening the PIPE database files), the time that a given thread waits for these I/O calls can be hidden by simply swapping this thread out and swapping another thread that is ready to do some processing in. The performance increase is not overwhelming but it is certainly noteworthy.

The most important aspect of the overall performance of PIPE3's parallel architecture is how well it scales when multiple nodes are used. In this case, we generally see a linear if not superlinear speedup. The reason that it scales so well comes down to the data itself. Since the data is so irregular (with individual pairs having vastly different running times) the ability of PIPE3 to spread work around essentially smoothes out the data irregularity, causing the code to actually perform better than expected with more resources. The only time this effect is not observed is when the slave processes utilize an extremely large number of threads which leads to threads being idle while a single thread waits for new work from the master process. In general one could assume that the best performance on a given node would be using slightly more threads than the number of hardware supported concurrent processes and can expect at least linear speedup with respect to the number of nodes used. The major upside of protein-protein interaction prediction in general is that, for any given organism, there are generally millions of pairs to process which would allow PIPE3 to perform well on any given cluster.

Chapter 6

Scientific Results

6.1 *C. Elegans* Proteome-wide Prediction Results

The first major job that PIPE3 was given was to do a complete scan of the *C. Elegans* proteome. When PIPE3 was run on the *C. Elegans* proteins, the following stats represent the data that was available at the time:

- total number of *C. Elegans* proteins: 23,684
- total number of proteins with at least one known interacting partner: 3,460
- total number of proteins with no known interacting partners: 20,224
- total number of possible interactions: 280,454,086
- largest number of known interactions partners for a single protein: 512
- smallest number of known interactions partners for a single protein: 0
- total number of known interactions: 6,607
- average number of known interactions per protein: 0.55
- average number of known interactions per protein with at least one interaction: 3.82

When the time came to try and tackle the *C. Elegans* proteome, it was evident that this would be PIPE's most ambitious task to date. With nearly four times the number of proteins than yeast (which translates into over 14 times more possible interacting protein pairs to test), this would be a good test of PIPE3's abilities.

PIPE3 was run on all 280,454,086 possible protein pairs in the *C. Elegans* proteome. The work was split between the Beowulf cluster and the Victoria Falls cluster. On the Victoria Falls cluster, 50 nodes were used each with their own slave process running 512 threads. This translated into 25,600 parallel computational threads running on 6,400 hardware supported threads. This computation represents the largest job in terms of scale that has ever been run on an HPCVL machine, which is a large feat in and of itself. On the Beowulf cluster, 60 nodes were used each with their own slave process running 8 threads. This translated into 480 parallel computational threads running on 240 cores. Both clusters worked on jobs of roughly 10% of the overall data set until all of the pairs were computed. After over a week of straight computation, the proteome wide protein-protein interactions were predicted for the *C. Elegans* organism. The number of positive results depends on the cutoff value used. Refer to Table 6.1 to see the total number of predicted results for a given specificity, as well as the number of expected false positives.

Specificity(%)	Number of Positive Results	Expected Number of False Positives
99.0	2,903,322	2,804,541
99.1	2,614,164	2,524,087
99.2	2,359,614	2,243,633
99.3	2,075,830	1,963,179
99.4	1,797,746	1,682,725
99.5	1,468,575	1,402,270
99.6	1,157,629	1,121,816
95.7	858,100	841,362
99.8	593,163	560,908
99.9	242,482	280,454
99.95	136,930	140,227
99.99	37,572	28,045

Table 6.1: Number of positive results as well as the number of expected false positives for various specificities found in the overall *C. Elegans* proteome results.

At a specificity of 99.99%, 37,572 interactions were predicted, 32,548 of which are novel. This specificity was chosen to ensure a low number of false positives as well as to align the total number of predicted interactions to the number of interactions roughly expected within the *C. Elegans* proteome. To add biological credibility to these results, the protein pairs were grouped according to their molecular function, biological process and location inside the cells (as has been done in previous predictions to add confidence to the results). The percentage of pairs that have similar function, occur in the same cellular component and participate in the same cellular process is 1.3%, which is consistent with the percentage for previously reported protein pairs (1.9% for 6,607 pairs). In contrast, for randomly selected protein pairs, the percentage of pairs that have similar function, occur in the same cellular component and participate in the same cellular process is only 0.2% (for 100,000 tested random pairs). For a summary of these results, refer to Figure 6.2.

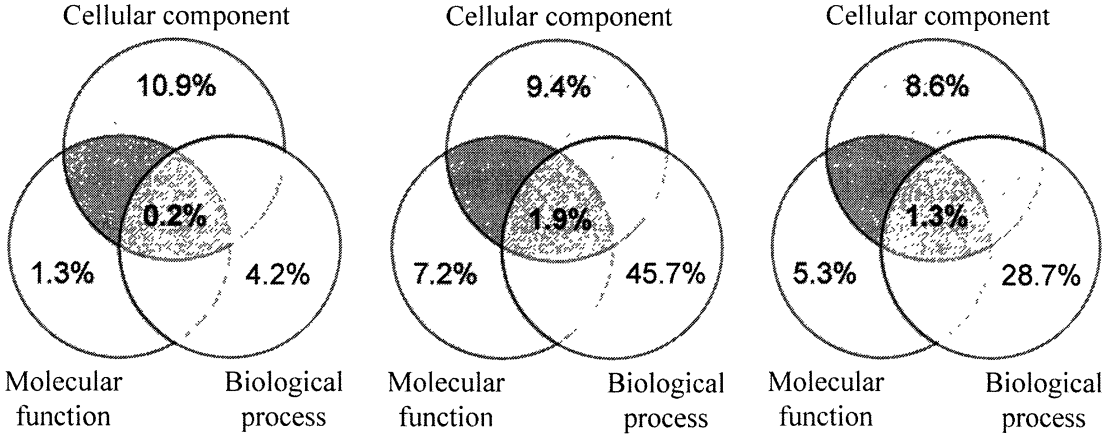

Figure 6.1: A comparison of the grouping of random protein pairs, previously detected interactions and PIPE3 predictions in terms of cellular component, molecular function and biological process for *C. Elegans* proteins.

6.2 *Homo Sapiens* Proteome-wide Prediction Results

After PIPE3 successfully tackled the *C. Elegans* proteome, it was decided to attempt a run a proteome-wide scan on the *Homo Sapiens* proteins. The following stats represent the *Homo Sapiens* data at the time it was compiled:

- total number of *Homo Sapiens* proteins: 22,513
- total number of proteins with at least one known interacting partner: 9,459
- total number of proteins with no known interacting partners: 13,054
- total number of possible interactions: 253,406,328
- largest number of known interactions partners for a single protein: 265
- smallest number of known interactions partners for a single protein: 0
- total number of known interactions: 41,678
- average number of known interactions per protein: 3.70
- average number of known interactions per protein with at least one interaction: 8.81

PIPE3 was run on all 253,406,328 possible protein pairs in the *Homo Sapiens* proteome. At this point, the *C. Elegans* proteome was by far the most complex set of proteins that PIPE had faced. The *Homo Sapiens* proteome contains almost 7 times more known interactions than the *C. Elegans* proteome, which means that the average *Homo Sapiens* protein has more than double the known interactions than a *C. Elegans* protein. This, coupled with the fact that the *Homo Sapiens* proteins are, on average, longer than the *C. Elegans* proteins, would prove to increase the complexity of scanning the entire *Homo Sapiens* significantly.

This feat was processed mainly on the Victoria Falls cluster, but the Beowulf cluster was also used to help out in certain cases. On the Victoria Falls cluster, 50 nodes

were used each with their own slave process running 256 threads. This translated to 12,800 parallel computational threads running on 6,400 hardware supported threads. The number of threads was scaled down from 512 threads used in the *C.Elegans* scan due to the fact that each individual thread needed significantly more memory. Early on in the processing of the *Homo Sapiens* proteome it was found that certain pairs were incredibly complex to compute (matching several high degree proteins in the interaction graph causing an abnormally high number of computations) which translated into an upwards of 6 days to compute a single pair. It was decided that a 12 hour time-limit would be imposed on any given pair and if a prediction on a pair took longer than 12 hours, it would simply set it aside and continue on with another pair. This is where the Beowulf cluster stepped in. Since its individual cores are much more powerful than a single Victoria Falls thread, it could compute the hard pairs a lot faster. After over 3 months, the *Homo Sapiens* proteome was completely mapped out. Again, the number of positive results depends on the cutoff value used. Refer to Table 6.2 to see the total number of predicted results for a given specificity, as well as the number of expected false positives.

Specificity(%)	Number of Positive Results	Expected Number of False Positives
99.0	2,449,598	2,534,063
99.1	2,213,514	2,280,656
99.2	1,941,294	2,027,251
99.3	1,685,394	1,773,844
99.4	1,498,687	1,520,437
99.5	1,283,021	1,267,032
99.6	1,013,199	1,013,625
99.7	758,566	760,219
99.8	510,158	506,813
99.9	306,369	253,406
99.95	172,184	126,703
99.99	67,031	25,341

Table 6.2: Number of positive results as well as the number of expected false positives for various specificities found in the overall *Homo Sapiens* proteome results.

At a specificity of 99.95%, 172,184 interactions were predicted, 130,470 of which are novel. This specificity was chosen to ensure a low number of false positives as well as to align the total number of predicted interactions to the number of interactions roughly expected within the *Homo Sapiens* proteome. To add biological credibility to these results, the proteins were grouped according to their molecular function, biological process and location inside the cells (as was done for the *C.Elegans* results and has been done in previous predictions to add confidence to the results). The percentage of pairs that have similar function, occur in the same cellular component and participate in the same cellular process is 50.3%, which is consistent with the percentage for previously reported protein pairs (67.7% for 41,678 pairs). In contrast, for randomly selected protein pairs, the percentage of pairs that have similar function, occur in the same cellular component and participate in the same cellular process is only 11.3% (for 100,000 tested random pairs). For a summary of these results, refer to Figure 6.2.

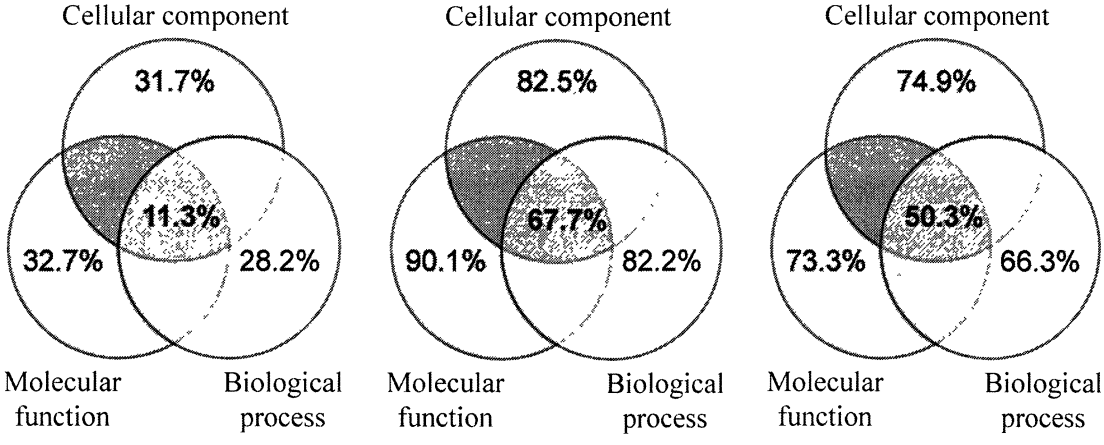

Figure 6.2: A comparison of the grouping of random protein pairs, previously detected interactions and PIPE3 predictions in terms of cellular component, molecular function and biological process for *Homo Sapiens* proteins.

Chapter 7

Conclusion and Future Work

The overall PIPE project has come a long way since its inception. The first version of PIPE proved that the underlying hypothesis of the PIPE algorithm was biologically sound as well as proved that it could identify novel protein-protein interactions. Even though this was an encouraging start, much work would need to be done to the PIPE implementation to produce some really significant biological results. Major improvements to the algorithm as well as finer parameter tuning in the second version of PIPE allowed for a complete scan of all possible protein-protein pairs in the *S. Cerevisiae* proteome, a first in the biological and bioinformatics world. Although PIPE2 was able to now produce proteome-wide protein-protein interaction predictions, it was unable to do these type of scans on more complex organisms. Due to memory mismanagement, a lack of load balancing and the lack of a comprehensive overall parallel strategy, PIPE2 simply wasn't able to handle proteome-wide scans for more complex organisms.

This is where the work presented in this thesis comes in. PIPE3 was developed to address the main issues that plagued the earlier versions of PIPE with the goal of producing some new, proteome-wide protein-protein interaction predictions of more complex organisms. To achieve this, a new parallel architecture was designed and implemented. This architecture exploited parallelism on two levels, combining a master/slave model with and all-slaves model. The master/slave model was used to distribute work and over a series of cluster nodes efficiently by having a master process control the workflow, distribute work as well as retrieve results accompanied by a series of slave processes which actually computed the predictions. The all-slaves model on the other hand was implemented to fully utilize all of the available resources by a slave process on a given cluster node. This was accomplished by having multiple

threads compute PIPE predictions in parallel while sharing as much data as possible. This eliminated a significant amount of redundant data and freed up enough memory to allow for multiple threads to compute predictions on more complex organisms. PIPE3 performed well overall, showing a significant speedup due to the multithreaded all-slaves approach used in the slave processes. The master/slave portion of the overall parallel architecture also scaled well, proving to have linear speedup when more slave processes were used.

PIPE3 was then used to produce some novel scientific results. The first major task PIPE3 accomplished was a proteome-wide scan of all possible protein pairs in the *C. Elegans* proteome. After over a week of computation on both the Beowulf and Victoria Falls cluster the scan was complete. The portion of the scan completed on the Victoria Falls cluster also represented the largest threaded calculation ever run at HPCVL, which is an added accomplishment over the biological results. The scan resulted in 32,548 novel predicted interactions. Shortly after this run, a major goal of the overall PIPE project was attempted: a proteome wide scan of all *Homo Sapiens* protein pairs. This task proved to be much more computationally intensive than the previous scan, taking over 3 months of straight computation on the Victoria falls cluster with help from the Beowulf cluster. In the end this scan produced 130,470 novel predicted interactions. These novel predictions, as well as those produce during the *C. Elegans* run, are also supported by other lines of evidence including the cellular location of the predicted interacting proteins, their shared molecular function and biological process.

Moving forward, there are still opportunities for PIPE3 to make an impact. Although the inner workings of the main PIPE algorithm itself remains sequential, the multi-threaded nature of the slave processes of PIPE3 make up for this by computing multiple PIPE predictions in parallel. For the parallelization of the PIPE algorithm itself to be worth investigating it would have to produce a larger speedup than was produced in PIPE3's slave processes. Since it isn't clear if the parallelization of the

PIPE algorithm could produce a significant speedup (or if it is in fact possible), further optimizations of the overall PIPE architecture is not a part of the future plans of the PIPE project. However, there are two major areas with respect to PIPE (and more specifically, the PIPE3 implementation) that will be explored. These are:

- **Applying PIPE3 to organisms in the plant kingdom.** At this point, all versions of the PIPE software have only worked with organisms from the fungi and animal kingdoms, however there are many organisms in the plant kingdom that are of interest to the scientific community. Some of these organisms include arabidopsis, corn, rice, canola and soybean, to name a few. Background research, parameter tuning and testing will have to be done to apply PIPE3 to these new organisms before a scan can take place, and then the significant time investment in the actual PIPE3 run will have to be made. Even though the code is ready to make these predictions, collecting the data and allocating computational resources takes a significant amount of time. In spite of that, this avenue has the potential to produce some very significant results and will most likely be pursued shortly.

- **Building a public database cataloging all of PIPE's predictions.** This database will hold the *S. Cerevisiae*, *C. Elegans*, *Homo Sapiens* data as well as any further protein-protein interaction predictions done by PIPE3 in the future. The hope is that this database will become a central hub for protein-protein interaction prediction data. Since PIPE3 can produce proteome-wide predictions in a relatively short period of time, this database will be continuously growing to serve the scientific community with current, planned and proposed protein-protein interaction predictions. The plan is to make the database available online so anyone can choose their organism of interest, set their desired sensitivity or specificity and then submit the protein pairs they are interested in.

Since there are numerous organisms of interest in the scientific community (from the plant kingdom and others), there is essentially an unending supply of work for PIPE3

to do. These organisms are roughly being ranked by the amount of interest in them and will be run in the near future. Once the public database is in place, these results will be readily available to the scientific community and the potential for more interest in PIPE3's predictions will most likely rise.

Bibliography

[1] P. Aloy and R. B. Russell. Interprets: protein interaction prediction through tertiary structure. *Bioinformatics*, 19(1):161–162, January 2003.

[2] M. R. Arkin and J. A. Wells. Small-molecule inhibitors of protein-protein interactions: progressing towards the dream. *Nat Rev Drug Discov*, 3(4):301–317, April 2004.

[3] J. R. Bock and D. A. Gough. Whole-proteome interaction mining. *Bioinformatics*, 19(1):125–134, 2003.

[4] L. Chen, L.-Y. Y. Wu, Y. Wang, and X.-S. S. Zhang. Inferring protein interactions from experimental data by association probabilistic method. *Proteins*, January 2006.

[5] M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt. A model of evolutionary change in proteins. In M. O. Dayhoff, editor, *Atlas of Protein Sequence and Structure*, pages 345–352+. 1978.

[6] M. Deng, S. Mehta, F. Sun, and T. Chen. Inferring domain-domain interactions from protein-protein interactions. *Genome Res*, 12(10):1540–1548, October 2002.

[7] A. J. Enright, I. Iliopoulos, N. C. Kyrpides, and C. A. Ouzounis. Protein interaction maps for complete genomes based on gene fusion events. *Nature*, 402(6757):86–90, November 1999.

[8] J. Espadaler, O. Romero-Isart, R. M. Jackson, and B. Oliva. Prediction of protein-protein interactions using distant conservation of sequence patterns and structure relationships. *Bioinformatics*, 21(16):3360–3368, August 2005.

[9] S. Fields and O. Song. A novel genetic system to detect protein-protein interactions. *Nature*, 340(6230):245–246, July 1989.

[10] G. Franzot and O. Carugo. Computational approaches to protein-protein interaction. *J Struct Funct Genomics*, 4(4):245–255, 2003.

[11] A.-C. Gingras, R. Aebersold, and B. Raught. Advances in protein complex analysis using mass spectrometry. *The Journal of Physiology*, 563(1):11–21, February 2005.

[12] D. S. Han, H. S. Kim, W. H. Jang, S. D. Lee, and J. K. Suh. PreSPI: a domain combination based prediction system for protein-protein interaction. *Nucleic Acids Res*, 32(21):6312–6320, 2004.

[13] J.-D. D. Han, D. Dupuy, N. Bertin, M. E. Cusick, and M. Vidal. Effect of sampling on topology predictions of protein-protein interaction networks. *Nature biotechnology*, 23(7):839–844, July 2005.

[14] H. Huang and J. S. Bader. Precision and recall estimates for two-hybrid screens. *Bioinformatics (Oxford, England)*, 25(3):372–378, February 2009.

[15] T. Ito, T. Chiba, R. Ozawa, M. Yoshida, M. Hattori, and Y. Sakaki. A comprehensive two-hybrid analysis to explore the yeast protein interactome. *Proceedings of the National Academy of Sciences of the United States of America*, 98(8):4569–4574, April 2001.

[16] W. Kyu, K. J. Park, and J. K. Suh. Large scale statistical prediction of protein-protein interaction by potentially interacting domain (pid) pair. In *In Genome Inform Ser Workshop Genome Inform*, pages 42–50, 2002.

[17] E. D. Levy and J. B. Pereira-Leal. Evolution and dynamics of protein interactions and networks. *Current Opinion in Structural Biology*, 18(3):349 – 357, 2008. Nucleic acids / Sequences and topology.

[18] E. M. Marcotte, M. Pellegrini, H.-L. Ng, D. W. Rice, T. O. Yeates, and D. Eisenberg. Detecting protein function and protein-protein interactions from genome sequences. *Science*, 285(5428):751–753, July 1999.

[19] E. M. Marcotte, M. Pellegrini, M. J. Thompson, T. O. Yeates, and D. Eisenberg. A combined algorithm for genome-wide prediction of protein function. *Nature*, 402(6757):83–86, November 1999.

[20] S. Martin, D. Roe, and J. L. Faulon. Predicting protein-protein interactions using signature products. *Bioinformatics*, 21(2):218–226, January 2005.

[21] A. R. Mendelsohn and R. Brent. Protein Biochemistry: Protein Interaction Methods-Toward an Endgame. *Science*, 284(5422):1948–1950, 1999.

[22] R. Mrowka, A. Patzak, and H. Herzel. Is there a bias in proteome research? *Genome Res*, 11(12):1971–1973, Dec 2001.

[23] M. A. N. Chepelev, L. Chepelev and A. Golshani. Large-scale protein-protein interaction detection approaches: Past, present and future. *Biotechnology & Biotechnological Equipment*, 22(1):513–529, February 2008.

[24] U. Ogmen, O. Keskin, A. S. Aytuna, R. Nussinov, and A. Gursoy. PRISM: protein interactions by structural matching. *Nucleic Acids Res*, 33(Web Server issue), July 2005.

[25] R. Overbeek, M. Fonstein, M. D'Souza, G. D. Pusch, and N. Maltsev. The use of gene clusters to infer functional coupling. *PNAS*, 96(6):2896–2901, 1999.

[26] A. Pandey and M. Mann. Proteomics to study genes and genomes. *Nature*, 405(6788):837–846, June 2000.

[27] F. Pazos and A. Valencia. Similarity of phylogenetic trees as indicator of protein-protein interaction. *Protein Eng*, 14(9):609–614, September 2001.

[28] M. Pellegrini, E. M. Marcotte, M. J. Thompson, D. Eisenberg, and T. O. Yeates. Assigning protein functions by comparative genome analysis: protein phylogenetic profiles. *Proc Natl Acad Sci U S A*, 96(8):4285–4288, April 1999.

[29] J. M. Peltier, S. Askovic, R. R. Becklin, C. L. Chepanoske, Y.-S. J. Ho, V. Kery, S. Lai, T. Mujtaba, M. Pyne, P. B. Robbins, M. von Rechenberg, B. Richardson, J. Savage, P. Sheffield, S. Thompson, L. Weir, K. Widjaja, N. Xu, Y. Zhen, and J. J. Boniface. An integrated strategy for the discovery of drug targets by the analysis of protein-protein interactions. *International Journal of Mass Spectrometry*, 238(2):119 – 130, 2004. Drug Discovery.

[30] S. Pitre, M. Alamgir, J. Green, M. Dumontier, F. Dehne, and A. Golshani. Computational methods for predicting protein-protein interactions. *Refereed Survey Article, Advances in Biochemical Engineering/Biotechnology, H. Seitz (Ed.)*, Springer-Verlag, 2008.

[31] S. Pitre, F. Dehne, A. Chan, J. Cheetham, A. Duong, A. Emili, M. Gebbia, J. Greenblatt, M. Jessulat, N. Krogan, X. Luo, and A. Golshani. Pipe: a protein-protein interaction prediction engine based on the re-occurring short polypeptide sequences between known interacting protein pairs. *BMC Bioinformatics*, 7:365+, July 2006.

[32] S. Pitre, C. North, M. Alamgir, M. Jessulat, A. Chan, X. Luo, J. R. Green, M. Dumontier, F. Dehne, and A. Golshani. Global investigation of protein-protein interactions in yeast saccharomyces cerevisiae using re-occurring short polypeptide sequences. *Nucl. Acids Res.*, pages gkn390+, June 2008.

[33] S. Roy, D. Martinez, H. Platero, T. Lane, and M. Werner-Washburne. Exploiting amino acid composition for predicting protein-protein interactions. *PLoS ONE*, 4(11):e7813, 11 2009.

[34] D. P. Ryan and J. M. Matthews. Protein-protein interactions in human disease. *Current Opinion in Structural Biology*, 15(4):441 – 446, 2005. Membranes/Engineering and design.

[35] R. Sharan, I. Ulitsky, and R. Shamir. Network-based prediction of protein function. *Molecular systems biology*, 3, March 2007.

[36] L. Skrabanek, H. K. Saini, G. D. Bader, and A. J. Enright. Computational prediction of protein-protein interactions. *Molecular biotechnology*, 38(1):1–17, January 2008.

[37] E. Sprinzak and H. Margalit. Correlated sequence-signatures as markers of protein-protein interaction. *J Mol Biol*, 311(4):681–692, August 2001.

[38] A. Stein, R. B. Russell, and P. Aloy. 3did: interacting protein domains of known three-dimensional structure. *Nucleic Acids Res*, 33(Database issue), January 2005.

[39] P. Uetz, L. Giot, G. Cagney, T. A. Mansfield, R. S. Judson, J. R. Knight, D. Lockshon, V. Narayan, M. Srinivasan, P. Pochart, A. Qureshi-Emili, Y. Li, B. Godwin, D. Conover, T. Kalbfleisch, G. Vijayadamodar, M. Yang, M. Johnston, S. Fields, and J. M. Rothberg. A comprehensive analysis of protein-protein interactions in saccharomyces cerevisiae. *Nature*, 403(6770):623–627, February 2000.

[40] J. P. Vert. A tree kernel to analyse phylogenetic profiles. *Bioinformatics*, 18 Suppl 1, 2002.