MULTIPARTITE GRAPH CLUSTERING FOR STRUCTURED DATASETS AND AUTOMATING ORTHOLOG EXTRACTION

BY AKSHAY VASHIST

A dissertation submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Computer Science

Written under the direction of

Casimir A. Kulikowski and Ilya B. Muchnik

and approved by

_____

_____

_____

_____

_____

New Brunswick, New Jersey

January, 2006

© 2006

Akshay Vashist

ALL RIGHTS RESERVED

ABSTRACT OF THE DISSERTATION

Multipartite Graph Clustering for Structured Datasets and Automating Ortholog Extraction

by Akshay Vashist

Dissertation Director: Casimir A. Kulikowski and Ilya B. Muchnik

The problem of clustering a structured dataset, where besides pair-wise similarities between objects, there is additional prior information about partitioning of objects into predefined classes, is important in many fields, yet has been inadequately studied. A natural and powerful representation for this problem is a multipartite graph, where the vertices represent the objects and the partite sets the predefined classes, enabling work with multiple levels of relationships simultaneously. The clustering problem itself is formulated as the iterative finding of multipartite quasi-cliques.

We have used the quasi-concave set functions for modeling clusters. This model is realized by associating any multipartite subset with a score that assesses the homogeneity among elements in that subset. Using the degree of membership of the least similar element in the subset as the score, a cluster is defined as the subset with the highest score. Properties of this score function allow the global solution of this optimization problem to be found efficiently. Further improvements in speed have been achieved by careful implementation choices, making it suitable for use with very large datasets.

The method has been tested and evaluated on gene function annotation and ortholog clustering problems in comparative genomics: given a set of genetic sequences along with the information about the species they belong to, the objective is to find evolutionarily related genes performing similar functions across species. Our clustering results are highly consistent with the expertly curated ortholog clusters on a set of 43 genomes. The method was also applied for functionally annotating Rice proteins based on ortholog clusters from cereal genomes and yielded highly accurate annotations based on gene functional and structural similarity. As demonstration of the broader applicability of the method, it was applied to two problems in computer vision with encouraging results.

Acknowledgements

I thank my graduate advisors Dr. Casimir Kulikowski and Dr. Ilya Muchnik for their intellectual guidance, support, constant encouragement, and patience. My discussions with them have been instrumental in shaping my thinking and chiseling me as an individual. I am grateful to them for all their efforts and advice.

I thank Dr. Joachim Messing for introducing me to the ortholog detection problem. I would like to thank my collaborators who introduced me to computer vision. It was indeed pleasure working with Zhipeng Zhao and Dr. Ahmed Elgammal on the discriminative patch selection for part based on object recognition. I also benefited from working with Rong Zhang and Dr. Dmitris Metaxas on the gait recognition project. I thank all those with who I had the opportunity to discuss and have feedbacks on this work, in particular I would like to thank Dr. Shamil Sunyaev. I thank Dr. Vladimir Pavlovic for his comments on the thesis.

I would like to thank the Department of Computer Science, DIMACS and the Waksman Institute for financially supporting projects that shaped my dissertation work. The workshops organized by DIMACS provided excellent exposure to computational biology by allowing me to interact with pioneers in the field. The LCSR provided an excellent computational infrastructure, without which this work wouldn't have been possible.

My discussions with Suhrid Balakrishnan and Vasisht Reddy Tadigotla were a constant source of support and encouragement, both intellectually and philosophically. I thank them for their friendship. I also thank Buddhaditya Deb, Evangelia Chnari, Samrat Ganguly, Navin Goyal, Nikita Lytkin, Jiankuan Ye, and Hwaseob Joesph Yun.

I thank my parents and family members for their love and support. My thanks also go to my wife, Shilpa, for her steady encouragement and unconditional support in all aspects.

Dedication

To my late grandmother

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

One of the central problems in data analysis is to find groups of objects sharing a certain pattern of features. Identification of such groups of objects that can be considered similar with respect to features highlights data's possible categorizations and makes it observable by focusing on representative examples. This process also reduces the dimensionality and helps visualize the data since objects inside a group share many common attributes. Finding such naturally existing groups in data is primarily the subject of cluster analysis.

A convenient way to represent pair-wise relationships between objects is a graph derived from the data. A graph is a collection of nodes and edges, where the nodes represent objects and the edges connecting pairs of vertices represent the pair-wise relationships between the objects. A graph provides an intuitive representation for a clustering problem with the added benefit that graph theoretic concepts can readily be used to formulate the criteria for clustering, and existing graph algorithms can be adapted to find clusters. This has made clustering based on similarity graph representations a popular method for finding groups in the data in biology, computer vision, ecology etc.

Most clustering methods, including graph theoretic methods, consider the problem where the input data consists of a set of individual objects with comparable data describing each, and the only relationship defined between them is that of pair-wise similarity. Edges between pairs of vertices in a graph can then be assigned weights to reflect the strength of similarity. In practice, however, there is often additional information about the objects beyond the pair-wise similarity graph. For instance, consider the

problem of clustering peoples faces in a sequences of frames captured by a surveillance camera. Assuming that faces have been segmented, one could perform clustering using the graph of pair-wise similarity between all the extracted facial data. Depending on how this data is obtained, however, there may be additional information, say regarding the organization of faces in the image, which may be critical to consider while clustering faces across the sequence of images. As data analysis advances, considering such inter-object relationships which are independent of the pair-wise similarities, may become increasingly important in clustering objects. Vertices can be described by vectors of the object characteristics, which may reveal an underlying structure of the objects which is not fully captured in a particular set of similarity measures. Existing clustering methods do not incorporate this additional information, and this "gap" in current clustering methods was our main motivation to develop a new type of clustering approach which can simultaneously take into account pair-wise object similarities as well as their inter-relationships derived from additional data sources. This thesis focuses on formulating and efficiently solving this problem and is demonstrated with specific examples.

1.2 Data analysis for given prior partition model

Structure in the input data can be characterized by a partition model of the given collection of objects. The classes defined by a specific partition model may be organized into a hierarchy (or other complex organizations) which may suggest whether certain pair-wise similarities should be emphasized more than others. A natural framework for representing the structural data, which is a-priori divided into classes, is a multipartite graph. In this graph, the objects in the input are represented by the vertices and any *a priori* classes of objects correspond to the partite sets in a multipartite graph. Such a representation is well-suited for problems where *non-overlapping classes* of objects are specified as input and one needs to consider relationships only across different classes while ignoring relationships between objects within the same class. In other words, we represent the input data by two graphs: besides the similarity graph between the objects, there is also a graph associated with relationships between the classes of objects. Both graphs can be weighted graphs. The weights on the edges of the first graphs can

represent the strength of pair-wise similarity between objects, while the weights on the graph between the given classes of objects represent a similarity relationship.

The problem for the input data represented by a multipartite graph can be formulated as finding a dense multipartite subset - a subset containing similar elements from at least two classes of objects. We call such a dense subset a quasi-clique (or a cluster) and formulate the problem of finding it as a combinatorial optimization problem. We also present and analyze the algorithm for finding the solution for this problem.

The main idea is to formulate the clustering problem as an optimization problem. In order to do this, we needed to solve several general methodological problems: (i) how to measure similarity between a pair of objects taking into account their complex inter-relation structure (ii) how to measure similarity between an object and a subset of objects (iii) how to measure the density of a subset of objects. Solutions of these problems will allow us to find a subset having maximum density, which is not trivial for several reasons. The usual definitions of quasi-cliques [66, 3] satisfy the monotone property, so the corresponding problem becomes trivial, yet, if the definition does not satisfy the monotone property, the problem becomes intractable. So, the problem is to find a non-trivial density function which is relevant to problems in practice and has properties which guarantee efficient procedures for solving the optimization problem. Based on prior work by [55, 54], we sought the solution in the form of quasi-concave function optimization. Using a particular rescaling procedure we were able to integrate the similarities for the two graphs between objects and between partite sets and achieve the desired clustering. Also, we were able to find efficient data structures for implementing the algorithm that allowed us to work with hundreds of thousands of objects.

We proposed a general type of objective function which satisfies the above properties (see chapter 2); and have analyzed a very specific type of function to make it relevant to the biological requirements to find clusters from different organisms.

The clustering method we have developed has many applications. One of the applications related to genomics was an important independent goal of this dissertation. We adapted our general method specifically to this problem and used the results to

demonstrate the power of the method. The second application, in image processing, was implemented as a proof of generalizability.

1.3 Comparative Genomics

Comparative genomics deals with studying and inferring the commonalities and differences between organisms based on comparisons of their nucleic acid and amino acid sequences. Such comparisons provide insights into similarities between organisms at a molecular level and also enable us to accurately estimate the evolutionary relationships between organisms. A fundamental problem in comparative genomics is the identification of genes from different organisms that are involved in similar biological functions. This requires identification of homologous (evolutionarily related) genes that have evolved through vertical descent from a single ancestral gene in the last common ancestor [29]. Such genes are called orthologs.

As genes are responsible for carrying out various processes in a cell, the problem of finding orthologous genes is central to understanding the organisms at a molecular level. Such genes, usually, carry out similar molecular functions (function for a gene broadly means whatever the gene does) across organisms making their clustering an important problem. Furthermore, they are also highly conserved compared to other genetic elements whose conservation is subtle. Usually, these other genetic elements occur in the vicinity of functional genes, so orthologs are used as anchors to target the detection of the relatively less conserved genetic elements. Other uses of orthologs include detecting and measuring the effects of selection, mutations, estimating their rates, and inferring reliable phylogenetic relationships between the organisms.

1.3.1 Ortholog detection using multipartite graph clustering

The input for the ortholog clustering problem is a set of genetic sequences along with information about the organisms they belong to. The objective is to find evolutionarily related sequences, and this translates into finding similar sequences from different organisms. The ortholog detection problem is complicated due to the presence of another

type of very similar sequences in the same organism. These sequences are results of duplication events (a common evolutionary event) which produce two copies of the same gene. Although both types of genes are similar, only orthologs are likely to be involved in the same biological role. Due to this, for detecting orthologs it is critical to focus on similarities between genes from different organisms while ignoring the similarities between genes within an organism.

The requirement of selectively ignoring gene similarities for paralogous sequences can be conveniently represented in a multipartite graph, where the organisms correspond to the a-priori given partite sets and the genes correspond to the nodes in the partite sets. Another specific problem in finding ortholog clusters is that the observed similarities between genes from different pairs of organisms cannot be considered at par. It is expected that orthologous genes from closely related organisms will be much more similar than those from distantly related organisms. Fortunately, we often have estimates of evolutionary relationships between the organisms and these define a hierarchical graph over the partite sets. Using these evolutionary relationships between organisms, it is possible to correct the observed gene similarities by scaling up the similarities between the orthologs from the distantly related organisms. This rescaling of observed similarities will make the similarities between orthologs from distant organisms closer to the similarities between orthologs from closely related organisms, so that a computational method based on quantitative values of pair-wise similarities will produce more reliable ortholog clusters. Making such corrections to observed gene similarities is possible within the multipartite graph framework and can improve the ortholog clustering results.

A biological description of the orthologs, and their importance for various purposes in described in Appendix B. A survey of methods for ortholog detection and an analysis of these methods is also present in this appendix.

1.3.2 Ortholog clusters for gene function annotation

The most important practical use of orthologs is for functional annotation of genes in a newly sequenced genome. The sequencing of genomes is prioritized by studying the

representative and simple organisms first; their genomes are usually comprehensively studied and their genes are assigned functions using a combination of state-of-the-art automatic methods and manual curation. This repertoire of well-studied and annotated genomes is later used for functionally annotating genes in the newly sequenced genomes by transferring the annotations from the orthologs in the model organism to those in the new genome.

For functionally annotating genes in a newly sequenced genome, the best source of annotation is the well-studied ortholog groups from closely related organisms (because, as indicated earlier orthologs from closely related species are likely to have the most similar functions). However, in practice, most of the closely related genomes to a target genome are likely to be incomplete genomes. Ortholog clusters from such genomes could be extracted de novo but finding candidate orthologs in partially complete genomes is an inadequately addressed problem. The multipartite graph clustering method is well suited for finding orthologs in incomplete genomes. Its application to annotate sequences from the rice genome using ortholog clusters extracted from cereal (incomplete) genomes is shown.

1.4 Computer Vision[1]

In many target recognition problems, the target objects are represented as parts of a sequence or collection of images. Informally, these smaller images are the smallest unit of information used in the task, but depending on the problem, the target object may either be contained in these smaller images or be defined by a sequence of images. For instance, in the gait recognition problem where the goal is to recognize a subject from a given sequence of images taken during a gait cycle, the target has to determined from the sequences of images. In other object recognition problems we may be given a single image and the goal is to determine if it contains an instance of the target object. In the second problem, the target object is defined by a collection of image-segments contained

[1]The problems presented in this section were solved collaboratively: the Gait recognition problem with Rong Zhang and Dmitris Metaxas [93] and discriminative patch selection with Zhipeng Zhao and Ahmed Elgammal [88].

in the image. In either case, it is important to note that not all observations (gait frames or image segments) are equally useful for the recognition task. Ubiquitously present objects are usually unhelpful, or may even hinder by creating "confusion" in the process of classification. So, it is important to find observations that uniquely represent the object. Such problems can also be solved by formulating them as multipartite graph clustering for which the framework proposed in this thesis is applicable.

1.4.1 Gait Recognition

In human subject identification using gait analysis, an individual's gait is captured by cameras as a sequence of image frames. The goal is to construct a supervised classifier which can be trained using gait sequences of an individual to later identify the subject using just the gait data. Although, the entire sequence of gait frames is important, in many cases, certain "most informative" stages in the gait cycle can uniquely characterize the subject. So, for constructing a supervised classifier, the gait frames characteristic of the subject are more informative [93]. Such informative frames can be identified by comparing the target subject's gait frames to those of others. However, this may lead to the extraction of noisy frames which may not really be representative of the target subject. An alternative approach is to exclude the gait frames from a subject which are similar to those from many other subjects. Such exclusions will retain gait frames specific to the subject including those that are characteristic of subject's gait style.

The problem of finding the gait frames that characterize a subject is solved as an inverse problem where we identify clusters of similar frames from multiple subjects. Frames in such clusters are uninformative for constructing a supervised classifier and are therefore removed from training of the classifier. The uninformative frames are identified as quasi-cliques in a multipartite graph where human subjects are the partite sets and gait frames from the subject correspond to vertices in a partite set [93].

1.4.2 Discriminative Patch selection

A central problem in classification is to determine the best subset of input information which yields the highest accuracy. Usually, we have additional information which helps

us to identify this subset of information. The patch based object recognition, where an image is represented as collection of patches, the goal is to recognize the target object using a subset of extracted patches. The patch extraction process is oblivious to the target object and is carried out in isolation for individual images using general purpose approaches. Consequently, the patches represent an image characteristic, where these characteristics correlate well with the desired target object and are likely to be located on it. However, since the target is not directly captured by the criteria, several other patches not representing the object are also likely to be extracted. While constructing a supervised classifier that uses the collection of patches representation from the image, it is wise to filter out the patches that either represent the background in the image or that are not characteristic of the target object.

Identifying patches that characterize a target object can be done using our multipartite graph clustering method. We use a collection of images that contain the target object along with another collection of images that do not contain the target. We assume that saliencies of the target object are well captured in the patches coming from those images and also form a good representation for the target object. We want to exclude patches that can also be present in the images not containing the target object. So, the goal is to find clusters of patches that are present in most of the images containing the target object and at the same time are very different from the images not containing the target object. Furthermore, while finding the discriminative image patches that characterize the target object, it is best to focus on similarities between patches across different instances of the target object, rather than similarities between patches from an image (although they may be very similar). These two informal requirements can be conveniently expressed in the multipartite graph representation of similarities between patches [88].

We construct a weighted multipartite graph using the two classes of image: the images containing the target object and those that do not contain it. The patches corresponding to images not containing the target object form a single class and we are interested in extracting patches that are dissimilar to patches in this negative class.

Each image containing the target object forms a class; this class contains vertices corresponding to patches from this image. As we construct the multipartite graph we ignore the similarity between patches contained within a class. While extracting the discriminative subsets of patches we focus on subsets of patches that are similar to patches from images containing the target object, and distant from patches in the negative class. We have applied this method for patch-based object recognition to a database of benchmark images and obtained results that are competitive with those from the current methods and in most cases show improvement recognition over existing methods [88].

An extended description of the two problems including the details of problem formulation and brief results is presented in Appendix C.

1.5 Organization of the thesis

In Chapter 2, we formulate multipartite graph representations for depicting relationships in data with a given partition of objects and with an emphasis on analyzing relationships across the given partite sets. We also give a precise description of cluster extraction for this model as a combinatorial optimization problem. The clustering formulation and the methodology has been developed precisely for the genomics application. This clustering problem is solved using an efficient polynomial time procedure for which we describe several implementation choices using data structures that lead to significant performance gains in practice.

Chapter 3 generalizes and studies the properties of the functions used for defining the multipartite clusters in chapter 2. We study a general class of functions called the quasiconcave functions and quasiconvex functions. Interestingly, under a relatively mild condition, they can both be optimized simultaneously, yielding an efficient algorithm for both of them.

The formulation of the ortholog clustering problem in the multipartite graph clustering framework is described in chapter 4. In this chapter we also present various criteria for validating the results obtained from the proposed multipartite graph clustering method. These include validating ortholog clustering using independent sources

and their comparison to existing ortholog clusters based on statistical tests and a set of indices that we have devised.

In chapter 5, we describe two data sets for ortholog clustering: one containing 43 complete genomes with known ortholog clusters and the other containing mostly the partially completed genomes from the cereals. We describe the rationale for the choice of these datasets along with the choices for calculating the similarities between sequences and the species tree for calculating the distance between the genomes. We also describe the additional independent data used for validating the ortholog clusters. These sources - the Pfam and the SCOP databases, are used for functionally and structurally annotating sequences, respectively.

The results of ortholog clustering on the two data sets are presented in chapter 6. On the first data set, the results show a high consistency with the existing ortholog clusters. We show that using the phylogenetic tree (i.e., considering observed similarities in the context of the relationships between the partite sets) improves the results as demonstrated by a higher correlation with existing manually curated results. We also present a brief qualitative analysis of the results from a biological perspective. The ortholog clustering results on the second data are validated using independent data, thus showing their consistency. Here again, we perform a qualitative biological analysis of some representative clusters.

Chapter 7 discusses the contributions of the thesis and future directions. The thesis also contains 3 appendices containing supplemental information.

Chapter 2

Multipartite graph clustering using element-to-set similarity

2.1 Introduction

In chapter 1 and appendix B, we reviewed methods for finding clusters of orthologous genes, which are similar genes from different organisms. Abstractly, the aim of the ortholog clustering problem is to find similar objects (genes) where objects are naturally partitioned into different subsets (genomes), and one wishes to consider only similarity between objects across subsets. A natural framework for representing this problem is a multipartite graph where the objects are represented by vertices and the *a priori* classes of objects correspond to the partite sets in a multipartite graph. In general, the multipartite graph representation is well-suited to problems where *non-overlapping classes* of objects are specified as input and one needs to only consider relationships across different classes while ignoring relationships between objects within the same class. For the ortholog finding problem, since the information about the genome to which a gene belongs is given, the relationship between genomes (i.e., their evolutionary relations) can be incorporated into the ortholog analysis. In other words, apart from the graph between objects, we also have a graph associated with relationships between the partite sets. In the analysis below we represent the relationships between partite sets by a complete weighted graph. In the application for inter-genome analysis, the weights are distances between genomes, defined by using the evolution tree.

In this chapter, the multipartite graph framework, with the associated complete weighted graph of partite sets, is presented in the context of graph clustering for finding ortholog clusters. The problem of finding a dense multipartite subset (or, a multipartite cluster) is formulated as a combinatorial optimization problem. Algorithms for finding

the exact global solution for such an optimization problem are described. This is followed by an analysis of these algorithms along with data structures that enable their efficient implementation. A generalization of the methods developed in this chapter is presented in chapter 3 where the abstract properties of the criterion proposed here are studied and described in the context of more general multipartite graph clustering.

2.2 Multipartite Graph Framework

A graph $G = (V, E)$ is a set of vertices V and a set of edges $E \subseteq V \times V$ connecting these vertices. The edges describe relationships between objects represented by the vertices. A graph in which weights are associated with the edges is a weighted graph $G = (V, E, W)$ where $W : E \rightarrow \mathbb{R}^1$. In this chapter, the relationship denoted by the weight on a edge is the similarity or dissimilarity (distance) between the objects connected by the edge.

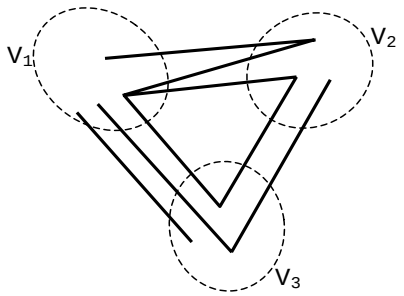

Figure 2.1: A tri-paritite graph with V_1,V_2,V_3 as three partite sets.

A multipartite graph with k partite sets is denoted as $G^k(V, E)$, where $V = \cup_{\ell=1}^{k} V_\ell$ such that $V_i \cap V_j = \emptyset$ for all $i \neq j$ and each V_ℓ is a pre-specified partite set of vertices in V. The edges can be present only between vertices from different partite sets in V, i.e, $E \subseteq \cup_{\ell \neq m} V_\ell \times V_m,\ \ell, m \in \{1, 2, \ldots, k\}$. A weighted multipartite graph is represented as $G^k(V, E, W)$ where w_{ij} is the weight associated with the edge $e_{ij} \in E$ connecting the vertices i and j from different partite sets. An instance of a multipartite graph with three partite sets is shown in Fig. 2.1 . In addition to the relationship between objects, there are relationships between the partite sets represented by the complete weighted graph $G(U, P)$ between the partite sets of G^k. Each vertex in the

set $U = \{V_1, V_2, \ldots, V_k\}$ is associated with a partite set in the graph G^k, and $P = ||\rho_{ab}||$ is a matrix of non-negative weights $(a, b \in U)$.

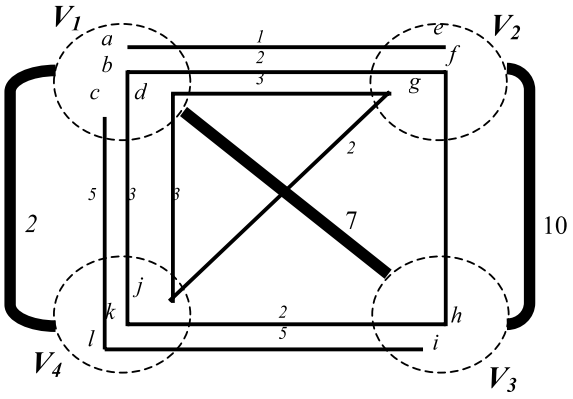

Figure 2.2: A multipartite graph with levels of relationships. The relationships between the vertices (represented by lower case alphabets inside the dotted ellipses) are represented by the thin edges constitute multipartite graph, G^4, with four partite sets V_1 through V_4. The other graph, G, represented by the thick edges, expresses relationships between the partite sets 1 through 4.

P=

	V_1	V_2	V_3	V_4
V_1	0	0	7	2
V_2	0	0	10	0
V_3	7	10	0	0
V_4	2	0	0	0

W=

	a	b	c	d	e	f	g	h	i	j	k	l
a					1	0	0	0	0	0	0	0
b					0	2	0	0	0	0	3	0
c					0	0	0	0	0	0	0	5
d					0	0	3	0	0	3	0	0
e	1	0	0	0				0	0	0	0	0
f	0	2	0	0				3	0	0	0	0
g	0	0	0	3				0	0	2	0	0
h	0	0	0	0	0	3	0			0	3	0
i	0	0	0	0	0	0	0			0	0	5
j	0	0	0	3	0	0	2	0	0			
k	0	3	0	1	0	0	0	2	0			
l	0	0	5	0	0	0	0	0	5			

Table 2.1: Adjacency matrices for the multipartite graph shown in Fig. 2.2. The adjacency matrix **P** represents the relationships between the partite sets while the matrix **W** represents the relationships between vertices across partite sets. Clearly, the two adjacency matrices are beyond the representation capabilities of a simple graph, or a single adjacency matrix.

So, a multipartite graph enables us to represent two graphs simultaneously - (a) the graph representing relationships between elements where edges are between vertices from different paritite sets and (b) the graph representing relationships between different partite sets where edges relate the partite sets themselves (see Fig. 2.2). This type of

data associated with two matrices, represented here by a weighted multipartite graph G^k and the associated complete weighted graph G will be considered as an input for our clustering method. However, before describing our method, we present a survey of methods relevant to clustering on a multipartite graph.

2.3 Clustering related problems on a multipartite graph

For a simple graph, the process of clustering strives to detect "naturally" occurring structures such as those corresponding to highly similar objects or a dense subgraph. On the other hand, for a multipartite graph special considerations may arise - in addition to ignoring similarities between objects within a partite set and focusing on similarities across partite sets, there can be other requirements such as objects from some partite sets being preferentially clustered together depending on the relationship between the partite sets. To the best of our knowledge, there are no studies that simultaneously consider the two types of relationships (the two graphs G^k and G) for clustering. So, we review some clustering methods for multipartite graphs based on relationships expressed by the graph G^k.

Consider the simplest case of a multipartite graph - the bipartite graph $G^2 = (V_1 \cup V_2, E)$, where V_1 and V_2 are the two partite sets of the vertices in G^2 and $E \subseteq V_1 \times V_2$ is the set of undirected edges. A pair of two disjoint subsets $A \subseteq V_1$ and $B \subseteq V_2$ is called a biclique if for all $a \in A$ and $b \in B$ the edge $(a, b) \in E$. Although bicliques are dense subgraphs and maximal biclique(s) can be interpreted as clusters (non-extensible dense subgraphs), it turns out that under most definitions of biclique, finding a maximal biclique is an intractable problem. For a bipartite graph, the *maximum vertex biclique problem* in which the criteria to be maximized is $|A|+|B|$ such that $|A|+|B| \geq k$, where k is positive integer, is solvable in polynomial time via the matching algorithm [65] [1]. The decision problem for *maximum edge biclique*, where the criterion to be maximized

[1] Another polynomial time solution can be obtained by transforming the given instance of a problem to an integer program which has three variables per inequality such that one of those variables appears in exactly one of the inequalities [38]; since the constraint matrix for such an integer program would be totally unimodular, the linear programming relaxation finds an optimal solution, thereby providing a polynomial solution.

is $|A| * |B|$ such that $|A| * |B| \geq k$ is NP-complete for a bipartite graph [65].

There are two definitions of cliques in a multipartite graph, although these are usually discussed only in the context of bipartite graphs. The maximum edge clique, mentioned above, and the so called maximum vertex clique. The latter is supported by polynomial procedures [38]. In [20] it has been shown that considering how a clique is defined does not alter the complexity of these problems, and the weighted versions are as hard as the unweighted ones. The clique definition for a multipartite graph, $G^k = (V, E)$, is analogous - a collection of subsets of vertices $V'_\ell \subseteq V_\ell \; \ell \in < 1, 2, \ldots, k >$ is said to be a multipartite clique if $\cup_{\ell \neq m} V'_\ell \times V'_m \subseteq E \; ; \; \ell, m \in < 1, 2, \ldots, k >$. Clearly, there is an intentional ambiguity in the definition about how many non-empty multipartite subsets should a multipartite clique contain. This freedom leads to two different versions of multipartite cliques - one requiring the multipartite clique to contain exactly k partite sets and a relaxed version where a multipartite clique contains vertices from at least two nonempty partite sets. Both problems and their weighted variants are NP-complete [20]. In [20], the authors studied a restricted class of a multipartite graphs where partitions are labeled as layers and edges exist only between adjacent layers, then a vertex may be connected to vertices from two other partitions. It is clear that their class of multipartite graphs is contained in the multipartite graphs defined here, and therefore problems on multipartite graphs, as defined here, are at least as hard.

2.3.1 Clustering formulated as quasi-cliques

In principle, the clique paradigm is the simplest model for real world applications which should allow some incompleteness in subgraphs extracted as clusters. From an application perspective the requirement for finding clusters is not to find exact cliques, rather, one expects to find groups of closely related objects that form dense subgraphs. Extraction of such dense subgraphs is more robust to noise in the input data and often better models the domain knowledge. This is the reason that most applications relax the clique finding requirement and focus on clusters that can more appropriately be called *quasi-cliques*. This raises the need to extend the model to the more appropriate model of a quasi-clique.

The relaxation from cliques to quasi-cliques usually involves a threshold parameter, γ where $0 < \gamma \leq 1$. Depending on whether this threshold is used for imposing a condition on the degree of the nodes in a quasi-clique, or a condition on the number of edges in the quasi-clique, two popular variants appear in the literature. The first variant requires that each node in the quasi-clique, H, has a degree above a predefined threshold γ, i.e, every node in H is connected to at least $\gamma \cdot |H|$ nodes in H [66]. The second variant imposes the condition on the number of edges in the quasi-clique H by requiring that the number of edges in H is at least $\gamma \frac{|H|(|H|-1)}{2}$ [3]. It is clear that for the same value of the threshold γ, the first variant will produce more homogeneous quasi-cliques. The first relaxation has been studied along with methods for finding such quasi-cliques in [66] while a comprehensive treatment of quasi-cliques using the second definition was recently carried out by Abello et al. [3]. It must be emphasized that quasi-cliques like those in [66, 3] are usually defined in the context of simple graphs, although [3] also defined quasi-cliques for a bipartite graph. A quasi-clique for a bipartite graph $G^2 = (V_1 \cup V_2, E)$ is defined as the bipartite subgraph induced on the set of vertices H (where $H = H_1 \cup H_2$ and $H_1 \neq \emptyset$, $H_2 \neq \emptyset$) such that H contains at least $\gamma |H_1| \cdot |H_2|$ edges. It is clear to see how the definition for the bipartite quasi-cliques can be modified to the case of multipartite quasi-cliques.

So, in practice, quasi-cliques are formulated as relaxations of the definition of clique and are implemented by heuristic procedures. Such definitions have at least one main disadvantage: extraction of a subgraph in a weighted graph becomes critically dependent on the predefined threshold. It must be emphasized that in these methods the concept of quasi-clique is analyzed from a general graph clustering perspective and does not consider the specifics of the multipartite structure. Below we describe a new type of "context dependent quasi-clique" where *one does not need to predefine any thresholds.* The thresholds are defined automatically according to the properties (structure) of the instance of graph analyzed.

2.4 Finding a cluster in a multipartite graph

One of the fundamental challenges in clustering is modeling the intuitive notion of clusters. The first step in implementing such model is to choose an appropriate definition of the quasi-cliques. To do this, we propose to define clusters as solutions to an optimization problem by introducing an objective function over subsets of the vertices of a given multipartite graph. So, to find a weighted multipartite quasi-clique as a cluster, we need to define a score for any arbitrary multipartite subset[2], H of V. This score will formalize the notion of a quasi-clique as the subset, H^*, with the highest score.

While the quasi-clique model described above will find one cluster, the final clustering output desired is a set of clusters. This can be achieved by removing the clusters already found, and iteratively finding a cluster as the quasi-clique in the remaining data. This iterative clustering paradigm is different from traditional clustering methods such as K-means [27], hierarchical clustering [27], etc. where all the clusters are identified simultaneously. In this paradigm clusters are extracted until the input data is exhausted or no more clusters can be found (leaving behind the singletons). So, this has the advantage that we do not need to specify the number of clusters *a priori*. The number of clusters is automatically determined depending on the structure of relationships in the graph.

As we mentioned earlier, we want to incorporate the contributions from the two different levels of weighted relationships. Recall that we are given $\mathbf{W}$, the matrix of pair-wise weighted relationships between vertices and the matrix $\mathbf{P}$ expressing pair-wise weighted relationships between the partite sets. So, the score for any subset H must be defined using these two different relationships.

The informal requirements for clustering are: a low intra-cluster variance (elements in a cluster must be very similar) while also yielding a high inter-cluster variance (elements from different clusters must be dissimilar, or elements in a cluster must be

[2]Here the term arbitrary "multipartite subset" means that the subset includes vertices from at least two partite sets

dissimilar from those outside the cluster) [27]. Moreover, due to the multipartite nature of the problem, an implicit requirement is that clusters must contain elements from multiple (at least two) partite sets. Further, *we require that elements from distantly related partite sets should be given a higher chance of being in clusters even if their similarities are weaker*. The underlying assumption is that objects (vertices) from distantly separated classes (partite sets) are closer than the numerical value of the similarity observed between them.

2.4.1 Criterion for clustering

Our problem is to design a score addressing the above requirement using the two levels of pair-wise relationships in the input graphs: the weights w_{ij} between the vertices i and j on the graph G (if $(i,j) \in E$), and the weights ρ_{ab} between the partite sets V_a and V_b of the graph. To score a multipartite subset H, we begin by designing a linkage function, $\mathcal{L}(i,H)$ that calculates the degree of membership of the element $i \in H$ to other elements in H. Considering the above informal requirements, the function $\mathcal{L}(i,H)$ should be designed such that its value is high when the elements in H are highly similar so that they form a dense subgraph. Since H is a multipartite subset, it can be decomposed as $H = \cup_{\ell=1}^{k} H_\ell$ where H_ℓ is the subset of objects from V_ℓ present in H, i.e, $H_\ell = H \cap V_\ell$. Recall that the similarity between vertices in the multipartite graph are represented by the matrix of weights, $\mathbf{W} = ||W_{ij}||$. The distance relationships between the partite sets are represented by the matrix $\mathbf{P} = ||\rho_{ab}||$, where $\rho_{ab} \ \ (\geq 0)$ represents the distance between the partite sets a and b (Fig. 2.1). Then, the function $\mathcal{L}(i,H)$ can be defined as:

$$\mathcal{L}(i,H) = \sum_{\substack{\ell=1 \\ \ell \neq g(i)}}^{k} \left(\sum_{j \in H_\ell} w_{ij} + \sum_{H_\ell \neq \emptyset} \rho_{g(i)\ell} \right) \tag{2.1}$$

The similarity between the element i and the subset H is computed using pairwise similarity across the partite sets only, so we ignore similar objects from the same partite set (expressed by the condition $\ell \neq g(i)$ in the outer summation). The term $\sum_{j \in H_\ell} w_{ij}$,

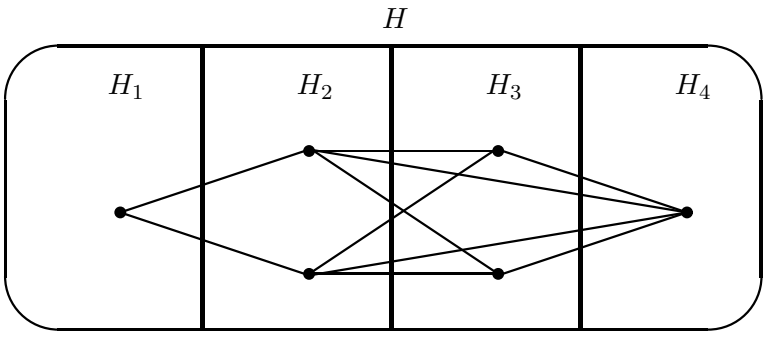

Figure 2.3: A schematic description of the element to subset relationship for scoring a multipartite subset. The subset H of V can be divided into four partite sets H_1 through H_4. The edges represent the similarity relationships between elements represented by the vertices, and $\rho_{H_iH_j}$ is assumed to be 1 for all i and j.

in (2.1), aggregates the pair-wise similarity values between the vertex i from the partite set $H_{g(i)}$ and all other vertices in the subset H that do not belong to the partite set $H_{g(i)}$. The second term, $\sum_{H_\ell \neq \emptyset} \rho_{g(i)\ell}$, is sum of the distances between the partite set containing the element i and other partite sets whose elements are included in H. A large positive value of this function $\mathcal{L}(i, H)$ ensures that the individual terms are large, implying that the element i is highly similar to other elements in H, and that H includes elements from distantly related partite sets. From a clustering point of view, this ensures large values of intra-cluster homogeneity and that clusters contain vertices from diverse partite sets. Of course, we could use some other similarity function, instead of the distance function $\rho_{g(i)\ell}$, to ensure that a cluster includes elements from closely related partite sets, but from the perspective of our biological (evolutionary) application it is important to make sure that the most diverse of such sets will be included in the cluster.

Then, the multipartite quasi-clique is defined as the subset with the highest score, or the subset containing elements such that similarity of the least similar element in H is maximum. So, the criterion for finding the quasi-clique can be written as[3]:

$$\max_{H \subseteq V} \mathcal{L}(i_H, H) = \max_{H \subseteq V} \min_{i \in H} \mathcal{L}(i, H) \tag{2.2}$$

[3]We could define the score of the subset H as the average of similarity between elements in H and the subset H, but maximizing this score will give a trivial solution. Alternately, it could be defined as the similarity of element with the median value of the element to subset relationship, in which case finding the maximum turns out to be a hard problem

And the cluster (or, quasi-clique) H^* satisfies the equality:

$$\begin{aligned} H^* &= \arg\max_{H \subseteq V} \mathcal{L}(i_H, H) \\ &= \arg\max_{H \subseteq V} \min_{i \in H} \mathcal{L}(i, H), \end{aligned} \tag{2.3}$$

where the element at which this minimum is attained is represented as i_H, so that the score for the subset H can be written as $\mathcal{L}(i_H, H)$.

2.5 Finding the optimal solution

To develop an algorithm for solving the optimization problem defined by (2.1) and (2.2), we observe the effect of increasing or augmenting (by including more elements) the subset H in $\mathcal{L}(i, H)$. The value of the function $\mathcal{L}(i, H)$ can only increase when H is augmented - this is because the distance function between the partite sets ρ_{ab} and similarity values w_{ij} take only positive values, and including more elements in H amounts to adding more terms with positive values. Formally, when an element k is added to the subset H, the function $\mathcal{L}(i, H)$ increases because:

$$\begin{aligned} \forall i \in H, \quad \forall k \in V \setminus H & \\ & \underbrace{\sum_{\substack{\ell=1 \\ \ell \neq g(i)}}^{k} \left(\sum_{j \in H_\ell} w_{ij} + \sum_{H_\ell \neq \emptyset} \rho_{g(i)\ell} \right) + w_{im} + \alpha}_{\text{value after adding the element } m \text{ to } H} \\ - & \quad \underbrace{\sum_{\substack{\ell=1 \\ \ell \neq g(i)}}^{k} \left(\sum_{j \in H_\ell} w_{ij} + \sum_{H_\ell \neq \emptyset} \rho_{g(i)\ell} \right)}_{\text{value before adding the element } m \text{ to } H} \\ = & \quad w_{im} + \alpha \geq 0 \end{aligned} \tag{2.4}$$

where α is 0 if the elements from the partite set $g(i)$ are not included in H, otherwise $\alpha = \rho_{g(k)g(i)} \geq 0$. This increasing property of the function $\mathcal{L}(i, H)$ when more elements are added to the subset H, can be exploited in designing an algorithm for finding the

H^* in (2.3).

Interestingly, there exists an algorithm by Matula [52] for finding the largest subgraph with maximum minimum degree, which can solve a very restricted case of our problem, when all edges have unit weights on them [4]. However, it does not work for weighted graphs, or when we must consider more complex relationships as those taking into account the weights between the partite sets as described in section 3.2 . We have therefore generalized the problem and now describe an algorithm for solving it.

2.5.1 The Algorithm for the Optimal Solution

The objective of the problem in (2.3) is to find the subset H for which the worst element is highly similar to other elements in H, i.e, $\min_{i\in H}\sum_{\ell=1:\ell\neq g(i)}^{k}(\sum_{j\in H_\ell} w_{ij} + \sum_{H_\ell\cap V_\ell\neq\emptyset}\rho_{g(i)\ell})$ is maximum over all possible subsets of V. As result of the above observation (2.4), increasing the subset H can only increase the value of the linkage function for any $i \in H$. We note that starting from V the only possibility to get a subset with a larger score than the score for the set V is to remove the element i_V (the element which is the least similar to other element in V). This is because the least similar element, i_V, has the lowest value of the linkage function among all elements in V, so it defines the score for the set V. If we were to remove any other element, $i^{'} \neq i_V$ the linkage function value for the element i_V will be further decreased. This is because according to (2.4) by keeping the element i_V but removing other elements from the set V, further decreases the similarity of i_V to other elements in V. So, the resulting subset $V \setminus \{i^{'}\}$ will have a score less than the score for the set V. But, we are interested in moving towards a subset with the largest score value, and the only way to do it is to remove the element i_V with the minimum value of the linkage function. Based on these observations, the algorithm for finding the optimal solution for (2.3) is presented in Table 2.2 and described below.

Algorithm. The algorithm is iterative and begins by calculating the linkage function value (2.1) for all $i \in V$ and finds the element(s) in V with the minimum value of the

[4]The original algorithm by Matula works with a simple graph, but it is easy to adapt it for analyzing a multipartite graph.

Input: The graphs G^k and G.
Output: The optimal solution, Γ and a nested family of subsets,
$\mathcal{H} = \{H^1, H^2, \ldots, H^{T-1}, H^T\}$ such that $V = H^1 \supset H^2 \supset \ldots \supset H^{T-1} \supset H^T$

Initialization: Set $t := 1$; $H^1 := V$; $\Gamma := V$;

STEP 1: Find $M^t := \{i : \mathcal{L}(i, H^t) = \min_{j \in H^t} \sum_{\substack{\ell=1 \\ \ell \neq g(j)}}^{k} \left(\sum_{m \in H_\ell^t} w_{jm} + \sum_{H_\ell^t \cap V \ell \neq \emptyset} \rho_{g(j)\ell} \right)$

comment: *Find the least similar element(s) in the set H^t as the elements for which (2.1) takes the minimum value.*

STEP 2: if $((H^t \setminus M^t = \emptyset) \vee (\mathcal{L}(i, H^t) = 0 \ \forall i \in H^t))$ STOP.

comment: *Stop when no more elements are left, or the left over elements have no similarity between them.*

else $\{ H^{t+1} := H^t \setminus M^t;\ t := t + 1; \}$

comment: *Update the iteration number and find the next nested subset.*

if $(\min_{j \in H^t} \sum_{\substack{\ell=1 \\ \ell \neq g(j)}}^{k} \left(\sum_{m \in H_\ell^t} w_{jm} + \sum_{H_\ell^t \cap V \ell \neq \emptyset} \rho_{g(j)\ell} \right) > \min_{j \in \Gamma} \sum_{\substack{\ell=1 \\ \ell \neq g(j)}}^{k} \left(\sum_{m \in \Gamma_\ell} w_{jm} + \sum_{\Gamma_\ell \cap V \ell \neq \emptyset} \rho_{g(j)\ell} \right))$

$\{\Gamma = H^t; \}$

comment: *If the score value of the current subset is higher than any of the subsets encountered so far, make it the subset with the highest score.*

go to Step 1.

Table 2.2: Pseudocode for finding the optimal solution, H^*.

linkage function, thus calculating the score for the set V as $\min_{i \in V} \sum_{\substack{\ell=1 \\ \ell \neq g(i)}}^{k} \left(\sum_{j \in V_\ell} w_{ij} + \rho_{g(j)\ell} \right)$. The set M^1 containing these elements with minimum value is determined. The elements in the set M^1 are removed from V to get $H^2 = V \setminus M^1$. At the iteration t, it considers the set H^{t-1} as input, identifies the subset M^t such that $M^t = \{i \in H^{t-1} : \sum_{\substack{\ell=1 \\ \ell \neq g(i_t)}}^{k} \left(\sum_{m \in H_\ell^{t-1}} w_{im} + \sum_{H_\ell^{t-1} \cap V \ell \neq \emptyset} \rho_{g(i)\ell} \right) = \min_{j \in H^{t-1}} \sum_{\substack{\ell=1 \\ \ell \neq g(j)}}^{k} \left(\sum_{m \in H_\ell^{t-1}} w_{jm} + \sum_{H_\ell^{t-1} \cap V_\ell \neq \emptyset} \rho_{g(j)\ell} \right) \}$ and finds the score for the subset H^{t-1} as the value $\min_{j \in H^{t-1}} \sum_{\substack{\ell=1 \\ \ell \neq g(j)}}^{k} \left(\sum_{m \in H_\ell^{t-1}} w_{jm} + \sum_{H_\ell^{t-1} \cap V_\ell \neq \emptyset} \rho_{g(j)\ell} \right)$. Then, the subset M^t is removed from H^{t-1} to produce $H^t = H^{t-1} \setminus M^t$. The algorithm terminates at the iteration T when $H^T = \emptyset$, or when elements in the set H^T are not related to each other. It outputs the optimal solution H^* as the subset, Γ which is the the largest subset, H^j in the sequence $\mathcal{H}$ (the subset with the smallest t) such that $\min_{i \in H^t} \sum_{\substack{\ell=1 \\ \ell \neq g(i)}}^{k} \left(\sum_{m \in H_\ell^t} w_{im} + \sum_{H_\ell^t \cap V_\ell \neq \emptyset} \rho_{g(j)\ell} \right) \geq \min_{i \in H^s} \sum_{\substack{\ell=1 \\ \ell \neq g(i)}}^{k} \left(\sum_{m \in H_\ell^s} w_{im} + \sum_{H_\ell^s \cap V_\ell \neq \emptyset} \rho_{g(i)\ell} \right)$

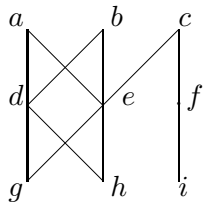

Figure 2.4: An example graph for illustrating the algorithm

$\forall s \in \{1, 2, \ldots, T\}$. Below we will prove that this algorithm does indeed find H^* satisfying (2.3). Moreover, the solution found by this algorithm is the largest in a set-theoretical sense and includes all the smaller subsets which are solutions. In other words, it is the largest solution.

An example

We now illustrate the procedure with the help of an example. Consider the three partite graph with nine nodes, $V = \{a, b, c, d, e,\ f, g, h, i\}$, where $V_1 = \{a, b, c\}, V_2 = \{d, e, f\}$ and $V_3 = \{g, h, i\}$ as shown in Fig. 2.4 . An edge, e_{kl}, between any pair of nodes k and l implies that objects represented by the nodes k and l are similar. For simplicity, we consider a unit weight on edges and equi-distant partitions i.e, the distance $\rho_{ab} = 0$ for all pairs of partite sets (V_a, V_b). In this case, the linkage function $\mathcal{L}(k, H) = \sum_{l \in H: k \neq l} e_{kl}$ defines the similarity between an element and a subset H as the cumulative similarity of the element, k, to the subset, H. The first iteration of the algorithm removes the weakest element i. The elements f and c are removed at iterations 2 and 3, respectively. The corresponding values of the objective function, or the score value, at these iterations are $\mathcal{L}(i, V) = \mathcal{L}(f, V \setminus \{i\}) = \mathcal{L}(c, V \setminus \{i, f\}) = 1$. At iteration 4, elements $a, b, g,$ and h are removed since the linkage function value corresponding to each of these elements is 2. Finally, at iteration 5 the remaining elements are removed because both have the linkage function value 0. The optimal solution is the subset $\{a, b, g, h, d, e\}$, and is like a "quasi-clique" which corresponds to an intuitive notion of a cluster for the example, so one could consider the optimal subset as a cluster. □

The optimal solution H^* obtained above has the property that it is a unique subset isolated from the rest of the set in such a way that adding elements not in H^*, or

removing any group of elements from H^* will affect its score adversely. This property allows us to interpret the optimal solution as a cluster.

Theorem 1. *The subset* Γ *output by the above algorithm is the optimal solution* H^* *(2.3) for the set* V*, and the linkage function defined by (2.1).*

Proof. According to the algorithm, $\Gamma = \arg\max_{H^t \in \mathcal{H}} \min_{i \in H^t} \sum_{\substack{\ell=1 \\ \ell \neq g(i)}}^{k} \left(\sum_{m \in H_\ell^t} w_{im} + \sum_{H_\ell^t \cap V_\ell \neq \emptyset} \rho_{g(i)\ell} \right)$, where $\mathcal{H} = \{H^1, H^2, \ldots, H^T\}$ and $H^T \subset \ldots \subset H^2 \subset H^1 = V$. Depending on the relationship of an arbitrary subset H with Γ, the proof for Γ as the optimal solution can be divided into two cases: (i) $H \setminus \Gamma \neq \emptyset$ and, (ii) $H \subseteq \Gamma$.

Case (i) $[H \setminus J \neq \emptyset]$: In this case, we prove that if the subset H has any elements not in Γ, then it will have a score value strictly less than that of Γ. Consider an arbitrary set H such that $H \setminus J \neq \emptyset$ and let H^t be the smallest set in the sequence $\mathcal{H}$ containing H, so that $H \subseteq H_t$ but $H \not\subseteq H^{t+1}$. Since $M^t = H^t \setminus H^{t+1}$, there is at least one element, say j, common to both M^t and H. By construction, j is the element in H^t with the least value of the linkage function and so by definition, the the score for H^t is $s = \sum_{\substack{\ell=1 \\ \ell \neq g(j)}}^{k} \left(\sum_{m \in H_\ell^t} w_{jm} + \sum_{H_\ell^t \cap V_\ell \neq \emptyset} \rho_{g(j)\ell} \right)$. Since j is also an element in H, the score for H can be at most s, which is the score for the set $H^t \neq \Gamma$. The score for the subset H will be less than the score for the subset Γ, because Γ is the highest scoring in the family of subsets, $\mathcal{H}$ which includes H^t. So, we have:

$$\forall H \setminus \Gamma \neq \emptyset$$

$$\min_{i \in \Gamma} \sum_{\substack{\ell=1 \\ \ell \neq g(i)}}^{k} \left(\sum_{m \in \Gamma_\ell} w_{im} + \sum_{\Gamma_\ell \cap V_\ell \neq \emptyset} \rho_{g(i)\ell} \right) > \min_{i \in H} \sum_{\substack{\ell=1 \\ \ell \neq g(i)}}^{k} \left(\sum_{m \in H_\ell} w_{im} + \sum_{H_\ell \cap V_\ell \neq \emptyset} \rho_{g(i)\ell} \right)$$

Case (ii) $[H \subseteq \Gamma]$: Here we prove that any subset H of Γ cannot have a larger score than that of Γ. Similarly to the previous case, there exists a smallest subset H^t in the sequence $\mathcal{H}$ that includes H. Then, the score value of H cannot be larger than the score value of H^t because (as in the previous case) the element $i_{H^t} = \arg\min_{i \in H^t} \sum_{\substack{\ell=1 \\ \ell \neq g(i)}}^{k} \left(\sum_{m \in H_\ell^t} w_{im} + \sum_{H_\ell^t \cap V_\ell \neq \emptyset} \rho_{g(i)\ell} \right)$ with the least value of the linkage function among all

elements in H^t also belongs to the subset H (by construction of the subset H^t). So, the score value of the subset H is at most $\min_{i \in H^t} \sum_{\substack{\ell=1 \\ \ell \neq g(i)}}^{k} \left(\sum_{m \in H^t_\ell} w_{im} + \sum_{H^t_\ell \cap V_\ell \neq \emptyset} \rho_{g(i)\ell} \right)$ but this value can at most be $\min_{i \in \Gamma} \sum_{\substack{\ell=1 \\ \ell \neq g(i)}}^{k} \left(\sum_{m \in \Gamma_\ell} w_{im} + \sum_{\Gamma_\ell \cap V_\ell \neq \emptyset} \rho_{g(i)\ell} \right)$, since according to the algorithm the subset Γ has the maximum score value in the family of subsets $\mathcal{H}$. So, this case is also proved.

□

2.6 Analysis and Implementation

The run-time of the algorithm for finding a cluster (Table 2.2) depends on the efficiency of evaluating the linkage function (2.1) due to the frequent evaluation of this function during each iteration. The linkage function is additive and can be updated efficiently when vertices are removed from the set simply by subtracting the sum of two terms. For instance, when a vertex j is removed, the new linkage function value $\mathcal{L}(i, H)$ for the element i can be obtained from the already-computed linkage function value $\mathcal{L}(i, H \cup \{j\})$ by subtracting the term $w_{ij} + \alpha$ as shown in (2.4).

During the initialization of the algorithm, we must compute the linkage function for all the vertices in V. The linkage function value for a single element i depends on the weights on edges (from other elements) incident on it and on the relationship of the partite set $g(i)$ to other partite sets in H. Thus, to compute the initial linkage function values for all the vertices in V, we must look at all the edges in E and all the relationships between the partite sets. We assume that the number of partite sets k is very small compared to the size of V, so the complexity of computing the initial linkage function values is $O(|E|)$. At subsequent iterations, each vertex is deleted at most once; the vertex deletion entails deletion of all the incident edges and updating the linkage function values for all the neighboring vertices of the deleted vertex. As the resulting linkage function value can be updated by subtracting the contribution due to the deleted edge, the total number of updates to the linkage function is bounded by the total number of edge deletions which is at most $|E|$. So, the initial linkage function

computation and all subsequent updates to it together can be $O(|E|)$ time. In what follows, the cost of calculating and updating the linkage function values remains fixed ($O(|E|)$ and for calculating the overall efficiency of the algorithm we focus on other steps in the algorithm.

Before going into specific data structures for an efficient implementation of the algorithm given in Table 2.2, we describe a naive implementation of the algorithm. In step 1 of this algorithm, we need to find the vertices with the minimum value of the linkage function. This can be done by looking at the vertices active at that iteration. Since the total number of vertices at any iteration i is at most $|V| - i$ and the total number of iteration is at most $|V|$ (achieved when a single vertex is deleted at each iteration), the total time required for finding the minimum values over all iterations is $\sum_{i=1}^{|V|} |V| - i = \sum_{i=1}^{|V|} i = |V|(|V|-1)/2 = O(|V|^2)$. In step 2, two conditions are tested which take a constant time, and some elements are removed from the set H^t. Since each element is removed only once, the time required for executing the step 2 during all the iterations is $O(|V|)$. Thus, the total time required for this straightforward implementation is $O(|E| + |V|^2 + |V|) = O(|E| + |V|^2)$.

As described in the algorithm in Table 2.2 and discussed above, three operations are performed at each iteration of the algorithm. These are:

- `find-min:` This corresponds to Step 1 of the algorithm where M^t, the set of elements with the minimum value of the linkage function are determined.

- `delete-min:` This corresponds to the statement $H^{t+1} = H^t \setminus M^t$ in the step 2 of the algorithm. It involves removing the elements in the set M^t to find the list of active elements H^{t+1} for the subsequent iteration.

- `decrease-key:` Deleting the elements in the set M^t entails updating the linkage function values for the neighbors of elements (in the Graph G^k) in the set M^t. The total cost of updating the linkage function values across all iterations is $O(|E|)$ and is already considered. This operation refers to the cost of finding the new position, for elements whose linkage function values are affected (decreased), in the data structure for storing the elements according to their linkage function

values.

Having decomposed steps at each iteration into the three operations described above, an efficient implementation attempts to minimize the total time consumed in these operations. We now give some data structure choices that enable us to implement exactly the same algorithm efficiently. These choices and their corresponding complexities are stated in the theorem below.

Theorem 2. *[30] Using Fibonacci heaps, the* `find-min` *and the* `delete-min` *operations can be done in worst case time* $O(1)$ *and* $O(\log n)$ *time. The decrease-key operation can be done in* $O(1)$ *amortized time.*

The details of the Fibonacci heaps and how to perform the three operations are given in Appendix A. A Fibonacci heap is a collection of trees (binomial heaps) satisfying the minimum-heap property, i.e., the key (value) stored in a child is at least the key of the parent. The collection of binomial heaps is organized into a circularly linked list with a pointer to the minimum element. Thus the `find-min` operation can be performed in constant time. The `delete-min` operation involves removing the element pointed to by the minimum pointer from its binomial heap. To restore the invariant that minimum pointer points to the minimum value, this pointer must be updated to the new minimum element. The new minimum element can be found in $O(\log |V|)$ worst case time by searching the circularly linked list since the size of this list is bounded by $O(\log |V|)$.

The `find-min` operation in $O(1)$ time and the `delete-min` operation in $(\log |V|)$ time can be carried out on any heap data structure. The actual advantage of using Fibonacci heaps is in efficiency of the `decrease-key` operation. In the `decrease-key` operation, the key of an element is decreased, and consequently a new position which preserves the heap property must found. Fibonacci heaps are flexible data structures that allow certain deviation from the heap property. This fact, coupled with lazy restoration to the heap property allows them to prolong the restoration operations until a sufficient amount of work has accumulated. By taking action only when impending work reaches a critical threshold, they avoid unnecessary repetition in work and this

results in improving overall efficiency. So, although a particular `decrease-key` operation may take much longer, the average over a large number of such operations (which is precisely the case we have in our algorithm as we must perform this operation at each of the $O(|V|)$ iterations) is $O(1)$. The details of this operation are described in Appendix A.

Theorem 3. *The algorithm, given in Table 2.2, for finding a cluster runs in time* $O(|E| + |V| \log |V|)$ *and space* $O(|E| + |V|)$.

Proof. Clearly, step 0 of the algorithm in Table 1 takes a constant time. The initialization includes computing $\sum_{\substack{\ell=1 \\ \ell \neq g(i)}}^{k} \left(\sum_{m \in H_\ell^t} w_{im} + \rho_{g(i)\ell} \right) \ \forall i \in V$. For computing this summand, we must look at all edges incident on i, thus computing this value for all the vertices takes $O(|E|)$ time. At subsequent iterations, due to the additive property of the linkage function (2.1), efficient updates are possible without recomputing from scratch. So, updating the linkage function value, $v = \sum_{\substack{\ell=1 \\ \ell \neq g(i)}}^{k} \left(\sum_{m \in H_\ell} w_{im} + \rho_{g(i)\ell} \right)$ for an element i in H after deleting an element $j \in H$ amounts to subtracting $w_{ij} + \alpha$ from v, where $\alpha = \rho_{g(i)g(j)}$ if j is the only element from partite set $g(j)$ in H, otherwise $\alpha = 0$. As each edge is deleted once, as described earlier in this section, all linkage function updates together requires $O(|E|)$ time.

Step 1 involves determining the set M_t by finding the vertices with the minimum value of the linkage function. Observe that in a sparse multipartite graph, only a few edges are deleted at each iteration, implying that only a few linkage function values are updated. Consequently, the order of vertices determined by the linkage function values remains approximately fixed. We use the Fibonacci heap [17, 30] data structure for storing the vertices according to their linkage function values.

As a consequence of Theorem 2, by using the Fibonacci heaps, the elements in the set M_t can be found in constant time ($O(1)$) using the `find-min` operation. In Step 2, the set $H_{t+1} = H_t \setminus M_t$ can be found in $O(\log |H_t|)$ amortized time using the `delete-min` operation and each update to the linkage function value can be performed in $O(1)$ amortized time using the `decrease-key` operation. Thus, using the Fibonacci

heaps, each iteration takes $O(\log |H_t|)$ amortized time [5]. The maximum number of iterations is $|V|$, and each iteration takes at most $O(\log |V|)$. Thus, the algorithm runs in $O(|E| + |V| \log |V|)$ time.

Space Complexity: We use the adjacency list format for storing the graph during computation. By storing the edges between the vertices of the multipartite graph, G^k, in the adjacency list for each of the vertices, each edge is stored in the adjacency lists of the two vertices it is incident on. Thus, we store $2|E|$ edges in total, requiring $O(|E|)$ space for storing edges. The identification of the partite set for each vertex is stored in a special field (size 1) in its adjacency list. There are $|V|$ lists which can be stored in a total size $2|E|$. Thus, the space complexity for storing the multipartite graph is $O(2|E| + |V|) = O(|E| + |V|)$ (ignoring the space required for the matrix $\mathbf{P}$ which we assume is negligible compared to $|E|$ and $|V|$). □

We now show that by choosing efficient data structures and by placing a restriction on the weights on the edges, it is possible to further improve the complexity of the algorithm. This involves using a data structure for implementing Bucket sort [17] which is briefly described below.

Bucket sorting (also called bin sorting) allows sorting in linear time for elements whose range of keys is linear in the number of elements to be sorted. For instance, if sorting n elements whose key values are between 1 and n, can be sorted (according to their key values) by using an array of size n and placing each element at the array index equal to the key value associated with that element. In case there are multiple elements with the same key value, they are tied together into a linked list and a pointer to the linked list is stored at that array index.

In the implementation that will be described shortly, we store the elements according to their linkage function values (keys) in an array of size linear in $|V|$ (see Figure 2.5).

[5] It must be emphasized that we are interested in total time spent over the entire algorithm, so amortized time is a better performance parameter than the worst case time. For Fibonacci heaps the worst case time for a particular instance of `delete-min` or `decrease-key` operation can be higher, but it is balanced by the other cheaper instances of operations in the same class. As we perform a large number of each kind of operation, amortized time certainly is a more realistic performance evaluation measure.

Each bucket (bin or cell of the array) is indexed using its array index and contains a pointer to a linked list of elements having the same linkage function value. In particular, there is a pointer to the first non-empty cell which acts as a pointer to the element with the minimum value of the linkage function. Also, from each non-empty cell, there is a pointer to the next non-empty cell. The `find-min` operation is carried out by accessing the element(s) in the cell pointed to by the minimum pointer and is clearly a constant time operation on this data structure. The `delete-min` operation is also a constant time operation and is performed by simply removing the elements found using the `find-min` operation and updating the minimum pointer by following the pointer (from the cell pointed to by the minimum pointer) to the next occupied cell. The complexity of the `decrease-key` operation depends on the actual amount by which the key of an element (linkage function value of an element) is decreased. If this decrement is bounded by a constant, as will happen in the case discussed below, the new position for the element is a constant number of cells away from its current cell and can be found in constant time. Thus, `decrease-key` also takes a constant time.

Theorem 4. *If the values in the similarity matrices,* $\mathbf{W}$ *and* $\mathbf{P}$*, are discretized into* c *different values, the algorithm runs in* $O(|E| + |V|)$ *time.*

Proof. The initialization step, as in Theorem 2, takes $O(|E|)$ time. Due to the additive property of the linkage function described earlier in Theorem 2, all linkage function value computations and their updates can be done in $O(|E|)$ time. The essential difference lies in using another efficient data structure which exploits the bounded and known range of values taken by the linkage function values.

The key observation for this proof follows from the discretization of the edge weights, since the weights on the edges can assume only a constant number, say c, of different values, the linkage function values are bounded. Without loss of generality, we assume that these c values range from 0 to $c-1$. Then, the initial linkage function values are bounded i.e., $0 \leq \sum_{\substack{\ell=1 \\ \ell \neq g(i)}}^{k} \left(\sum_{m \in H_\ell^t} w_{im} + \rho_{g(i)\ell} \right) \leq (c-1) \cdot (|V| + k)$. The value 0 is achieved when all the terms in the parenthesis are 0; the upper bound is achieved when all the terms in the parenthesis have the value c.

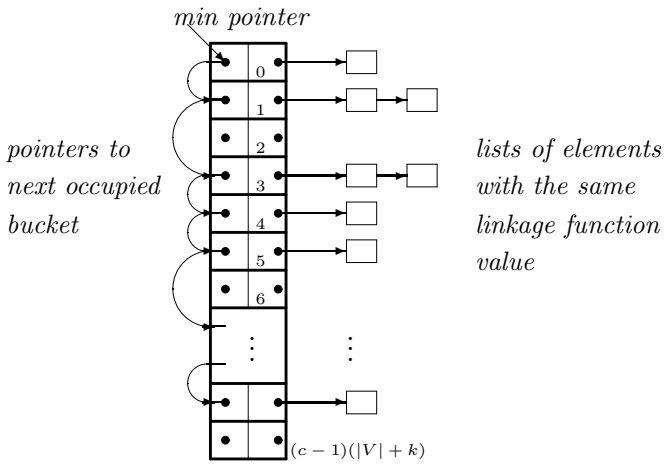

Figure 2.5: A detailed illustration of using Bucket sort for efficiently accessing the elements with the minimum value of the linkage function value. Apart from the index indicated by the number in the boxes in second column, each bucket (entry in the array) contains two pointers: one to the list of elements corresponding to index of that bucket, and another to the next non-empty bucket. Also, there is a special min-pointer that points to the first non-empty bucket.

To reduce the time complexity of finding the minimum at each iteration, we use the bucket sort algorithm [17] with $c \cdot (|V| + k)$ buckets. We store the elements in an array which has $c \cdot (|V| + k)$ entries as shown in Fig. 2.5. In this array, each element is stored at the index equal to the linkage function value for that element. By using such a storage, this process results in sorting the elements according to their linkage function values, so the element with the minimum linkage function value occupies the first non-empty index of the array. Note that it is possible that two or more elements have the same linkage function value. In such cases these elements are stored in a linked list with the head of the list stored at the index given by the linkage function value of those elements. Apart from this, we keep a pointer to the smallest occupied index in the array to be able to find and delete the minimum element in constant time. It is likely that many slots in the array will be empty, so we also keep a pointer at each index to the next occupied index in the array for efficiently navigating the array indices.

At each iteration, finding the vertex with the minimum value of the linkage function takes a constant time because the vertices are sorted according to the linkage function values, and we keep a pointer to the first nonempty slot (this slot contains the minimum element) of the array. Deletion of a vertex entails updates to the linkage function values

for the neighboring vertices. The cost for updating linkage function values has already been already considered (first paragraph of this proof). To preserve the sorted order of elements, we must find the new place for each updated element. Since the edge weights are discretized, the new place must lie at most c bins away from the current position (towards the minimum, or the decreasing array indices). Thus, the new place is found in at most c, or $O(1)$ time. Every iteration requires $O(1)$ time, and since the number of iterations is at most $|V|$, the total time is bounded by $O(|V|)$. Combining this with the total cost for computing the linkage function values, the run time of the algorithm is $O(|E| + |V|)$. □

2.7 Partitioning data into multipartite clusters (MPC)

The algorithm in Table 2.2 produces as output one multipartite cluster. However, many such clusters are likely to be present in the set V. If *we assume that these clusters are unrelated to one another*, then iteratively applying the above procedure will extracting all the clusters. To do this we remove the elements belonging to the first cluster H^* from V and extract another multipartite cluster in the set $V \setminus H^*$. This procedure is formalized in Table 2.3.

Initialization: $V^0 := V$; $m := 0$; $\mathcal{C} = \emptyset$;
Step 1: Extract H_m^* from V^m algorithm in Table 2.2; Add H_m^* to $\mathcal{C}$;
Step 2: $V^{m+1} := V^m \setminus H_m^*$; $m := m + 1$;
Step 3: if ($(V^m = \emptyset) \wedge (w_{ij} = 0 \, \forall i, j \in V^m)$)
Output $\mathcal{C}$, V^m as R, and m; STOP;
else go to step 1

Table 2.3: Pseudocode for finding a series of multipartite clusters.

This procedure outputs an ordered set, $\mathcal{C} = \{H_0^*, H_1^*, \ldots, H_m^*\}$, of m clusters, and a set of residual elements $R = \{i : i \in V \setminus \mathcal{C}\}$. The vertices in the residual set R are trivial clusters - the singletons, which are not similar to any element from other partite sets. The number, m, of non-trivial clusters is automatically determined by the method. The clusters in $\mathcal{C}$ are ordered according to their score values, i.e., any cluster H_s^* has score value strictly larger than the score values for clusters H_t^* where $t = \{s + 1, \ldots, m\}$. It

must be emphasized that every cluster in $\mathcal{C}$ contains members from at least two partite sets.

As a result of Theorem 3, the procedure for finding all the m clusters given in Table 2.3 runs in time $O(m(|E| + |V|))$. We now give some implementation details which do not improve the complexity of the algorithm but enable significant speedups in practice both for finding one cluster (Table 2.2) and for the iterative finding of multiple clusters (Table 2.3).

Vertices in different connected components of a graph cannot come together to form a cluster. So, different connected components can be processed in isolation. Furthermore, the input multipartite graph, in general, is large but very sparse, so when a dense subgraph is extracted as the optimal solution, the remaining graph becomes disconnected. As a consequence, in practice, a significant speedup is achieved when the procedure in Table 2.3 is run on individual connected components after extraction of a cluster. Also, finding an optimal solution within different connected components is amenable to parallelism.

2.8 Cluster Aggregation

We have so far defined a cluster as the optimal solution for the optimization problem defined in (2.3). From a practical point of view, this definition can be considered as suitable for very tight (homogeneous) clusters. After finding such clusters, we should be able to merge them while maintaining homogeneity within resulting clusters. In other words, this will allow us to estimate hyper-relationships between the clusters and obtain larger clusters. Moreover, we have empirically observed that merging clusters produced by the procedure in Table 2.3 improves correlation with known manually extracted clusters (see the chapter on ortholog clustering). In this section, we describe a procedure for aggregating existing clusters.

Merging an existing set of clusters, $\mathcal{C} = \{H_0^*, H_1^*, \ldots, H_m^*\}$, can be formulated as a graph clustering problem on the weighted graph $\hat{G}(\mathcal{C}, \hat{E}, \hat{W})$ whose vertices correspond

to existing clusters, i.e., they are elements of the set $\mathcal{C}$. The weight $\hat{w}_{ij} \in \hat{W}$ corresponding to the edge $\hat{e}_{ij} \in \hat{E}$ represents the strength of (hyper)-similarity between the existing clusters H_i^* and H_j^*. Then, we can apply the proposed clustering paradigm to the graph $\hat{G}$; however, it must be emphasized that this graph is not a multipartite graph. We can still respect the multipartite nature of the original input graph G^k by designing an appropriate similarity measure between clusters. Furthermore, unlike the initial clustering where we had two matrices of similarities (one describing similarities between the elements and the other between the partite sets), in this stage we have only one type of similarity.

A central issue in aggregating clusters is to find a relevant measure of similarity between the clusters. We define two classes of similarity measures a) cluster similarity based on nearest neighbors of cluster members, b) cluster similarity based on actual similarity values among cluster members. These measures of pair-wise similarity between clusters depend on the sizes of the clusters, so we normalize these similarity measures by the size of the constituent cluster so that similarity between any two clusters depends on the pair-wise similarity between cluster members and not on the cluster size.

2.8.1 Nearest-neighbor similarity between cluster members

As the name suggests, this measure of similarity between two clusters indicates the number of nearest neighbor relationships between the members of the two clusters. This similarity is motivated by the fact that if the nearest neighbors for most of the elements in the two clusters are contained in each other, the elements in them are likely to be very similar. So, the elements from two such clusters can be aggregated to form a single cluster.

Denoted by $s_{nn}(C_l, C_m)$, the nearest-neighbor based similarity between the clusters C_l and C_m is the sum of cardinalities of the two sets: $s(C_l, C_m)$ and $s(C_m, C_l)$. The set $s(C_l, C_m)$ contains those elements from the clusters C_l whose nearest neighbors, outside C_l, lie is the cluster C_m. Likewise, the set $s(C_m, C_l)$ is a subset of elements from C_m whose nearest neighbors, in the set $V \setminus C_m$, are contained in the cluster C_l. Formally,

$s(C_l, C_m)$ is defined as:

$$s(C_l, C_m) = \{i \in C_l : (j^* = \arg \max_{j \in V \setminus C_l} w_{ij}) \wedge (j^* \in C_m)\} \tag{2.5}$$

In other words, $s(C_l, C_m)$ is a collection of those elements from C_l which are most similar to some element in C_m. The measure $s(C_l, C_m)$ is not symmetric, i.e., $s(C_l, C_m) \neq s(C_m, C_l)$ but $s(C_m, C_l)$ can be computed using (2.5). Then, the nearest-neighbor based cluster similarity $s_{nn}(C_l, C_m)$ is defined as

$$s_{nn}(C_l, C_m) = |s(C_l, C_m)| + |s(C_m, C_l)| \tag{2.6}$$

It must be remarked that $s_{nn}(C_l, C_m)$ implicitly depends on the size of the clusters, so for a pair of large related clusters (clusters whose members are similar) we would expected this similarity to be higher than that between a pair of similar smaller clusters. The number of nearest neighbor relations between any two clusters is limited by the size of the smaller cluster. And clearly, $0 \leq s_{nn}(C_l, C_m) \leq 2\min(|C_l|, |C_m|)$. Moreover, $s_{nn}(C_l, C_m) = 0$ when C_l and C_m are unrelated and this measure assumes non-zero value for related clusters. For any two related clusters, the similarity value is high when these clusters have similar cardinalities, but when one cluster is relatively large compared to the other, the similarity value is lower. As we would like to consider clusters as individual vertices for subsequent clustering, we introduce the following similarity measure that is normalized for cluster size:

$$s'_{nn}(C_l, C_m) = \left\lfloor \frac{s(C_l, C_m) + s(C_m, C_l)}{2\min(|C_l|, |C_m|)} \right\rfloor \tag{2.7}$$

2.8.2 Mean similarity between cluster members

The cluster similarity measure discussed above estimates the similarity between two clusters using the nearest neighbor relationships between the elements from those clusters. Since this measure does not consider the actual quantitative similarity, the nearest neighbors could actually be very far from each other. Considering this, a simple count

of the number of nearest neighbors as expressed by $s_{nn}(C_l, C_m)$ is not the best cluster similarity measure. To overcome this drawback, we now introduce a cluster similarity measure that depends on the actual strength of similarities between elements across the clusters. Denoted by $s_s(C_l, C_m)$, the pair-wise element similarity strength based cluster similarity measure between the clusters C_l and C_m is defined as:

$$s_s(C_l, C_m) = \sum_{i \in C_l} \sum_{\substack{j \in C_m \\ l \neq m}} w_{ij} \tag{2.8}$$

In other words, $s_s(C_l, C_m)$ is defined as the sum of the weights of the edges whose two end points lie in C_l and C_m respectively. This similarity measure has an advantage over the nearest-neighbor based similarity - being based on the actual measures of similarity between cluster members, it reflects the similarity between clusters observed in the original data, so that we can decide if it is desirable to merge the two clusters. However, this measure also needs to be normalized by the sizes of the two clusters. We normalize $s_s(C_l, C_m)$ by the total number of possible edges between C_l and C_m, or

$$s'_s(C_l, C_m) = \left\lfloor \frac{\sum_{i \in C_l} \sum_{\substack{j \in C_m \\ l \neq m}} w_{ij}}{|C_l||C_m|} \right\rfloor \tag{2.9}$$

2.8.3 Aggregating Clusters

The two similarity measures described above allow us to transform hyper-relations between clusters into simple relationships between existing clusters to obtain a weighted graph. The clustering methods described in this chapter can then be used to find phase 2 clusters. In fact, this process can be continued until we are left with a single cluster. Each phase of such a cluster aggregation process provides an overview of the organization of clusters from the previous stage, and also allows us to see into the global structure in the original data.

2.9 Conclusions

In this chapter we discussed the necessity and suitability of multipartite graphs for representing relationships in data with a given partition of elements and with an emphasis on analyzing relationships across the given partitions. We presented a definition of a quasi-clique as a subset with strongly-related elements. This definition admits an efficient polynomial time procedure for which we described several implementation choices that lead to significant performance gains in practice. Finally, we described a similarity measure between clusters for merging them into highly aggregated clusters. Although the formulations studied in this chapter have been studied for a specific linkage function for a multipartite graph, we shall see in the next chapter that the methods described here are very general and applicable whenever the linkage function and the optimization criterion derived from it satisfy certain abstract properties.

Chapter 3

Generalized framework for multipartite graph clustering using quasi-concave set functions

3.1 Fundamental property of the objective function whose maximum produces a cluster

In the previous chapter, the clustering problem on a multipartite graph was formulated as a combinatorial optimization problem using the very specific objective function $F(H) = \min_{i \in H} \mathcal{L}(i, H)$, where $\mathcal{L}(i, H)$ is the element-to-set linkage function defined by (C.1). In this chapter, we consider the maximal generalization of the function which preserves the fundamental property of only spending polynomial time in finding the global optimum (maximum). This is stated by the following theorem.

Theorem 5. *[53] Let $\mathcal{L}(i, H)$ be a monotone increasing linkage function: $\forall H', H$: $H' \supseteq H \Rightarrow \mathcal{L}(i, H) \leq \mathcal{L}(i, H')$. If for any (i, H) calculation of the value $\mathcal{L}(i, H)$ has polynomial complexity at each of its elementary steps, then the optimization of the function*

$$F(H) = \min_{i \in H} \mathcal{L}(i, H) \tag{3.1}$$

also has polynomial complexity.

From this perspective, a principal proof that H^* from (3.2) gives the global maximum for $F(H) = \min_{i \in H} \mathcal{L}(i, H)$ is reduced to trivially noticing that $\mathcal{L}(i, H)$ from (3.1) is a monotone function of H [1].

[1] Our independent proof in chapter 3 which is based on specifics of (3.1) is not only more simple and direct than the general from [53], but automatically yields many additional properties of the optimization procedure. As mentioned in chapter 3, this particular objective function is the basis for

This general perspective on the objective function $F(H)$ whose optimum can be found efficiently in polynomial time encourages us to find different criteria other than those in (3.1) and (3.2) which have essentially the same abstract properties and hence follow similar directions in finding a cluster like those of using the score function analyzed in chapter 3. Intuitively, the three parametric families of the score functions presented below preserve the same flavor of the algorithm for finding the optimum value:

$$F_1(H) = \min_{i \in H} \sum_{\substack{\ell=1 \\ \ell \neq g(i)}}^{k} \sum_{j \in H_\ell} w_{ij} + \left(\sum_{H_\ell \cap V_\ell \neq \emptyset} 1 \right)^{\alpha}, \text{ where } \alpha \geq 0 \tag{3.2}$$

$$F_2(H) = \min_{i \in H} \sum_{\substack{\ell=1 \\ \ell \neq g(i)}}^{k} \left(\rho_{g(i)\ell}\right)^{\alpha} \left(\sum_{j \in H_\ell} w_{ij} \right)^{\beta}, \text{ where } \alpha, \beta \geq 0 \tag{3.3}$$

$$F_3(H) = \min_{i \in H} \sum_{\substack{\ell=1 \\ \ell \neq g(i)}}^{k} \left(\rho_{g(i)\ell}\right)^{\alpha} \left(\sum_{j \in H_\ell} w_{ij} - \sum_{j \in V_\ell \setminus H_\ell} w_{ik} \right)^{\beta} \text{ where } \alpha, \beta \geq 0 \tag{3.4}$$

where the notation is consistent with that followed in chapter 2. The weights w_{ij} represents the weighted similarity between the elements i and j, H_ℓ is a subset of elements from the partite set V_ℓ and $\rho(g(i), \ell)$ denotes the distance between the partite set ℓ and the partite sets containing the element i.

These examples demonstrate the flexibility that is available to us in devising criteria for multipartite graph clustering.

3.2 Monotone Functions

A monotone function is the one that preserves a given order. The order relationship is usually defined as either increasing or decreasing. Generally, these functions take a single argument and the monotone relationship (increasing, or decreasing) is between

---

our application. And, this is the main reason why we have analyzed it in detail in chapter 3. Another more general usefulness of the objective function considered in chapter 3 is that it shows a connection of the optimization problem with classical graph theoretical problems (by using an analogy with the function considered by Matula [52]). Here we focus on a general optimization problem from a set theoretic perspective.

the order of the argument and the function values. Here, we look at slightly different more general linkage functions - the linkage functions, $\pi(i, H)$. These functions take two arguments, the first of which is an element i from a subset H of the set V, and the second is the subset H itself.

We consider linkage functions which take two arguments: the first is an element while the second is a subset. Furthermore, while considering monotonicity, the first argument remains fixed and the monotonicity relationship is between the second argument (the subset) and the value of the function. Depending on the kind of order preserved between the argument and the function value, there are two kinds of monotone functions - the monotonic increasing and the monotonic decreasing functions. As the definitions of these two types of functions are analogous, for simplicity, we will restrict the current presentation to monotonic increasing functions and describe the monotone decreasing functions later.

Definition 1. *A linkage function,* $\pi : (V \setminus \emptyset) \times (2^V \setminus \emptyset) \rightarrow \mathbb{R}$*, is monotone increasing if it satisfies the inequality:*

$$\pi(i, H) \geq \pi(i, H') \quad \forall i, \forall H', \forall H \ : \ i \in H' \subseteq H \subseteq V \tag{3.5}$$

A pictorial illustration of the definition of monotonic increasing linkage (set) function is given in Fig. 3.1.

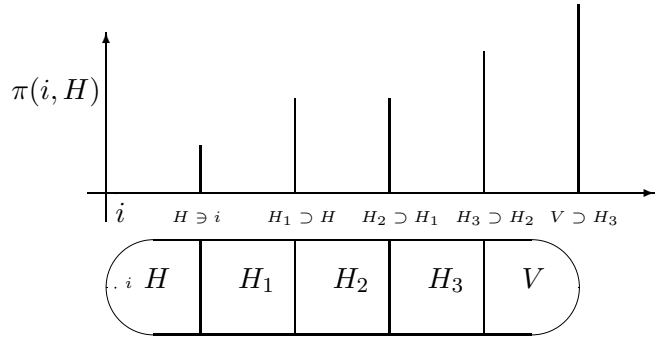

Figure 3.1: Illustration for the monotone increasing property of the linkage function. When considering the monotonicity of the linkage function, the element i remains fixed, the value of the function $\pi(i, H)$ increases as the second argument H increases by including more elements.

A consequence of the monotonicity property, $\pi(i, H) \geq \pi(i, H') \, \forall H' \subset H$, is that the element i which has the minimum value for $\pi(i, V)$ can be removed. This is because if such an element is kept while some other element is removed, the score value of the subset cannot increase according to the monotonicity property. On the other hand, if the element with the minimum value is removed, there is a possibility that the resulting subset will have a larger score. This argument also holds for the subset obtained by removing the worst element and naturally leads to an algorithm. It turns out this greedy procedure of iteratively removing the worst element guarantees that the optimal solution is the highest scoring subset found while the elements are sequentially removed. The stopping criterion for this "shelling process" is either that all the elements are exhausted, or that no element remains that has a linkage function value larger than the score value for a subset already encountered. Furthermore, this sequential shelling-out of the worst element can be used to estimate the structure of the relationships in the input graph, as we will see later.

3.3 Quasi-concave set functions

We now study some properties of the function $F(H)$ (3.1)and describe its relationship to the monotone increasing functions.

Definition 2. *A set function, $F : 2^V \setminus \emptyset \rightarrow \mathbb{R}$, is quasiconcave if it satisfies*

$$F(H' \cup H'') \geq \min(F(H'), F(H'')) \qquad \forall H', H'' \subseteq V \tag{3.6}$$

The following relationship between a monotone increasing linkage function and a quasiconcave set function is the basis for an efficient procedure.

Proposition 1. *[53] The set function $F(H)$ as defined in (3.1) is quasiconcave if and only if the corresponding linkage function, $\pi(i, H)$ is monotone increasing.*

Proof. [$\Rightarrow$] Let $H', H'' \subseteq V$, and $i^* \in H' \cup H''$ be the element such that $F(H' \cup H'') = \pi(i^*, H' \cup H'')$. Suppose, $i^* \in H'$, then using (3.5) we get, $F(H' \cup H'') = \pi(i^*, H' \cup H'') \geq$

$\pi(i^*, H') \geq \min_{i \in H'} \pi(i, H') = F(H') \geq \min(F(H'), F(H''))$.

[$\Leftarrow$] The proof is by contradiction. For $i \in H' \subseteq H \subseteq V$, assume that $\pi(i, H' \cup H) < \pi(i, H')$ and (3.6) hold. From the assumption we get $\min_{i \in H'} \pi(i, H' \cup H) < \min_{i \in H'} \pi(i, H') = F(H')$. Further, $F(H' \cup H) = \min_{i \in H' \cup H} \pi(i, H' \cup H) \leq \min_{i \in H'} \pi(i, H' \cup H)$. Combining these two inequalities we get, $F(H' \cup H) < F(H')$ which contradicts the quasiconcavity property (3.6) in the assumption. □

Definition 3. *A maximizer for the set function $F(H)$ is a subset $H^* \subseteq V$ satisfying the following condition:*

$$F(H^*) \geq F(H) \quad \forall H \subseteq V \tag{3.7}$$

Proposition 2. *[53] For a quasiconcave set function $F(H)$ the set of all its maximizers, as defined by (3.1), is closed under the set union operation.*

Proof. This follows from the quasiconcavity of $F(H)$. We must prove that for arbitrary maximizers, H_1 and H_2 ($F(H') = F(H'') = \max_{H \in 2^V \setminus \emptyset} F(H)$), their union, $H' \cup H''$ will also be a maximizer. The assumption that H' and H'' are maximizers means that $F(H') = F(H'') \geq F(H)\, \forall H \subseteq V$. Let $H = H' \cup H''$, then from the assumption we have, $F(H') = F(H'') \geq F(H' \cup H'')$. As a consequence of the quasiconcavity property of $F(H)$ we get $F(H' \cup H'') \geq \min(F(H'), F(H''))$. Since $F(H') = F(H'')$, we can write $F(H') = \min(F(H'), F(H''))$. So, the following sequence of inequalities follows:

$$F(H') \geq F(H' \cup H'') \geq F(H').$$

In other words, the set $H' \cup H''$ is also a maximizer. The proof also extends to more than two maximizers, as we can use induction on the number of maximizers and progressively include all the maximizers. □

Definition 4. *A maximizer of $F(H)$ that contains all other maximizers is called the $\cup$-maximizer, $\hat{H}$.*

It is obvious from Proposition 2 that $\hat{H}$ is the largest maximizer; must contain all the other maximizers and hence it is unique.

Algorithm for finding the $\cup$-maximizer: The pseudocode for the algorithm is presented in Table 3.1. The Step 0 of this algorithm is an initialization of variables. This is an iterative algorithm and Steps 1 and 2 constitute an iteration. At the first iteration, Step 1 begins by calculating $F(V)$ and the set M^1 containing the subset of vertices that satisfy $F(V) = \pi(i, V)$, i.e., $M^1 = \{i \in V : \pi(i, V) = F(V)\}$. In Step 2, the vertices in the set M^1 are removed from V to get $H^2 = V \setminus M^1$. At the iteration t, it considers the set H^{t-1} as input, calculates $F(H^{t-1})$, identifies the subset M^t such that $F(H^{t-1}) = \pi(i_t, H^{t-1}), \forall i_t \in M^t$, and removes this subset from H^{t-1} to produce $H^t = H^{t-1} \setminus M^t$. Thus, we get a nested family of subsets, $\mathcal{H} = \{H^1, H^2 \ldots, H^T\}$, of V such that $H^T \subset H^{T-1} \subset \ldots \subset H^2 \subset H^1 = V$, where, $H^t \setminus H^{t+1} = M^t = \{i : \pi(i, H^t) = \min_{j \in H^t} \pi(j, H^t)\}$. Along with this, we also get another nested family of subsets $\mathcal{G} = \{\Gamma_1, \Gamma_2, \ldots, \Gamma_p\}$ such that $\Gamma_p \subset \Gamma_{p-1} \subset \ldots \subset \Gamma_2 \subset \Gamma_1 = V$. By construction, the $F(\Gamma_j)$ values for the members of nested family of subsets satisfy: $F(\Gamma_p) > F(\Gamma_{p-1} > \ldots > F(\Gamma_2) > F(\Gamma_1)$. The algorithm terminates at the iteration T when $H^T = \emptyset$ or $F(H^T) = 0$. It outputs $\hat{H}$ as the subset, H^j with the smallest j such that $F(H^j) \geq F(H^s)\, \forall s \in \{1, 2, \ldots, T\}$.

Step 0: Set $t := 1$; $p := 1$; $H^1 := V$; $\Gamma_1 := V$;
Step 1: Find $M^t := \{i : \pi(i, H^t) = \min_{j \in H^t} \pi(j, H^t)\}$;
Step 2: if $((H^t \setminus M^t = \emptyset) \vee (\pi(i, H^t) = 0\ \forall i \in H^t))$ STOP.
else $\{\ H^{t+1} := H^t \setminus M^t;\ t := t + 1;\}$
if $(F(H^t) > F(\Gamma_p))$ $\{p = p + 1;\ \Gamma_p = H^t;\}$
go to Step 1.

Table 3.1: Pseudocode for extracting $\hat{H}$.

F-spectrum: In the process of finding the maximizer for $F(H)$, the algorithm in Table 3.1, builds a nested chain of subsets H^j and also computes the corresponding values $F(H^j)$. As the chain of subsets is nested, they can be ordered and shown on a line where a subset on the left contains the ones shown to its right on the line. We will call the graphical plot of the relations between the subsets H^j and their corresponding

F values the F-spectrum. Figure 3.2 illustrates this spectrum corresponding to a hypothetical instance, where the algorithm begins with $H^1 = \Gamma_1 = V$ and elements in the subset M^1 are removed from H^1 yielding the set H^2 such that $F(H^2) < F(H^1)$, then elements in the subsets M^2 and M^3 are removed, sequentially, producing the subset H^4 such that $F(H^4) > F(\Gamma_1)$, so $\Gamma_2 = H^4$, and so on. The extremal value of $F(H)$ is encountered twice in $\mathcal{H}$. It must be emphasized that the solution, Γ_p, is equal to the H^j with the least index j in the sequence, $\mathcal{H}$. In other words, the solution Γ_p is the largest cardinality element, H^j in $\mathcal{H}$ that satisfies $F(H^j) \geq F(H^i),\ \forall H^i \in \mathcal{H}$. Furthermore, by construction we have: $F(\Gamma_p) > F(\Gamma_{p-1} > \ldots > F(\Gamma_1 = V)$. And also, $\forall H \subset V, \Gamma_i \in \mathcal{G}$: $H \setminus \Gamma_i \neq \emptyset \Rightarrow F(H) < F(\Gamma_i)$. Finally, in the case of this example, the algorithm terminates with values $T = 25$ and $\Gamma_p = H_{16}$.

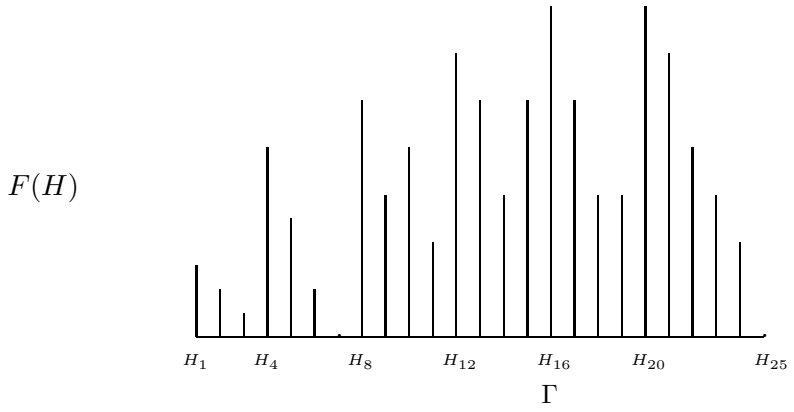

Figure 3.2: Spectrum for F-values

REMARK: Although we are only interested in finding the subset with the highest score value, the trace of the intermediate steps of the algorithm provides information about the structure of relationships in the input data. The change in the score value $F(H)$ as the elements are shelled out from the current set provides a view of the structure of the data, as shown by the spectrum. More precisely, the score value $F(H)$ suggests that elements in H are related to each other at least by the score value.

This algorithm is similar to the one described in chapter 3 for finding the multipartite quasi-clique, albeit in a more general framework.

The $\cup$-maximizer obtained above has the property that it is a unique subset isolated from the rest of the set in such a way that adding or removing elements from it affects its F-value adversely. This property allow us to interpret the $\cup$-maximizer as a cluster. Thus, according to the similarity notion defined by the monotone linkage function $\pi(i, H)$ the $\cup$-maximizer is a structurally distinct (homogeneous and dense) subset. This property of the cluster is summarized by the following theorem.

Theorem 6. *The subset Γ_p output by the above algorithm is the $\cup$-maximizer for F in the set V.*

Proof. According to the algorithm, $F(\Gamma_p) = \max_{H^i \in \mathcal{H}} F(H^i)$, where $\mathcal{H} = \{H^1, H^2, \ldots, H^T\}$ and $H^T \subset \ldots \subset H^2 \subset H^1 = V$. The proof for $F(\Gamma_p) \geq F(H)\ \ \forall H \subseteq V$ is divided into two cases: (i) $H \setminus \Gamma_p \neq \emptyset$ and, (ii) $H \subseteq \Gamma_p$.

Case (i) $[H \setminus \Gamma_p \neq \emptyset]$: Consider an arbitrary set H such that $H \setminus \Gamma_p \neq \emptyset$ and let H^i be the smallest set in the sequence $\mathcal{H}$ containing H, so that $H \subseteq H^i$ but $H \nsubseteq H^{i+1}$. Since $M^i = H^i \setminus H^{i+1}$, there is at least one element, say i_H, common to both M^i and H. By definition of $F(H^i)$, we have

$$F(H^i) = \min_{i \in H^i} \pi(i, H^i) = \pi(i^*, H^i) \tag{3.8}$$

By construction of M^i, $i^* \in M^i$, so $\pi(i^*, H^i) = \pi(i_H, H^i)$. Using (3.5) we get

$$\pi(i^*, H^i) = \pi(i_H, H^i) \geq \pi(i_H, H) \geq \min_{i \in H} \pi(i, H) = F(H) \tag{3.9}$$

Using (3.8) and (3.9) we obtain $F(H^i) \geq F(H)$. According to the algorithm $F(\Gamma_p) > F(H^i)$, $\forall H^i \supset \Gamma_p$, so we prove

$$F(\Gamma_p) > F(H),\ \forall H \setminus \Gamma_p \neq \emptyset \tag{3.10}$$

Case (ii) $[H \subseteq \Gamma_p]$: Similar to the previous case, there exists a smallest subset H^i in the sequence $\mathcal{H}$ that includes H. So, the inequalities in (3.9) hold here too, and we can

write $F(H^i) \geq F(H)$. On the other hand, $F(\Gamma_p) \geq F(H^i)$, which in conjunction with the previous inequality implies

$$F(\Gamma_p) \geq F(H),\ \forall H \subseteq \Gamma_p \tag{3.11}$$

Further, a maximizer satisfying inequalities (3.10) and (3.11) is the $\cup$-maximizer. □

Heuristics for speeding up the algorithm in Table 3.1: The algorithm in Table 3.1 removes the vertices corresponding to the minimum of the linkage function. If, however, we could remove a larger set of vertices without affecting the correctness of the procedure, the algorithm would be faster, as more vertices would be removed at each iteration. The following theorem determines such elements.

Theorem 7. *Define* $Q = \{i \in V\ :\ \pi(i, V) < \theta,\ \theta > 0\}$ *and let* $H^*_{V\setminus Q}$ *be the optimal solution in the set* $V \setminus Q$*, and* H^* *be the optimal solution in the set* V*. Then,*

$$\pi(i_{H^*_{V\setminus Q}}, H^*_{V\setminus Q}) > \theta \Rightarrow\ H^*_{V\setminus Q} = H^*. \tag{3.12}$$

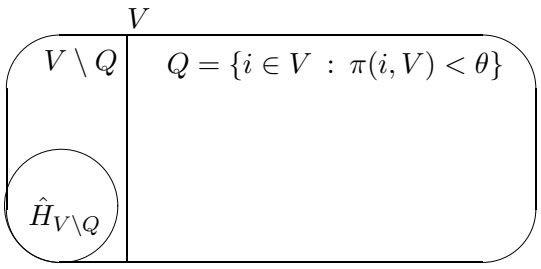

Figure 3.3: Implication of the theorem: it guarantees that the optimal solution found in the smaller set $V \setminus Q$ is exactly the same as the one that would be found in V, and thus saves computational time.

Proof. We first prove that $\pi(i_{H^*_{V\setminus Q}}, H^*_{V\setminus Q}) > \theta \Rightarrow \hat{H} \subseteq V \setminus Q$. The score-value for $H^*_{V\setminus Q}$, obtained from $V \setminus Q$ (a subset of V) can be at most the score value of H^* obtained from V, i.e., $\pi(i_{H^*}, H^*) \geq \pi(i_{H^*_{V\setminus Q}}, H^*_{V\setminus Q}) > \theta$. Then according to the definition of the score, it follows that $\forall i \in H^*, \pi(i, H^*) > \theta$. As discussed before, the function $\pi(i, H)$ satisfies the property (3.5) that its value for any fixed element i increases when more elements are added to H. Using this property, we get $\pi(i, V) \geq \pi(i, H^*) > \theta$. But, according

to the definition of Q, $\pi(i, V) > \theta \Rightarrow i \in V \setminus Q$. This proves that $\forall i \in H^*, i \in V \setminus Q$, in other words, $H \subseteq V \setminus Q$. But, by the optimality of $H^*_{V\setminus Q}$ in $V \setminus Q$, we have $\pi(i_{H^*_{V\setminus Q}}, H^*_{V\setminus Q}) \geq \pi(i_H, H)\ \ \forall H \subseteq V \setminus Q$ i.e., $\pi(i_{H^*_{V\setminus Q}}, H^*_{V\setminus Q}) \geq \pi(i_{H^*}, H^*)$. But we already had $\pi(i_{H^*}, H^*) \geq \pi(i_{H^*_{V\setminus Q}}, H^*_{V\setminus Q})$, so the two values must be equal which implies $H^*_{V\setminus Q} = H^*$. □

According to Theorem 7, we can remove all vertices whose linkage function values are less than the current estimate of the score value of the optimal solution. In the algorithm in Table 3.1, $\pi(i_\Gamma, \Gamma)$ is the current estimate of the score value for the optimal solution, so we can remove all vertices, $M_t = \{i : \pi(i, H_t) < \pi(i_\Gamma, \Gamma)\} \cup \{i : \pi(i, H_t) = \min_{j\in H_t} \pi(j, H_t)\}$. This reduces the number of iterations, as more vertices are likely to be removed at each iteration. Unfortunately, there are many situations where this modification does not lead to improvements.

A modified procedure incorporating this expedited removal of vertices is given below. This algorithm is a slight modification of the one described in Table 3.2. One of the key differences is that it takes an extra optional parameter θ, which is the estimate for the optimal solution. If this parameter is not part of the input, it is set to 0, implying that we have no estimate for the optimal value. The rest of the algorithm is essentially the same except for the statements (1), (2) (in Table 3.2) and the while loop marked as (3). Statement (1) is a direct consequence of applying theorem 7 (as discussed above) where we remove all elements whose score values is less than the current estimate of the score for the optimal solution. Statement (2) and the while loop statements beginning at (3) are an extensions of the previous statement and remove all those elements whose linkage function values have reduced lower than the current optimal value estimate as a result of removing elements at statement (1).

Like the nested family of subsets $\mathcal{H}$ obtained using the algorithm in Table 3.1, here we obtain another family of nested subsets $\Gamma = \{\Gamma_1, \Gamma_2, \ldots, \Gamma_p\}$ of V where $\Gamma_p \subset \Gamma_{p-1} \subset \ldots \Gamma_2 \subset \Gamma_1 = V$. Members of this family, by construction obey a strict ordering according to function values of the subsets, namely, $F(\Gamma_1) < F(\Gamma_2) < \ldots < F(\Gamma_p)$.

Algorithm 3.3.1: MODIFIED-MAXIMIZATION-PROCEDURE(V, θ)

$\imath \leftarrow 0$; $\Gamma_\imath \leftarrow V$; $F(\Gamma_\imath) \leftarrow \min_{\alpha \in \Gamma_\imath} \pi(\alpha, \Gamma_\imath)$;
$u_\imath \leftarrow \max(F(\Gamma_\imath), \theta)$;
while $(\Gamma_\imath \neq \emptyset)$
do {
  $M_\imath \leftarrow \{\alpha \in \Gamma_\imath : \pi(\alpha, \Gamma_\imath) \leq u_\imath\}$; (1)
  $S \leftarrow \{\alpha \in \Gamma_\imath \setminus M_\imath : \pi(\alpha, \Gamma_\imath \setminus M_\imath) \leq u_\imath\}$; (2)
  while $(S \neq \emptyset)$
  do {
    $M_\imath \leftarrow M_\imath \cup S$; (3)
    $S \leftarrow \{\alpha \in \Gamma_\imath \setminus M_\imath : \pi(\alpha, \Gamma_\imath \setminus M_\imath) \leq u_\imath\}$;
  }
  $\Gamma_{\imath+1} \leftarrow \Gamma_\imath \setminus M_\imath$;
  $\imath \leftarrow \imath + 1$;
  $u_\imath \leftarrow F(\Gamma_\imath) \leftarrow \min_{\alpha \in \Gamma_\imath} \pi(\alpha, \Gamma_\imath)$;
}
OUTPUT/RETURN $\Gamma_{\imath-1}$ as the optimal set and
$F(\Gamma_{\imath-1})$ as the optimal value.

Table 3.2: Modified algorithm for finding the maxima of the quasiconcave function.

3.4 Analysis and Implementation

As described in the previous chapter, a critical step at each iteration is to compute the value of the linkage function for every element of the current subset. Thus linkage functions which can be updated efficiently must be preferred, for instance, an additive linkage function enables an efficient update by modifying the current linkage function value.

The quasiconcavity of the objective function, induced by the monotonicity of the linkage function, only guarantees that the $\cup$-maximizer can be computed efficiently provided the linkage function is efficiently computable. Therefore, the overall efficiency of the algorithm rests on the design of the linkage function. However, there are some data structure choices that lead of efficient implementation, regardless of the choice of the linkage function.

For a linkage function which can be decomposed such that contribution from each edge is separable, the specific implementation choices discussed in the previous chapter hold equally well here. However, for a general monotone increasing linkage function, the time complexity depends on the particular linkage function. Assuming that initial

linkage function values for all the elements can be computed in time τ_i and all updates together take time τ_u, a straightforward implementation of the algorithm will take $O(\tau_i + \tau_u + |V|^2)$ time. Indeed, during the initialization of the algorithm, we must to compute the linkage function for all vertices which (by assumption) takes time τ_1. At subsequent iterations, each vertex is deleted at most once; the vertex deletion entails updates to the linkage function values. All updates together take τ_u. In step 1 of the algorithm, we need to find the vertices with the minimum value of the linkage function, and this can be done by looking at the vertices active at that iteration. Since the total number of vertices at any iteration i is at most $|V|-i$ and the total number of iterations is at most $|V|$ (achieved when a single vertex is deleted at each iteration), the total time required for step 1 is $O(\tau_i + \tau_u + |V|^2)$. The implementation where we need to store only the graph takes space $O(|E| + |V|)$.

In the previous chapter, the run time of the algorithm for a specific linkage function was analyzed in detail. Here, although the linkage function is not exactly specified, the properties of the linkage function which permitted an efficient implementation are preserved. For instance the linkage function's values for elements can only decrease as the algorithm progresses to a new iteration. As a consequence, the salient details of the algorithm, except the linkage function value computation and updates to it, are still valid and we can use the Fibonacci heaps and Bucket sort algorithm for efficiently performing the elementary operations (`find-min`, `delete-min` and `decrease-key` as defined earlier) at each iteration. These choices, as in chapter 3, result in an efficient implementation of the algorithm.

As shown in the previous chapter, using Fibonacci heaps for storing the linkage function values, leads to an implementation that takes time $O(\tau_i + \tau_u + |V| \log |V|)$. The details of implementation are exactly the same, since changing a linkage function affects only the steps where linkage function values are computed or updated. As the linkage function computations are independent of data structure, the time spent in updating the positions of elements in a data structure are not affected. Consequently, using the bucket sort algorithm makes the algorithm run in time $O(\tau_i+\tau_u+|V| \log |V|)$.

3.5 Clustering using quasiconcave functions

The algorithms described in Tables 3.1 and 3.2 can be viewed as a means of exploring the structure of relationships between elements in V. This structure is already visualizable by the F-spectrum, and here we want to show significant components of the spectrum, which motivated the authors [54] to refer to the whole structure as a layered clustering structure.

The optimal solution we obtain is the subset which has the highest Γ_p value of the score function $F(\Gamma_p)$, but on the way to find this solution, the algorithm also determines a nested family of subsets $\Gamma = \{\Gamma_1, \Gamma_2, \ldots, \Gamma_p\}$ such that $F(\Gamma_1) < F(\Gamma_2) < \ldots < F(\Gamma_p)$. Furthermore, as indicated earlier while discussing the F-spectrum, we have the following relationship involving F-values of these nested subsets: for all j, if $H \setminus \Gamma_j \neq \emptyset$, then $F(\Gamma_j) > F(H)$. So, Γ_j can be interpreted as a quasi-cluster. Each of these subsets have high scores and can thus be interpreted as revealing a stratification of elements in the set V. From this perspective, these nested sets can be interpreted as layered clusters of the set V, with Γ_p being at the most dense cluster at the center of the layered structure, followed by Γ_{p-1}, and so on. It must be remarked that the nested cluster structure discovered by the algorithm does not induce a partition on the set V, which many applications require. It is more natural to interpret the above clustering as a "tower cluster" presentation for the whose set V.

Partitioning V using the procedures for finding the $\cup$-maximizer: The elements in the $\cup$-maximizer $\hat{H}$ are very similar to each other whereas those in the subset $V \setminus \hat{H}$ are likely to contain many other clusters. So, like the procedure in chapter 3, we can devise a sequential procedure for finding all the clusters by iteratively finding one cluster in the remaining set using any of the procedures described in Table 3.1, or Table 3.2. The procedure for doing this has already been described in Table 3.3, which differs from the procedure in Table 2.3 only in Step 1.

Like the procedure in Table 2.3, this also determines an ordered set of clusters, $C = \{\hat{H}_0, \hat{H}_1, \ldots, \hat{H}_m\}$ and the set of residual elements $R = \{i : i \in V \setminus C\}$ which are "far" from other elements V. The clusters in set C are ordered according to the

Initialization: $V^0 := V;\ m := 0;\ C = \emptyset;$
Step 1: Extract H_m^* from V^m using procedure in Table 3.2; Add H_m^* to C;
Step 2: $V^{m+1} := V^m \setminus H_m^*;\ m := m + 1;$
Step 3: if ($(V^m = \emptyset) \wedge (\pi(i, H) = 0\ \forall i \in H \subseteq V^m)$)
Output C, V^m as R, and m; STOP;
else go to step 1

Table 3.3: Pseudocode for finding a series of multipartite clusters.

relationship $F(\hat{H}_0) > F(\hat{H}_1) > \ldots > F(\hat{H}_m)$.

3.6 Quasiconvex set functions

In chapter 3 and all previous sections in this chapter, we described quasiconcave set functions and an iterative algorithm for finding their optimal (maximizer) solution. The iterative algorithm shelled out the "worst" vertices from the graph and determined the objective function values for the subsets as they were encountered in the shelling-out procedure. From this perspective, the process can be interpreted as an exploration of the structure of a similarity graph representing the relationships between objects in the data. In particular, each iteration can be interpreted as peeling out layers of objects that are isolated from the rest of the objects, so one can call it a procedure for determining a single-layered cluster in the data [54]. This layered structure is determined by a single score function (the quasiconcave function being maximized), and it provides a 1-dimensional visual presentation of the data.

In this section, we describe the quasiconvex functions whose minimization determine a complementary structure for the same data. These functions may be interpreted as complements of the quasiconcave functions. The properties of these functions are analogous to those of quasiconcave functions and an algorithm similar to the one described above can be used to find their optimal solutions. The properties of these two kinds of functions are "opposite": whereas quasiconcave functions are defined using monotonically increasing linkage functions, quasiconvex functions are based on monotonically decreasing functions. The optimization problem for quasiconcave functions is maximization but that for the quasiconvex functions is minimization. Due to these contrasting

properties, it is convenient to think of quasiconvex functions complementary to the quasiconcave functions, and a pleasing consequence of this is the development of an efficient procedure for finding the optimal solution for both systems [2]. We now define these basic concepts for the quasiconvex functions.

Definition 5. *A linkage function* $\pi_d : V \times (2^V \setminus \emptyset) \rightarrow \mathbb{R}$ *is a monotonically decreasing function if*

$$\pi_d(i, H) \geq \pi_d(i, H') \quad \forall i \in H \subseteq H' \subseteq V \tag{3.13}$$

From the definition, if $\pi(i, H)$ is a monotone increasing function, $-\pi(i, H)$ is a monotone decreasing function. Another example is $\pi_d^{'}(i, H) = \pi(i, V \setminus H)$, where $\pi(i, H)$ is a monotonically increasing function. It is also possible for a function to be monotone increasing and decreasing simultaneously, for instance the constant function. Of course, there are many interesting examples of monotone decreasing functions, such as

$$\pi_{d_1}(i, H) = \sum_{\substack{\ell=1 \\ \ell \neq g(i)}}^{k} \sum_{j \notin H_\ell} w_{ij} + \left(\sum_{V_\ell \cap H_\ell = \emptyset} 1 \right)^{\alpha}, \text{ where } \alpha \geq 0 \tag{3.14}$$

$$\pi_{d_2}(i, H) = \sum_{\substack{\ell=1 \\ \ell \neq g(i)}}^{k} (\rho_{g(i)\ell})^{\alpha} \left(\sum_{j \notin H_\ell} w_{ij} \right)^{\beta}, \text{ where } \alpha, \beta \geq 0 \tag{3.15}$$

$$\pi_{d_3}(i, H) = \sum_{\substack{\ell=1 \\ \ell \neq g(i)}}^{k} (\rho_{g(i)\ell})^{\alpha} \left(\sum_{m \in V_\ell \setminus H_\ell} w_{im} - \sum_{j \in H_\ell} w_{ij} \right)^{\beta} \text{ where } \alpha, \beta \geq 0 \tag{3.16}$$

where w_{ij} is the weighted similarity between the elements i and j, H_ℓ is a subset of elements from the partite set V_ℓ, and $\rho_{g(i)\ell}$ is the distance between the partite set ℓ and the partite set $g(i)$ containing the element i. Clearly, these linkage functions are analogous to the monotone increasing linkage functions used for defining the quasiconcave set functions $F_1(H)$ through $F_3(H)$ in equations (4.2) through (4.4).

[2] This idea of applying both types of functions for speeding up the algorithms for finding the $\cup$-maximizer was described first in [56]. Our description provides more details.

Analogous to the definition of $F(H)$, we define $F_d(H)$ as

$$F_d(H) = \max_{i \in H} \pi_d(i, H) \quad \forall i \in H \;\; \forall H \subseteq V \tag{3.17}$$

Definition 6. *A set function $F_d : 2^V \rightarrow \mathbb{R}$ is quasiconvex if*

$$F_d(H' \cup H'') \leq \max(F_d(H'), F_d(H'')) \quad \forall H', H'' \subseteq V \tag{3.18}$$

And, the corresponding optimization problem seeks to minimize the function $F_d(H)$:

$$H_d^* = \min_{H \subseteq V} F_d(H) \tag{3.19}$$

All the properties for the quasiconcave set functions have their analogs in the quasiconvex set functions. In particular, the Propositions 3 and 4, given below, are analogous to Proposition 1 and 2, respectively, in section 4.3 .

Proposition 3. *The set function $F_d(H)$ (3.17) is quasiconvex if and only if the linkage function $\pi_d(i, H)$ is monotone decreasing.*

Proof. [$\Rightarrow$] Let $H^{'}, H^{''} \subseteq V$, and $\hat{i} \in H^{'} \cup H^{''}$ be the element such that $F_d(H^{'} \cup H^{''}) = \pi_d(\hat{i}, H^{'} \cup H^{''}) = \max_{i \in H^{'} \cup H^{''}} \pi_d(i, H^{'} \cup H^{''})$. Without loss of generality assume, $\hat{i} \in H^{'}$, then using the monotone decreasing property of the linkage function we get the following sequence of inequalities which proves this part.

$$\begin{aligned} F_d(H^{'} \cup H^{''}) &= \max_{i \in H^{'} \cup H^{''}} \pi_d(i, H^{'} \cup H^{''}) = \pi_d(\hat{i}, H^{'} \cup H^{''}) \\ &\leq \pi_d(\hat{i}, H^{'}) \leq \max_{i \in H^{'}} \pi_d(i, H^{'}) = F_d(H^{'}) \\ &\leq \max(F_d(H^{'}), F_d(H^{''}) \end{aligned}$$

[$\Leftarrow$] We prove this by contradiction. So, assume that $F_d(H)$ is quasiconvex i.e., $F_d(H^{'} \cup H^{''}) \leq \max(F_d(H^{'}), F_d(H^{''}))$ but the linkage function is not monotone decreasing, i.e., $\pi_d(i, H^{'} \cup H^{''}) > \pi_d(i, H^{'})$, where $i \in H^{'}$ and $H^{'}, H^{''} \subseteq V$. Let $\hat{i}$ be the element in $H^{'} \cup H^{''}$ such that $F_d(H^{'}, H^{''}) = \pi_d(\hat{i}, H^{'} \cup H^{''})$. Since $\hat{i}$ must belong to at least one of

the subsets, without loss of generality, we can assume $\hat{i} \in H^{'}$, then we get the following:

$$\begin{aligned} F_d(H^{'} \cup H^{''}) &= \max_{i \in H^{'} \cup H^{''}} \pi_d(i, H^{'} \cup H^{''}) \\ &= \max_{i \in H^{'}} \pi_d(\hat{i}, H^{'} \cup H^{''}) \\ &> \max_{i \in H^{'}} \pi_d(\hat{i}, H^{'}) = F_d(H^{'}) \end{aligned}$$

This implies $F_d(H^{'} \cup H^{''}) > F_d(H^{'}$. According to quasiconvexity we have $F_d(H^{'} \cup H^{''}) \leq F_d(H^{'})$, which is a contradiction, so it must be that $\pi_d(i, H^{'} \cup H^{''}) \leq \pi_d(i, H^{'})$. □

Proposition 4. *The set of solutions for the optimization problem in (3.19) is closed under the set union operation.*

Proof. Let $H^{'}$ and $H^{''}$ be the two different optimal solutions for $\underset{H \subseteq V}{\operatorname{argmin}} F_d(H)$, then we must prove that $H^{'} \cup H^{''}$ is also an optimal solution. From the assumption we obtain $F_d(H^{'}) = F_d(H^{''}) \leq F_d(H) \quad \forall H \subseteq V$, so

$$F_d(H^{'}) = F_d(H^{''}) \leq F_d(H^{'} \cup H^{''})$$

But, from the quasiconvexity property we obtain,

$$F_d(H^{'} \cup H^{''}) \leq \max(F_d(H^{'}), F_d(H^{''})) = F(H^{'}) = F_d(H^{''})$$

Combining the above two sequences of inequalities we obtain $F_d(H^{'}) \leq F_d(H^{'} \cup H^{''}) \leq F_d(H^{'})$, which proves that the union of the two optimal solutions is also an optimal solution. Using induction, we can prove that the union of all the optimal solutions is also an optimal solution. Furthermore, the union of all the solutions is the unique, largest optimal solution. □

The algorithm for finding the optimal solution to (3.19) is similar to the algorithm in Table 3.2 and is described below. Like the algorithm in Table 3.2, this is a shelling algorithm: it successively shells-out the "worst" elements but unlike the previous algorithm, these elements are the ones with the highest value of the linkage function

Algorithm 3.6.1: ALGORITHM FOR FINDING $\cup$-MINIMIZER $H_d^*\ (V, \delta)$

$\jmath \leftarrow 0$; $L_\jmath \leftarrow V$; $F_d(L_\jmath) \leftarrow \max_{\beta \in L_\jmath} \pi_d(\beta, L_\jmath)$;
$v_\jmath \leftarrow \min(F_d(L_\jmath), \delta)$;
while $(L_\jmath \neq \emptyset)$
do
- $W_\jmath \leftarrow \{\beta \in L_\jmath : \pi_d(\beta, L_\jmath) \geq v_\jmath\}$;
- $T \leftarrow \{\beta \in L_\jmath \setminus W_\jmath : \pi_d(\beta, L_\jmath \setminus W_\jmath) \geq v_\jmath\}$;
- **while** $(T \neq \emptyset)$
  - **do**
    - $W_\jmath \leftarrow W_\jmath \cup T$;
    - $T \leftarrow \{\beta \in L_\jmath \setminus W_\jmath : \pi_d(\beta, L_\jmath \setminus W_\jmath) \geq v_\jmath\}$;
- $L_{\jmath+1} \leftarrow L_\jmath \setminus W_\jmath$;
- $\jmath \leftarrow \jmath + 1$;
- $v_\jmath \leftarrow F_d(L_\jmath) \leftarrow \max_{\beta \in L_\jmath} \pi_d(\beta, L_\jmath)$;

OUTPUT $L_\jmath$ as the optimal subset;

Table 3.4: Algorithm for finding the minima of the quasiconvex function.

$\pi_d(i, H)$. As the elements are iteratively shelled-out from the set V, the distribution of score values of the intermediate subsets $F_d(H^j)$ is called the spectrum of F_d-values.

It must be emphasized that the steps of this algorithm are exactly the same as those of the previous algorithm (except that maximum is replaced by minimum and vice-versa). Along with producing the minima of the function $F_d(H)$, it also produces a nested family of subsets $\mathcal{L} = \{L_1, \ldots, L_q\}$ where $L_q \subset L_{q-1} \subset \ldots \subset L_1 = V$ and $F_d(L_q) < F_d(L_{q-1}) < \ldots < F_d(L_1)$. The complexity analysis and the choice of data structure for the implementation of the previous algorithm applies to this algorithm as well.

3.7 Combining algorithms for optimizing quasiconcave and quasiconvex set functions for designing an improved algorithm for finding their solutions simultaneously

As mentioned at the beginning of previous section, the idea of simultaneous analysis of quasiconcave and quasiconcave function was described by Mullat [56]. In this section, we give a description of this idea in more detail and from a different perspective, specifically

as a method for visualizing the structure of the data.

We have seen that the properties and algorithm for minimizing a quasiconvex set function are very similar to those for the quasiconcave set functions. The key difference between their optimization procedures is that the maximization procedure for quasiconcave functions shells out the elements with the least value of the corresponding linkage function value, while in the procedure for minimizing a quasiconvex set functions we shell out the elements with the largest value of the corresponding linkage function value. Up to now, we have described these optimization problems on different underlying graphs. The interplay between these problems is far more interesting when the underlying graph is same. Clearly, in such a case these two optimization problems seek to uncover complementary substructures in the same graph, but what is intriguing are the relationships between their intermediate steps in finding their respective solutions (substructures). These phenomena are better explained with the help of the following example.

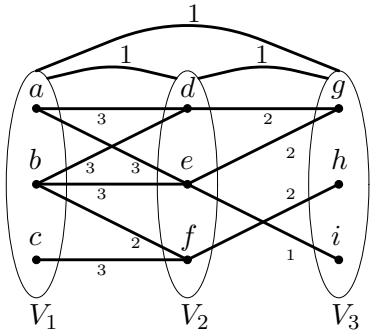

Figure 3.4: An example 3-partite graph for illustrating the structure of the graph explored by the quasiconcave and quasiconvex functions. V_1, V_2 and V_3 are the three partite sets with the curved arcs representing relationships between them, while the straight edges represent the weighted relationships between the elements across the partite sets.

Consider the example multipartite graph shown in Figure 3.4. The partitions of this 3-partite graph are $V_1 = \{a, b, c\}, V_2 = \{d, e, f\}$ and $V_3 = \{g, h, i\}$; the similarity relationships between vertices across the partite sets are represented using the straight edges and the weight on each edge describes the strength of similarities between the vertices incident with the edge. The curved edges represent with relationships between the partite sets - for simplicity they all have been assigned the weight 1. To be able to

study the behavior of maximization and minimization procedures on this example, we define the monotonic increasing linkage function, $\pi(i, H)$ and the monotonic decreasing function, $\pi_d(i, H)$ as:

$$\pi(i, H) = \sum_{\substack{\ell=1 \\ \ell \neq g(i)}}^{k} \sum_{j \in H_\ell} w_{ij} + \sum_{H_\ell \cap V_\ell \neq \emptyset} 1 \tag{3.20}$$

$$\begin{aligned} \pi_d(i, H) &= \sum_{\substack{\ell=1 \\ \ell \neq g(i)}}^{k} \left(\sum_{j \notin H_\ell} w_{ij} + \sum_{m \in V_\ell} w_{im} \right) + \sum_{H_\ell \cap V_\ell = \emptyset} 1 + \sum_{H_\ell \cap V_\ell \neq \emptyset} 1 \\ &= \left(\sum_{\substack{\ell=1 \\ \ell \neq g(i)}}^{k} \sum_{j \notin H_\ell} w_{ij} + \sum_{H_\ell \cap V_\ell = \emptyset} 1 \right) + \pi(i, V) \end{aligned} \tag{3.21}$$

For the example in Figure 3.4, we first follow the steps involved in finding the solution to the optimization problem $H^* = \text{argmax}_{H \subseteq V} F(H)$, where $F(H) = \min_{i \in H} \pi(i, H)$ and $\pi(i, H)$ is given by (3.20). Following the procedure described in Table 3.1, at the first iteration, the element i is shelled out because $F(V) = \pi(i, V) = 4$. The elements h, c, f, g are shelled out at iterations 2, 3, 4 and 5 respectively. Finally, at the sixth iteration all the remaining elements a, b, d and e have the same linkage value 8 and they are all removed. Since 8 is the largest value of the function $F(H)$ at any iteration, the subset $\{a, b, d, e\}$ is declared as the optimal solution H^*. A trace of the elements shelled out at each iteration, along the $F(H)$ values of subsets encountered, is shown as the F-spectrum in Figure 3.5a.

Now, for the same example graph, we find the optimal solution to the optimization problem, $H_d^* = \text{argmin}_{H \subseteq V} F_d(H)$, where $F_d(H) = \max_{i \in H} \pi_d(i, H)$ according to (3.21). At the first, the element e is removed because the linkage function value $\pi_d(e, V)$ is greater than the linkage function value for any other element. Then, the element b is removed, followed by d, a, g, f and c. At the final iteration, the elements h and i are removed because they both have the same linkage function value, $\pi_d(h, \{h, i\}) = \pi_d(i, \{h, i\}) = 6$. We find that in this iterative procedure of shelling out elements, the minimum value for the function $F_d(H)$, in the sequence of nested subsets, was 6 and

the largest subset to reach this value was $\{c, h, i\}$. So, $H_d^* = \{c, h, i\}$. Like the F-spectrum for the maximization procedure, the F_d-spectrum obtained while minimizing the function $F_d(H)$ shows the sequence in which the elements are shelled-out along with $F_d(H)$ for the nested subsets obtained at each iteration. The F_d-spectrum for this example is shown in Figure 3.5b.

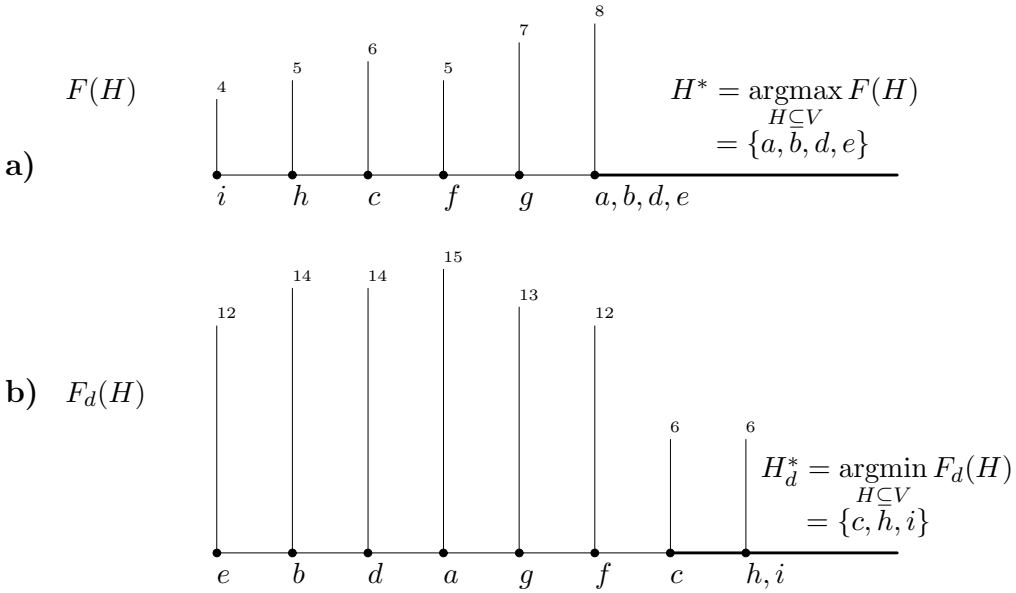

Figure 3.5: The contrast between the two spectra : F-spectrum and F_d-spectrum. Observe the relationship between the sequences in which elements are shelled out by the two procedures - the elements at the beginning of the first spectrum are located towards the end of the second spectrum and vice-versa.

Let us examine the sequence in which the elements are removed at each iteration while seeking the solutions to the two problems on the same example. The elements removed at the beginning of the iterative shelling algorithm for maximizing the quasiconcave set function are often the ones removed closer to the final iteration in the algorithm for minimizing the quasiconvex set function. It must be emphasized that this is a general observation and not an artifact resulting from the particular selection of the example, or the choice of the two linkage functions. This observation has two implications. First, a combination of these two spectra has the potential for improving the visualization of the structure of the relationships in the underlying graph. Secondly, from an algorithmic perspective, the information from these individual algorithms can be combined for solving both the problems by constructing a single algorithm which will likely be faster, in practice, than the individual algorithms,

The algorithms for the optimization problem described so far can have $|V|$ iterations in the worst case. At the same time, we have observed that the sequence in which elements are removed in the maximization and minimization algorithms are almost the opposite of each other. In other words, the elements removed at the earlier iterations by the minimizing procedure for the quasiconvex set function are likely to constitute the solution for maximizing the quasiconcave function. If so, one can run the minimizing procedure for a few iterations, and extract the solution that maximizes the quasiconcave function from the elements removed by the partial run of the minimizing procedure. Then, the only hurdle in designing a faster algorithm by reducing the number of iterations is how to recognize optimal solutions for the individual problems. In general, it is hard to recognize the optimal solution without completing the procedure of shelling out elements. However, when simultaneously working with the quasiconcave functions and quasiconvex functions, we can recognize the optimal solutions if their corresponding linkage functions satisfy a certain relationship.

When the monotonic increasing linkage function, $\pi(i, H)$ and the monotonic decreasing linkage function $\pi_d(i, H)$ satisfy the condition $\pi(i, V) \leq \pi_d(i, V)\ \forall i \in V$, the optimal solutions for the two problesm can be recognized. This is well formalized in the following theorem, which we will later use for designing an efficient procedure to efficiently solve the two problems efficiently.

Theorem 8. [3] *Let Γ_m be one of the subsets $\Gamma_\imath$; $(0 \leq \imath \leq p-1)$ produced by the iterative procedure to solve the original problem and similarly, let L_n be one of he subsets $L_\jmath$; $(0 \leq \jmath \leq q-1)$ produced by the iterative procedure to solve the complementary problem. Then,*

[a] If $F_d(L_n) = F(\Gamma_m)$, then $\Gamma_m \subseteq V \setminus L_{n+1}$ and $L_n \subseteq V \setminus \Gamma_{m+1}$.

[b] If $F_d(L_n) < F(\Gamma_m)$, then $\Gamma_m \subseteq V \setminus L_n$ and $L_n \subseteq V \setminus \Gamma_m$.

Proof. **[a]** We give a contradiction proof. Assume $F_d(L_n) = F(\Gamma_m)$ and $\Gamma_m \not\subseteq V \setminus L_{n+1}$ i.e., there is an element α such that $\alpha \in \Gamma_m$ and $\alpha \in L_{n+1}$. So, using

[3] In [56], this theorem is called the "duality theorem". Such relations between the extreme subsets of quasiconcave and quasiconvex functions, we prefer to call a complementary relation.

the definition of $F_d(\Gamma_m)$, quasiconvexity, and the relationship between the two linkage functions (in that order), we obtain the following sequence of inequalities:

$$F(\Gamma_m) \leq \pi(\alpha, \Gamma_m) \leq \pi(\alpha, V) \leq \pi_d(\alpha, V) \leq \pi_d(\alpha, L_n) \tag{3.22}$$

According to the procedure for finding the solution to the dual optimization problem we have $F_d(L_{n+1}) < F_d(L_n)$, and $\alpha \in L_{n+1}$. Further, the monotonically decreasing property of the linkage function $\pi_d(i, H)$ and the assumption implies the following

$$\pi_d(\alpha, L_n) \leq \pi_d(\alpha, L_{n+1}) \leq F_d(L_{n+1}) < F_d(L_n) \tag{3.23}$$

Combining (3.22) and (3.23) we get $F(\Gamma_m) < F_d(L_n)$ which is a contradiction, so there does not exist any element common to Γ_m and L_{n+1} implying that $\Gamma_m \subseteq V \setminus L_{n+1}$.

The proof for the other implicant is similar. Assume the contradiction i.e., $F_d(L_n) = F(\Gamma_m)$ and let there exist an $\alpha \in L_n$ such that $\alpha \in \Gamma_{m+1}$. Then, as in the previous case, we get

$$F_d(L_n) \geq \pi_d(\alpha, L_n) \geq \pi_d(\alpha, V) \geq \pi(\alpha, V) \geq \pi(\alpha, \Gamma_{m+1}) \geq F(\Gamma_{m+1}) > F(\Gamma_m)$$

This contradicts $F(\Gamma_m) = F_d(L_n)$, so it must be that $L_n \subseteq V \setminus \Gamma_{m+1}$.

[b] Similar to part [a]. Assume $F_d(L_n) < F(\Gamma_m)$ and $\alpha \in \Gamma_m$ such that $\alpha \in L_n$. Then, using the same logic as in part [a], we get

$$F(\Gamma_m) \leq \pi(\alpha, \Gamma_m) \leq \pi(\alpha, V) \leq \pi_d(\alpha, V) \leq \pi_d(\alpha, L_n) \leq F_d(L_n)$$

but this contradicts $F(\Gamma_m) > F_d(L_n)$, so $\Gamma_m \subseteq V \setminus L_n$. Furthermore, the second implicant directly follows from the first implicant which we have just proved. □

This theorem relating the solutions for the quasiconcave and quasiconvex functions resembles more a separation theorem because it characterizes the difference between

the solutions to the two problems. Moreover, the complementary problem is not about alternate representations/formulations of the same problem as in classical duality (for instance, convex optimization). The duality here involves two unrelated problems on a common set V and it states that there is a separation between the optimal solutions for these problems. However, the conditions stated in this theorem are sufficient to recognize the optimal solution.

Algorithm 3.7.1: ALGORITHM FOR ORIGINAL COMPLEMENTARY SYSTEMS(V)

$\jmath \leftarrow 0$; $L_\jmath \leftarrow V$;
$v_\jmath \leftarrow F_d(L_\jmath) \leftarrow \max_{\beta \in L_\jmath} \pi_d(\beta, L_\jmath)$;
$u_\jmath \leftarrow 0$;
while $(v_\jmath \geq u_\jmath)$
do
- $W_\jmath \leftarrow \{\beta \in L_\jmath : \pi_d(\beta, L_\jmath) \geq v_\jmath\}$; (1)
- $T \leftarrow \{\beta \in L_\jmath \setminus W_\jmath : \pi_d(\beta, L_\jmath \setminus W_\jmath) \geq v_\jmath\}$;
- **while** $(T \neq \emptyset)$
  - **do** $W_\jmath \leftarrow W_\jmath \cup T$; $T \leftarrow \{\beta \in L_\jmath \setminus W_\jmath : \pi_d(\beta, L_\jmath \setminus W_\jmath) \geq v_\jmath\}$;
- $L_{\jmath+1} \leftarrow L_\jmath \setminus W_\jmath$;
- $\jmath \leftarrow \jmath + 1$;
- $v_\jmath \leftarrow F_d(L_\jmath) \leftarrow \max_{\beta \in L_\jmath} \pi_d(\beta, L_\jmath)$; (2)
- $(\Gamma'_\jmath, u_\jmath) =$ MODIFIED-MAXIMIZATION-PROCEDURE $(V \setminus L_\jmath, u_{\jmath-1})$ (3)
- **comment:** *The modified-maximization-procedure is described in*
- **comment:** *Table 4.2 .*

OUTPUT $\Gamma'_\jmath$ as the solution for the original optimization, and
$L_\jmath$ as the solution for the complementary problem.

Table 3.5: An algorithm for simultaneously finding the maxima of the quasi-concave function (original problem) and the minima of the quasiconvex function (complementary problem).

We now present a modified algorithm for solving one of the optimization problems using the two complementary problems defined on the same multipartite graph, when their linkage functions satisfy the relationship discussed above. Theorem 8 provides us with a sufficient condition for recognizing the solution to both the problems. This test for recognizing the optimal solutions acts as a stopping condition for the modified algorithm.

The algorithm described in Table 3.5 is also an iterative algorithm. Each iteration can be divided into two phases - the shelling-out phase which includes statements (between statements (1) and (2) in Table 3.5) for removing elements during an iteration in the minimization of a quasiconvex set function, and the phase (statement (3) in Table 3.5) for solving maximizing the quasiconcave set function on the subset of elements removed in the iterations carried out so far by the shelling-out phase. Also, at each iteration the values of the current estimates of the solutions and their $F(H)$ and $F_d(H)$ are updated. Before beginning a new iteration, these values are compared to find if the optimal solutions to both the problems have already been reached by checking for the satisfaction of conditions in Theorem 8.

3.8 Conclusions

We have described monotone increasing functions for multipartite graphs and used them for constructive quasiconcave functions. We showed that when the quasiconcave functions are defined using the monotone increasing functions, they can be efficiently maximized and have described an algorithm for finding their largest maximizer. Analogously to the quasiconcave functions, we defined the quasiconvex functions using the monotone decreasing functions and using a one-to-one correspondence between the properties of these functions, we showed that, using an algorithm similar to the one for maximizing the quasiconcave functions, the quasiconvex functions can be minimized efficiently. Finally, we presented a relationship between the minimization and maximization problems and developed an algorithm for solving any one of the problems (minimization or maximization) using the shelling out procedure involving both the original and the complementary problems.

An advantage of simultaneously using the original and complementary optimization problems is that it allows a richer visualization of the data. We have already seen that the intermediate nested quasi-clusters obtained while solving original system, provide a layered structure view of the relationships between the elements. This is is also true for the intermediate subsets obtained while solving the complementary optimization problems. Together, they enable a visualization of the structure of relationships from

two different, independent perspectives.

The monotone functions, quasiconcave and the quasiconvex functions were described in this chapter in the context of multipartite graphs, but it is clear to see that their definition readily apply to simple graphs by removing the constraint imposed by the partite sets. So, the clustering methods described in these chapters are equally relevant for simple graphs.

Chapter 4

Ortholog Clustering: Problem Formulation

As discussed in Appendix B, one of the central problems in comparative genomics is the identification of genes in different organisms that are involved in similar biological functions. This requires identification of orthologs which are homologous genes that have evolved through vertical descent from a single ancestral gene in the last common ancestor of the considered species [29].

Generally, orthologs play the same biological roles in their respective species and are true functional counterparts in different species. From this perspective, a practical problem is finding clusters of genes, from multiple species, which perform the same biological function. In recent years, many genome-wide ortholog detection procedures have been developed [31, 43, 68, 81, 83, 96]. However, as discussed in Appendix B, they suffer from limitations that present real challenges for addressing the problem in a large set of genomes. Some of them are limited to identifying orthologs in a pair of genomes [31, 43, 68]; while [96] requires phylogenetic information, and they are not computationally efficient, and others [81, 83] require expert curation.

Known complete methods for finding ortholog clusters have at least two stages - automatic and manual. The role of the latter is to correct the results of the first stage which is usually a clustering procedure. Although specific implementations of clustering procedures in different methods vary, most successful methods include critical steps such as building clusters based on a set of "mutually most similar pairs" of genes from different genomes. These pairs are called BBH (bi-directional best hits [68, 81, 83]). This preprocessing is not robust since small changes in data or in the set of free parameters can alter the results substantially. So, currently there are three bottlenecks in ortholog extraction: (a) the manual curation, (b) time complexity, and

(c) the hypersensitivity of the automatic stage to parameter changes.

In this chapter, we describe a new method for finding orthologs in a large set of genomes using the combinatorial optimization problem formulation on a multipartite graph, which was studied in detail in chapters 2 and 3.

4.1 Issues addressed in ortholog clustering formulation

We address the drawbacks in the existing methods by overcoming their bottlenecks through a novel computational reformulation of the ortholog clustering problem. The precise formulation using the theory developed in the previous chapters is described in the following section. Here we briefly point out some of the issues in ortholog clustering addressed by the problem formulation on a multipartite graph. Our problem formulation has the following advantages:

a. *Avoids intra-genome paralogs.* As orthologs can be present only across species, only gene-similarity across genomes needs to be considered.

b. *Ortholog extraction from multiple genomes.* The popular approaches where seed orthologs from a couple of genomes are merged together to form ortholog clusters can be error-prone due to inconsistencies introduced, mainly, in the merging step. In our approach, an ortholog cluster is directly extracted as a subset of highly similar genes from multiple genomes. Since this approach analyzes all the similarities in the subset simultaneously, it reduces chances of introducing any inconsistency inside a cluster.

c. *Differentiates the anciently duplicated paralogs from the recent ones.* We focus on "functional orthologs", i.e., genes in different genomes that are potentially involved in the same function. From this perspective, paralogous genes that have duplicated recently are likely to be involved in the same cellular function; while those resulting from ancient duplications are likely to be involved in different functions. With this assumption, clustering the potential functional orthologs requires keeping the recently duplicated paralogs in the same ortholog clusters while placing anciently duplicated genes in separate clusters.

d. *Corrects for the evolutionary distance between the species.* The numerical value of the observed similarity between orthologs from closely related species is likely to be higher compared to the observed similarity between orthologs from distantly related species. A computational method based on numerical values of observed similarity values between sequences must correct for this bias in the observed similarities to produce meaningful results. Such a correction is possible using the species tree relating the species in the data.

e. *Enables working with incomplete genomes.* Finding orthologs in the partially completed genomes is critical for inferring the functional contents of their genomes. As most genomes remain in the unfinished state for a considerable time, the ortholog clustering problem in the partially complete genomes is a real challenge. This is addressed by using an optimization criterion that does not depend on the state of completeness of the genomes.

f. *Robustness to slight changes in the set of input sequences.* Due to the element to subset relationship, a slight change in the relationships (slight changes in the pair-wise similarities and exclusion of a few sequences) inside the subset is not likely to perturb the final clusters significantly.

g. *Intuitive interpretation.* The combinatorial optimization criterion gives clusters a straightforward interpretation.

h. *Computationally efficient algorithm.* As described in Chapter 2, there is an efficient algorithm for solving the optimization problem leading to a fast ortholog clustering for a large number of genomes.

These issues will be visited again in the next section while formulating the ortholog clustering problem. The desirable effects for addressing these issues are achieved by modeling the problem on a multipartite graph and designing an optimization criterion which is both suitable and efficient for extracting ortholog clusters (quasicliques in the multipartite graph).

4.2 Ortholog clusters on a multipartite graph

We use a multipartite graph for representing the similarities between the genes across species and for representing the evolutionary tree (species tree) between the species.

Consider the ortholog clustering problem with k genomes, where $V_i,\ i \in \{1, 2, \ldots, k\}$ represents the set of genes from the genome i. Then, the similarity relationships between genes from different genomes can be represented by an undirected weighted multipartite graph $G = (V, E, W)$, where $V = \cup_{i=1}^{k} V_i$ and V_i is the set of genes from the genome i, and $E \subseteq \cup_{i \neq j} V_i \times V_j$ is the set of weighted, undirected edges representing similarities between genes, as shown in Fig. 4.1.

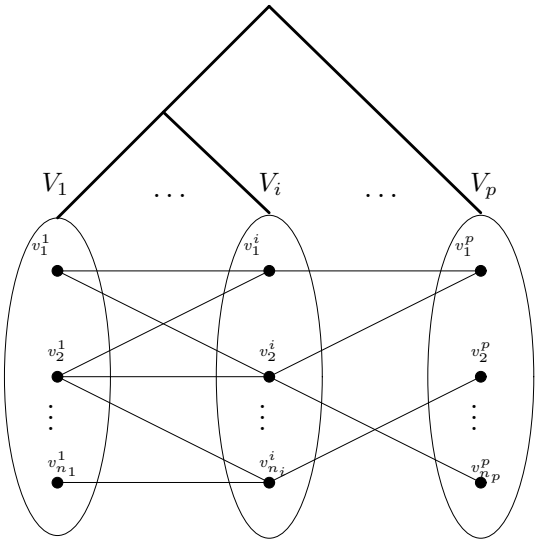

Figure 4.1: A multipartite graph representing relationships between genes (small solid circles) from different genomes (ellipses). There are k different genomes and the i^{th} genome V_i contains n_i elements v_1^i through $v_{n_i}^i$. The straight lines connecting the genes across the genomes represent the weighted similarity between them (dissimilar genes are not connected by edges), while the species tree is represented by weighted thick lines connecting the genomes. For visual clarity, weights are not shown on the edges.

The primary sequence similarity between sequences in an ortholog cluster is likely to be high. So, the problem of finding an ortholog cluster can be modeled as finding a maximum quasiclique as described in chapter 2. Then, (as in chapter 2), for finding a weighted multipartite quasi-clique as an ortholog cluster we assign a score $F(H)$ to any subset H of V. Ortholog clusters can contain genes from two or more genomes, so we require that H contain genes from at least two genomes. The score function denotes

a measure of proximity among genes in H. Then, our multipartite quasi-clique, or cluster, H^* is defined as the subset with largest score value, i.e.,

$$H^* = \arg\max_{H \subseteq V} F(H) \tag{4.1}$$

The subset H contains genes from multiple genomes, so according to (4.1) our approach finds an ortholog cluster as a set of genes from multiple genomes by simultaneously considering all the similarity relationships in H. This is novel since all sequence similarity based methods, such as [83, 68], find an initial set of orthologs from two genomes and may extend them at later stages. The function $F(H)$ is designed using a linkage function $\pi(i, H)$ which measures the degree of similarity of the gene $i \in H$ to other genes in H. The function $F(H)$ is then defined as

$$F(H) = \min_{i \in H} \pi(i, H), \quad \forall i \in H \ \ \forall H \subseteq V \tag{4.2}$$

Then, according to (4.1), the subset, H^* contains genes such that similarity of the least similar gene in H is maximum. Since we use the quasiconcave set function optimization approach, described in the previous chapters, these properties have to be achieved by designing a suitable linkage function. We could use the linkage function described in the previous chapter but another linkage function, seems more suitable for modeling ortholog clustering. This linkage function corresponds to the quasiconcave function, $F_3(H)$, given by the equation (4.4) (with the choice $\alpha = \beta = 1$) and is given below:

$$\pi(i, H) = \sum_{\substack{\ell=1 \\ \ell \neq g(i)}}^{k} \rho_{g(i),\ell} \left\{ \sum_{j \in H_\ell} w_{ij} - \sum_{j \in V_\ell \setminus H_\ell} w_{ij} \right\} \tag{4.3}$$

where the subset $H \subseteq V$ that contains genes from at least two genomes, and can be decomposed as $H = \cup_{i=1}^{k} H_i$ such that H_i is the subset of genes from V_i present in H. The weight, w_{ij} (≥ 0), on the edges of the similarity graph, is the pair-wise similarity value (estimated using Blast [4]) between gene i from genome $g(i)$ and gene

j from another genome $g(j)$ ($g(j) \neq g(i)$). The term $\rho_{g(i),g(j)}$ represents the distance between the genomes $g(i)$ and $g(j)$. Thus, the proposed function considers the pair-wise sequence similarity between genes within the ortholog cluster, their relationship to genes outside the cluster, and the phylogenetic distance between the corresponding genomes.

The term $\sum_{j \in H_\ell} m_{ij}$ aggregates the similarity values between the genes i from genome $g(i)$ and all other genes in the subset H that do not belong to genome $g(i)$, while the second term, $\sum_{j \in V_\ell \setminus H_\ell} m_{ij}$, estimates how this gene is related to genes from genome ℓ that are not included in H_ℓ. A large positive difference between these two terms ensures that the gene i is highly similar to genes in H_ℓ and at the same time very dissimilar from genes not included in H_ℓ. From a clustering point of view, this ensures large values of intra-cluster homogeneity and inter-cluster separability for extracted clusters. Applied to ortholog clustering, such a design enables a separation of ortholog clusters related to anciently duplicated paralogs.

The scaling term $\rho_{g(i),g(j)}$ is used for correcting the observed sequence similarities by magnifying the sequence similarities corresponding to genomes which diverged in ancient times. The idea behind such a magnification is to rescale the observed similarity values such that the pair-wise similarity values between orthologs from any pair of species are almost equal. Given the phylogenetic tree relating the species under study, the distance, $\rho_{g(i),g(j)} (\geq 0)$, between the genomes can be defined in various ways. It can be defined as the estimate of the divergence time between the species $g(i)$ and $g(j)$, but such estimates are usually not available. So, we used the topology of the species tree to define this distance. Using the tree topology, there are various ways to formalize $\rho_{g(i),g(j)}$ such as the height, $h_{g(i),g(j)}$, of the subtree rooted at the last common ancestor of $g(i)$ and $g(j)$. The exact definition of $\rho_{g(i),g(j)}$ as a function of $h_{g(i),g(j)}$ depends on the species in data for ortholog detection. When the species are closely related (such as the members of the grass family in plants), a function that depends on $h_{g(i),g(j)}$ but grows slower will better model the distance between the species. Choosing an appropriately growing function is critical because a faster growing function will have the undesirable effect of clustering together sequences from distance species but leaving out sequences

from closely related species. So, the distance $\rho_{g(i),g(j)}$ (≥ 0) between $g(i)$ and $g(j)$ is defined as $(1 + \log_2 h_{g(i),g(j)})$.

Through formulating the problem on the multipartite graph, we have been able to avoid the paralogs whose presence is limited to a single genome. When paralogs are present across genomes, we would like to discriminate between the anciently duplicated paralogs from the recently duplicated ones, as recently duplicated paralogs are likely to have the same function and so must belong to the same cluster of potential functional orthologs. We achieve this by carefully (but automatically) selecting the pair-wise similarity edges in the multipartite graph as described below.

Due to functional constraints, orthologs with the same function are likely to display higher levels of similarity compared to paralogs which tend to diverge in sequence and function. So, we consider only a subset of the most similar matches for any given protein sequence, as in [43]. To be precise, if a sequence i in species $g(i)$ has the sequence j as its best-match in species $g(j)$ with score w_{ij}, we considered all those matches for i from species $g(j)$ which score higher than θw_{ij} $(0 \leq \theta < 1)$. The idea behind this is to avoid low-scoring spurious matches (most likely paralogs) without filtering out potential orthologs. Although, the precise choice of the value of θ seems to be critical, in our experiments (described in the following chapter) we found that clustering results do not vary significantly in the range $0.5 \leq \theta \leq 0.8$. So, we chose a conservative value, $\theta = 0.5$, to avoid ignoring any potential orthologs [1].

4.3 Annotating sequences with existing ortholog clusters

Protein annotation using orthologs is widely recognized [25, 33], but in the absence of reliable automatic tools for identifying such evolutionarily related proteins, functional annotation is carried out by semi-automatic methods requiring expert intervention. A popular method for annotating a target protein is by transferring the annotation

[1] Here θ is chosen to be a constant for all sequences. However, its value will vary across protein functional families. It can be determined by a separate study of divergence in sequence similarity among paralogs following a duplication event. To be precise, it requires modeling sequence evolution for the duplicates to bring out the critical level of divergence in the sequence similarity that leads to neo-functionalization or sub-functionalization of paralogs [35].

of the nearest-neighbors from closely related species [33]. For proteins, the neighbor-relationship is multi-faceted and encompasses similarity in sequence composition, proximity on the chromosome, expression levels in the cell, etc. In practice, however, the nearest neighbor relationship is solely guided by sequence similarity, and is often implemented using the closest Blast [4] hit. Annotations based on best-Blast-hit can be error-prone [47], and have resulted in uncontrolled propagation of function annotation errors in protein sequence databases [2]. So, an effective automatic function annotation approach must have (i) a reliable method for ortholog clustering, (ii) a sensitive criterion for identifying ortholog neighbors for a target protein, and (iii) a source of data, with trustworthy annotations, for constructing ortholog clusters. The first of these issues is addressed by using the ortholog clustering method proposed in the previous section. In this section, we develop a criterion for annotating a target sequence with an orthologs cluster. The third issue is addressed (in Chapter 5) by choosing the partially complete genomes (whose proteins are experimentally annotated) for constructing the ortholog clusters which together with a robust criterion allow annotation of target proteins.

4.3.1 Criterion for annotating target proteins

Annotating a query sequence requires finding an ortholog cluster whose proteins are orthologous to the query protein. In an extreme case, ortholog clusters may need to be reconstructed using the target sequences as part of the input data. On the other hand, if the existing ortholog clusters are stable and the annotation criterion is stringent and similar to the one for extracting clusters, then existing clusters will be retained merely as extensions of clusters obtained on additional input data. However, such results depend on the criterion used for annotating the query sequences. We describe such a criterion below.

A query protein sequence q should be annotated with the cluster, $\widehat{H}$ whose proteins are orthologous to it. So, the degree of orthologous membership (C.1) of q to the ortholog cluster $\widehat{H}$ should be large. Also, since protein q is orthologous to proteins in

$\widehat{H}$, we expect the following to hold:

$$\pi(q, \widehat{H}) \geq F(\widehat{H}) \tag{4.4}$$

In other words, inclusion of q should not adversely affect the score $F(\widehat{H})$. We use (4.4) to select the candidate ortholog clusters with which a query protein can be potentially annotated. It is possible that no candidate ortholog cluster satisfies (4.4), and in such a case q cannot be annotated using the ortholog clusters. On the other hand, multiple ortholog clusters, $H_q = \{H^* : \pi(q, H^*) \geq F(H^*)\}$ may satisfy (4.4). To uniquely annotate q, we select the cluster with highest average similarity to q

$$\widehat{H}_q = \arg \max_{H \in H_q} \pi(q, H) \,/\, |H| \tag{4.5}$$

The criteria (4.4) and (4.5) to annotate proteins are derived from the score function for extracting the clusters. Unlike the nearest-neighbor based criterion for annotation queries with ortholog clusters, our criteria are less sensitive to slight changes in pair-wise similarity values between proteins. By integrating similarity values between the query protein and proteins in an ortholog cluster, our criterion (4.4) estimates the degree of orthologous relationship between the query and sequences in the ortholog cluster. Additionally, this criterion is very sensitive to the homogeneity of an ortholog cluster, as the condition (4.4) will not be satisfied unless all members of a cluster are similar. Our criteria are more stringent than the best-Blast-hit criterion (where a query sequence is annotated with the cluster of its best Blast hit). As indicated by (4.4), the set of query sequences annotated by our criteria is contained in those annotated by the best-Blast-hit criterion. So, our criterion provides less but more sensitive annotation coverage.

4.4 Validating Ortholog Clusters

To demonstrate the performance of the proposed method, we will be using multiple datasets ([85, 86, 87] and Chapter 5 for details about the data). From a validation

perspective, these datasets can be divided into two categories. The first contains sequence datasets where information on expertly curated ortholog clusters is available and can be taken as "gold standard" ortholog clusters for comparing our clustering results. The second category contains sequence datasets where information about the ortholog clusters is not known. The validation methodologies for ortholog clusters constructed from these two kinds of datasets will differ due to varying levels of knowledge about the ortholog clusters. Furthermore, the methods for validating clusters on the second kind of datasets are applicable for datasets in the first category by simply ignoring the additional information about known clusters, but not vice-versa.

While there are very many datasets in the second category which require no information on existing ortholog clusters, instances of the first kind of datasets are very few. An example dataset of the first kind is the COG (Clusters of Orthologous Genes) [83] database (details are described in the Chapter 5) containing expertly curated ortholog clusters along with the set of sequences from which those ortholog clusters are constructed. We will use the COG dataset for comparing the performance of our method with the existing COG clusters. As an instance of dataset in the second category, we will use the cereal genomes where we do not have prior knowledge of ortholog clusters; we will also use this dataset for demonstrating the efficacy of ortholog clusters discovered by our method in annotating protein sequences from the rice genome.

Validating ortholog clusters extracted from data with known ortholog clustering results is the classical (but difficult) problem of comparing two different clusterings on the same data. We will use multiple validation methodologies including some classical statistics for comparing results of two clusterings, indices for comparing partitions, and a few indices defined by us for a set theoretic comparison of clusters. Finally, we will describe a methodology for validating ortholog clusters which involves assessing the homogeneity of an ortholog cluster based on independently assigned functional and structural annotation for the member sequences. Such an approach for assessing the homogeneity within ortholog clusters is more appropriate for data where we do not have a-priori knowledge of ortholog clusters. The validation results using the various indices described in this section will be described in chapter 6.

4.4.1 Indices for comparing two ortholog clusterings

Generally speaking, validating clusters requires developing quantitative measures for summarizing the homogeneity inside clusters, but validating groups of orthologous genes is a very specific problem. Although classical statistics contains a wide variety of indices dealing with the problem of comparing two classifications, our validation methodology does not strictly subscribe to a classical hypothesis testing paradigm. This is because when two classifications are "highly dependent", as we expect in our case, the quantitative measures for measuring similarity remain largely unaddressed by classical statistics, especially for cases with a large number of clusters. Aggregative statistical measures summarize the relatedness of two classification schemes but fail to provide insights into per-cluster-based agreements between the clusterings. We propose a specific methodology oriented towards explorative screening of orthologous relations between genes from different genomes. Other applications of the method may well require different validation tests.

To obtain quantitative measures of overall similarity between two clusterings, we use a few statistical coefficients. Association between two classifications is estimated using the standard χ^2 statistics and a normalized version of the χ^2 coefficient, the *Cramer's V-coefficient* [27]. Furthermore, for a direct measure of degree-of-association between COG clusters and our clusters, we use the *Rand index* [67] and its modification the *Adjusted Rand index* [40]. The Rand indices involve counting the number of agreements between two classes based on how each of pair of elements in the underlying set is assigned to the classes in the two classification schemes.

The first group of indices for comparing the COG clustering with our clustering estimates the similarity between two data partitions into two classes: (i) COG, the set of sequences which belong to the COGs, and $\overline{COG}$, the set of sequences which are not part of any COG cluster; (ii) the set of proteins, Cl, which belong to the union of our clusters, and the set $\overline{Cl}$ containing the remaining proteins. This group of indices, called *indices for a* 2×2 *contingency table*, determines how sequences are classified into two classes - one class containing sequences that share orthologous relations to other

sequences in the data and the second class containing sequences which do not have any orthologous sequences in the data. These indices are denoted by αs with indices. The second group of indices, denoted by βs, is the set of *indices for assessing the homogeneity inside our ortholog clusters*. These indices evaluate each cluster according to: (a) what portion of a cluster contains sequences do not belong to any COG, (b) how does the complementary part distribute across all COGs. In effect, the β-indices summarize the homogeneity of our clusters in relation to their membership in COG clusters. The third group consists of *indices for assessing the homogeneity within our clusters based on homogeneity of COGs* and are denoted by γs. These indices are similar to the βs and focus on quantitating the relationship between a single COG and all our clusters. The γ-indices involve decomposing each COG into clusters and are useful in measuring the aggregating ability of our procedure (whether we produce very small but tight clusters or large but "loose" clusters). The α, β, and γ indices are described in this section.

Statistical coefficients for aggregative similarity between two partitions

The first and second groups of coefficients are based on an aggregative comparison of two partitions. In our case the two partitions are the classifications induced by COGs and our clusters. Although there are many statistical coefficients to measure the correspondence between two partitions, there is no single coefficient which suffices to present a case for strong validation or has a straightforward interpretation [6]. This is why we use the three different types of coefficients which complement each other in interpretation.

All measures of association we have used can be conveniently expressed on a contingency table (also called the cross-classification table) representing the joint distribution of elements according to the two classifications. An example of a contingency table is shown in Table 4.1. Assume that the elements in the set V are classified according to two classifications, $\mathcal{A}$ and $\mathcal{B}$, so that they induce partitions, $A = \{a_1, \ldots, a_r\}$, and $B = \{b_1, \ldots, b_c\}$, respectively, i.e., $\cup_{i=1}^{r} a_i = V = \cup_{j=1}^{c} b_j$ and $a_i \cap a_{i'} = \phi = b_j \cap b_{j'}$ for $1 \leq i \neq i' \leq r$ and $1 \leq j \neq j' \leq c$. A contingency table, $\mathbf{T}$, corresponding to these two classifications is an $r \times c$ matrix, $\mathbf{T} = ||t_{ij}||$, where t_{ij} is the number of elements

		Partition B				
	$Class$	b_1	b_2	$\cdots$	b_c	$Sums$
Partition A	a_1	t_{11}	t_{12}	$\cdots$	t_{1c}	$t_{1.} = \sum_{j=1}^{c} t_{1j}$
	a_2	t_{21}	t_{22}	$\cdots$	t_{2c}	$t_{2.} = \sum_{j=1}^{c} t_{2j}$
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	a_r	t_{r1}	t_{r2}	$\cdots$	t_{rc}	$t_{r.} = \sum_{j=1}^{c} t_{rj}$
	$Sums$	$t_{.1} = \sum_{i=1}^{r} t_{i1}$	$t_{.2} = \sum_{i=1}^{r} t_{i2}$	$\cdots$	$t_{.c} = \sum_{i=1}^{r} t_{ic}$	$t = \sum_{i=1}^{r}\sum_{j=1}^{c} t_{ij}$

Table 4.1: A $r \times c$ contingency table.

belonging to $a_i \cap b_j$ i.e., t_{ij} is the number of elements co-classified in the classes $a_i \in A$ and $b_j \in B$. The marginal sums $t_{i.} = \sum_{j=1}^{c} t_{ij}$ and $t_{.j} = \sum_{i=1}^{r} t_{ij}$ is the number of elements in the classes a_i and b_j, respectively, and $t = \sum_{i=1}^{r}\sum_{j=1}^{c} t_{ij}$ is the total number of elements in the set V ($t = |V|$).

Test of independence

One of the classical tests for assessing the independence of two classifications is the χ^2 goodness-of-fit test. For an $r \times c$ contingency table, $\mathbf{T}$, this coefficient is given by the formula:

$$\chi^2 = \sum_{i=1}^{r}\sum_{j=1}^{c} \frac{(t_{ij} - t_{i.}t_{.j}/t)^2}{t_{i.}t_{.j}/t} \tag{4.6}$$

In order to assess if the two classifications are independent, the χ^2 value computed using (4.6) must be compared with the value of χ^2-distribution for appropriate degrees of freedom in the contingency table. The degrees of freedom for a $r \times c$ contingency table is $(r-1)(c-1)$ [27]. Then, to assess the independence of two classifications, for some *a priori* determined significance level, α ($0 \leq \alpha \leq 1$, usually, α is 0.05 or 0.01), the χ^2 value in (4.6) is compared with the value attained by the χ^2-distribution for parameters

α and $(r-1)(c-1)$ degrees of freedom. If this value for the χ^2-distribution is larger than the χ^2 value computed using (4.6), the two classifications are inferred to be dependent. The χ^2 value can assume any value greater than 0 and this raw value by itself has no relevance in describing the degree of association between the two classifications. There are several modifications of the coefficients, which attempt to normalize the χ^2 value so that the values of resulting statistics lie between 0 and 1 and takes on extreme values under independence and complete association. One such modification that we use is Cramer's V-coefficient [27].

$$\text{Cramer's V-coefficient} = \frac{\chi^2/t}{\min\{r-1, c-1\}} \tag{4.7}$$

It is the most popular of the χ^2-based coefficients since it provides a good normalization on the 0 to 1 interval regardless of the table size. This coefficient can be interpreted as the association between two classifications as a percentage of their maximal possible variation; it attains the value 1 when the two marginals associated with each class are equal.

Pair-wise agreement based coefficients

The χ^2 statistics allows us to infer if two clusterings/classifications are randomly related, or they are dependent. The Rand index [67] belongs to the complementary class of coefficients that directly attempts to measure the degree of association between the two clusterings. It involves counting the number of pairs of elements of the set V which are similarly clustered by the two methods. Suppose t_1 is the number of pairs of elements that are classified in the same class by both methods, t_2 represents the number of pairs of elements that are classified into different classes by the two methods, t_3 represents the number of pairs of elements that are classified into same class in classification $\mathcal{A}$ but into different classes in classification $\mathcal{B}$, and t_4 represents the number of elements that are classified into different classes in $\mathcal{A}$ but into the same class in $\mathcal{B}$. Using these numbers, one can calculate two coefficients $R = t_1 + t_2$ and $D = t_3 + t_4$. Intuitively, two similar classifications produce large values of R and small values of D. The corresponding

	Cl	$\overline{Cl}$
COG	a	b
$\overline{COG}$	c	d

Table 4.2: A 2×2 contingency table (confusion table).

indices are calculated by the formulas:

$$R = \binom{t}{2} + \sum_{i=1}^{r}\sum_{j=1}^{c} t_{ij}^2 - \frac{1}{2}\left[\sum_{i=1}^{r} t_{i.}^2 + \sum_{j=1}^{c} t_{.j}^2\right] \tag{4.8}$$

$$D = \frac{1}{2}\left[\sum_{i=1}^{r} t_{i.}^2 + \sum_{j=1}^{c} t_{.j}^2\right] - \sum_{i=1}^{r}\sum_{j=1}^{c} t_{ij}^2 \tag{4.9}$$

In [40], Hubert and Arabie proposed the Adjusted Rand index, R_{adj}, which indicates how significant is the agreement compared to the agreement expected between two random classifications on the same size of data. Assuming a hyper-geometric distribution as the model of randomness, i.e., the $\mathcal{A}$ and $\mathcal{B}$ classifications are random as long as the number of elements in classes of the two classifications remains fixed. Clearly, this is a normalized index which is bounded above by 1 and takes the value 0 when the index equals its expected value, and is given by the formula:

$$R_{adj} = \frac{\sum_{i=1}^{r}\sum_{j=1}^{c}\binom{t_{ij}}{2} - [\sum_{i=1}^{r}\binom{t_{i.}}{2}\sum_{j=1}^{c}\binom{t_{.j}}{2}]/\binom{t}{2}}{\frac{1}{2}[\sum_{i=1}^{r}\binom{t_{i.}}{2} + \sum_{j=1}^{c}\binom{t_{.j}}{2}] - [\sum_{i=1}^{r}\binom{t_{i.}}{2}\sum_{j=1}^{c}\binom{t_{.j}}{2}]/\binom{t}{2}} \tag{4.10}$$

Indices for 2×2 contingency table, the α's

The α indices attempt to determine the agreement between two binary classifications. Almost all statistical coefficients for this case are functions of the values in a 2×2-contingency table. For our case, this table is shown in Table 4.2, where COG-$\overline{COG}$ and Cl-$\overline{Cl}$ are different partitions/classifications; the numbers a, b, c, and d are the cardinalities of intersections $COG \cap Cl$, $COG \cap \overline{Cl}$, $\overline{COG} \cap Cl$, and $\overline{COG} \cap \overline{Cl}$, respectively. One such average coefficient is $\alpha_1 = \frac{a+d}{a+b+c+d}$. Additionally, we calculate local coefficients α_2, α_3, α_4, and α_5. The α coefficients along with their formulas and explanation are given in Table 4.3.

Index	Explanation
$\alpha_1 = \frac{a+d}{a+b+c+d}$	Represents accuracy of classification. $\alpha_1 \in [0,1]$. Larger values are better.
$\alpha_2 = \frac{a}{a+b+c+d}$	Sequences classified as orthologous sequences in both classifications, expressed as a fraction of all elements. Larger values are better.
$\alpha_3 = \frac{b}{a+b+c+d}$	Fraction of sequences classified as orthologous sequences only in the COG database. Smaller values are better.
$\alpha_4 = \frac{c}{a+b+c+d}$	Fraction of sequences classified as orthologous sequences only by our procedure. Smaller values are better.
$\alpha_5 = \frac{d}{a+b+c+d}$	Sequences classified as non-orthologous sequences by both classifications, expressed as a fraction of all elements. Larger values are better.

Table 4.3: The expressions for the α-indices along with their explanations.

The β-indices for assessing homogeneity within our clusters

This group of indices determines the homogeneity of our clusters based on how the members of clusters are distributed in different COG clusters. If our clustering recapitulates the COG clustering, we would expect that for most of our clusters, their member sequences will belong to a single COG cluster. In essence, these indices are geared towards bringing out the set-theoretic relationship between a cluster and the clusters in the COG database. Then, this set of indices evaluate individual clusters for various notions of homogeneity, while the averages of these indices summarize the overall homogeneity of our clusters with respect to COG clusters. The various notions of homogeneity are based on the decomposition of a cluster into subsets such that elements in each subset belong to a single COG or to the set $\overline{COG}$. To formally define these indices, assume that the i^{th} cluster, $\hat{H}_i, 1 \leq i \leq m$, contains l_0^i (l_0^i possibly 0) sequences that do not belong to any COG, and the remaining sequences belonging to $k_i (k_i \geq 0)$ different COGs with l_1^i sequences belonging to the COG that shares the largest number of elements with this cluster, l_2^i to the COG that shares the second largest number of elements with this cluster, and so on, i.e., $l_1^i \geq l_2^i \ldots l_{k_i}^i$, $\sum_{j=0}^{k_i} l_j^i = n_i = |\hat{H}_i|$. This class of indices is summarized in Table 4.4.

The index, β_1^i, is the fraction of non-COG sequences in the i^{th} cluster, and the

Index	Explanation
$\beta_1 = \frac{1}{m}\left\{\sum_{i=1:\beta_1^i \leq 0.5}^{m} \beta_1^i + \sum_{i=1:\beta_1^i > 0.5}^{m}(1-\beta_1^i)\right\}$, $\beta_1^i = \frac{l_0^i}{n_i}$	Average fraction of sequences in a cluster that belong to the set $\overline{COG}$. $0 \leq \beta_1 \leq 0.5$, lower values are better.
$\beta_2 = \frac{1}{m}\sum_{i=1}^{m} \beta_2^i,\ \beta_2^i = k_i$	Average number of COGs in a cluster. $\beta_2 \geq 1$, lower values are desirable.
$\beta_3 = \frac{1}{m}\sum_{i=1}^{m} \beta_3^i$ where $\beta_3^i = \min_j\{(\sum_{w=1}^{j}) \geq 0.5\}$	Average of the minimum of number of COGs required to make at least half the cluster. $\beta_3 \geq 1$, lower values are desirable.

Table 4.4: Expressions and explanation of the indices to evaluate homogeneity of clusters with respect to COGs.

correspond index, β_1, is the average of β_1^i's over all clusters. So, β_1 is an estimate of homogeneity within our clusters with respect to distinguishing elements in the sets COG and $\overline{COG}$. The lower values of this index are better, however, when cluster, $\hat{H}_i$, is a subset of the set $\overline{COG}$ this index assumes a value 1 but according to the notion of homogeneity which this index is expected to capture, it is a perfectly homogeneous cluster, so, β_1 is not a simple average but modified so as to be appropriate for the intended notion of homogeneity (see Table 4.4). The β_2^i index is the number of different COGs in the cluster and β_2 is the simple average over all clusters. Since, for the purpose of evaluating, we count the set $\overline{COG}$ as a COG, it is clear that β_2 is at least 1 and the value 1 indicates that all clusters are perfectly homogeneous, i.e., each cluster contains either elements from single COG, or all its elements belong to the set $\overline{COG}$. The index β_2 provides only an average estimate and as such it fails to provide a good estimate of the degree of homogeneity within a cluster. For instance, if β_2 is high but most sequences in a cluster belong to a single COG, the cluster should still be considered as acceptable. To avoid being misled by this scenario, we devised another index β_3 which is an average of β_3^i's over all clusters. Index β_3^i is defined as the minimal number of COGs required to make up at least half the cluster, $\hat{H}_i$. Obviously, β_3 is at least 1, and since the lower values indicate that at least half the elements are in a few COGs, 1 is the best possible value it can achieve.

Index	Explanation
$\gamma_1 = \frac{1}{\lvert COG \rvert} \sum_{j=1}^{\lvert COG \rvert} \gamma_1^i,\ \gamma_1^i = \frac{c_j^0}{c_j}$	Average number of sequences in a COG that are not part of any cluster. $0 \leq \gamma_1 \leq 1$, smaller values are preferred.
$\gamma_2 = \frac{1}{\lvert COG \rvert} \sum_{j=1}^{\lvert COG \rvert} \gamma_2^i,\ \gamma_2^i = s_j$	Average number of clusters into which a COG is shattered. $\gamma_2 \geq 1$, smaller values are better.
$\gamma_3 = \frac{1}{\lvert COG \rvert} \sum_{j=1}^{\lvert COG \rvert} \gamma_3^i$ where $\gamma_3^i = \sum_{w=1}^{s_j} \beta_2^{c_j^w}$	Average number of COGs in the clusters into which a COG is shattered. $\gamma_3 \geq 1$, smaller values are better.

Table 4.5: Expressions and explanation of indices used to indirectly evaluate our clusters, through the evaluation of COGs.

Indices assessing homogeneity within clusters based on homogeneity of COGs, γ's

The third group of indices, similar to the β-indices, give an indirect measurement of the homogeneity within our clusters. For measuring this, we actually evaluate COGs in such a way that they provide an evaluation of our clusters. Suppose that j^{th} COG, C_j, contains c_j elements of which c_j^0 elements that do not belong to any cluster and the remaining c_j^1 elements belong to s_j different clusters, $\hat{H}_{c_j^1}, \hat{H}_{c_j^2}, \ldots, \hat{H}_{c_j^{s_j}}$. Then, the index γ_1^j measures the fraction of elements in the COG C_j that do not belong to any clusters and the average index γ_1 is an indicator of fraction of COG elements that are not accounted for in the clusters. The index γ_1 can take values between 0 and 1 inclusive, and lower values indicate that a larger fraction of data is correctly screened as orthologs by our clustering. Another measure of quality of our clusters is the number of clusters into which a single COG is shattered, a low value of such measure would mean that we correctly determine the orthologous relationships across orthologous family present in the COG, on the other hand, a higher value would mean that we recognize the orthologous relationships in closely-related sequences only. To study this effect, we designed the index γ_2^j to assess the number of different clusters present in the j^{th} COG, and the related average index, γ_2 measures this effect for all COGs. It is apparent that

this index is at least 1 and lower values are desirable. Although, the index γ_2 provides a good insight into the shattering phenomenon of our clusters, it fails to bring out an important homogeneity related assessment of our clusters, viz., do the clusters that make up a COG contain elements from that specific COG only or they contain elements from other COGs as well? To investigate this, we studied another index, γ_3^j, which is the number of different COGs in the clusters required to make up the j^{th} COG, the corresponding index γ_3 measures the average number of number of different COGs in the clusters that make up a COG.

4.4.2 Validating ortholog clusters using independent sources of information

In this section, we describe a methodology for validating ortholog clusters which are discovered from data in which we do not have any prior information about "true" ortholog clusters. In such cases, we must rely on some properties (which are not used for finding clusters) of the elements in a cluster. In a good clustering, we would expect that the members of clusters are homogeneous with respect to those properties. In the context of clustering orthologous proteins, two such properties are the functional annotation of proteins, and their 3D structures. Fortunately, there are reliable databases of sequences which have classified proteins according to these properties. We intend to use these classifications (as surrogates for the information on known ortholog clusters) for evaluating the homogeneity of ortholog clusters found by our method.

Before describing the details of evaluating ortholog clusters using the functional and/or the 3D-structural annotations for sequences, we must point out three caveats to this approach.

- First, the databases containing such information (function/structure) are not complete, i.e., they do not cover the entire space of functions of protein sequences (this is especially true for the coverage of space of structures for proteins, where the coverage is significantly sparse). Consequently, a significant proportion of clusters for which we cannot provide functional (or structural) annotations cannot be

validated, despite the fact that they actually contain orthologous sequences.

- Secondly, the databases contain functional (or structural) annotations for fragments of protein sequences. So, a protein sequence can have multiple and different functional annotations which can potentially introduce complications in the evaluation process.

- The third issue is related to the fact that a similar functional/structural annotation is only indicative of an orthologous relationships (because non-orthologous sequences can also have the same function annotation) but it does not imply orthologous relationship among sequences. Then, even for clusters where all sequences belong to the same functional family (class), this validation method provides weaker claims compared to comparison with known ortholog clusters.

When evaluating results of protein function annotation using ortholog clusters, there is, however, an advantage in evaluating ortholog clusters using independent sources of information. If the target sequence for annotation and the member(s) of the ortholog cluster closest to the target sequence have the same functional annotation, it provides a strong evidence supporting the functional annotation.

Validation using Pfam annotations for sequences

For validating the candidate ortholog clusters from a protein-function perspective, we used the Pfam (Protein Families) [8] annotations for protein sequences. As discussed abstractly earlier, the same Pfam annotation for proteins from different species does not imply orthology but as orthologous proteins are likely to be involved in similar functions they must have similar Pfam annotations. Moreover, when multiple Pfam families are associated with orthologous proteins, they must appear in the same (domain) order in all members of the ortholog family.

While assessing the homogeneity of clusters using a Pfam annotation, sequences were assigned the Pfam family ids of their "close" match(es) (details about the Pfam database and the process of Pfam id assignment are described in chapter 5) in the Pfam database. If the "closest" match found wasn't significantly close, we did not assign any

Pfam annotation to the sequence. Having done so, we computed the homogeneity of the ortholog clusters as the fraction of sequences which shared the same Pfam annotation(s).

Validation using SCOP annotations for sequences

Another independent source of information for validating ortholog clusters is the 3D-structure they assume while performing cellular functions. Due to their involvement in similar functions, orthologous sequences are likely to assume similar 3D-structure and so, 3D-structures annotation for protein sequences can serve as a means to validate ortholog clusters. We have used the SCOP (Structural Classification Of Proteins) [5], which contains a collection of protein fragments along with the 3D structures they assume.

As in the case of validation using Pfam, sequences were assigned the SCOP ids of their "close" match(es) (details about are described in chapter 5) in the SCOP database, if these matches were significantly close. Having done so, we computed the homogeneity of the ortholog clusters as the fraction of sequences which shared the same SCOP annotation(s).

4.5 Conclusion

We have described a novel reformulation of the ortholog clustering problem by using the multipartite graph framework which allows us to simultaneously represent the similarity relationships between sequences across genomes and the evolutionary relationships between the species (using the species-tree). We have also described a linkage function that addresses many issues in ortholog clustering, and reduces the ortholog clustering to a problem of iteratively finding quasicliques using the approach described in the previous chapters. Following this, we have also developed a robust criterion for annotation of a target sequence with an ortholog cluster.

In the later part of this chapter, we developed two kinds of methodologies for validating the ortholog clusters obtained using the proposed method. The first kind of methodology is applicable when the information about the ortholog clusters is known.

For this case, we used various classical statistical coefficients for comparing two clusterings and supplemented these coefficients by a set of indices we developed for a set-theoretic comparison of two clusterings. The second methodology can be applied for validating ortholog clusters discovered on data where we do not have a priori knowledge of ortholog clusters. This involved using independent sources of information such as the functional annotation by Pfam and structural annotation by SCOP for evaluating the homogeneity between the members of ortholog clusters found by our method.

Chapter 5

Data for Ortholog Clustering

In this chapter, we describe the datasets to which we applied our method for constructing ortholog clusters and those for validating the results of the method. As discussed in chapter 4, we have used two types of data : the first is a set of protein sequences with a known "gold standard" ortholog clustering result obtained using expert curation; the second is a set of protein sequences for which we do not have any *a priori* knowledge of ortholog clusters. We describe an example of each kind of dataset. An example of the first kind is the COG database of ortholog clusters, while for the second, we have used the partially complete genomes of four cereals and the complete genome from the model plant species, *Arabidopsis thaliana*. The latter dataset also demonstrates the utility of the proposed method for ortholog clustering on *incomplete genome* data, along with its use for protein function annotation.

Our method takes two graphs as input for constructing ortholog clusters: the first is the multipartite graph describing the sequence similarities across genomes, and the second is the species tree showing the relationships between the species. The edges in the multipartite graph are determined using the sequence search tool BLAST and we will describe the details of determining these similarity edges in this chapter. The second graph is inferred from the species tree (available from the literature) for genomes in the input sequence data; we will describe this input data also.

For validating the ortholog clusters, we determined the functional and structural annotations, for all sequences in the input data, using sources which are independent of the input sequence data. The Pfam [11] and SCOP [5] classifications, respectively, were used for functionally and structurally annotating sequences. These databases along with the procedure used for annotating sequences and their classifications will also be

described.

5.1 Dataset I : 43 genomes from COG with known ortholog clusters

The first dataset for ortholog clustering consists of protein sequences from the 43 complete genomes used for constructing the ortholog clusters in the COG (Clusters of Orthologous Groups) database [82, 83]. It is a publicly available (`http://www.ncbi.nlm.nih.gov/COG/`) trusted resource for ortholog clusters which is widely used for studying evolutionary relationships between prokaryotes. For this study, we used the version which was updated on Feb. 2003, containing the complete genome sequences from 43 species along with 3,307 ortholog clusters. The current (August 2006) recent version contains 66 complete genomes in 4,873 ortholog clusters. The 43 species are mostly (42) prokaryotes with 9 species from the kingdom archaea and 33 bacteria; the only eukaryote is the baker's yeast *Saccharomyces cerevisae*. A description of these 43 species is given in Table 5.1. An advantage of having prokaryotic genomes for ortholog clustering is that they contain very few multi-domain proteins, defined as two or more distinct sequence regions which can independently acquire a 3D-structural conformations, and are individually capable of performing biological roles [57, 76]. Ortholog clustering for multi-domain proteins presents difficulties for any method. Our choice of the 43 genomes allows us to focus on issues beyond parsing the domains in the input protein sequences.

5.1.1 Sequence Data

The 43 genomes together contain 104,101 protein sequences of which 74,059 sequences (approx. 73%) belong to ortholog clusters. As can be seen from Table 5.1, the size of the genomes ranges from 484 for the parasite *Mycoplasma genitalium* to 7,275 for the nitrogen-fixing plant symbiont *Mesorhizobium loti*. Likewise, the percentage of sequences that are in COGs show a large variation - mostly the smaller genomes, usually parasites, have a larger fraction of their genome in the ortholog clusters since presumably they have retained only absolutely essential genes [46].

Kingdom	Organism Code	Name	Sequences in genome	Sequences in COGs
A	A	*Archaeoglobus fulgidus*	2420	1872
A	O	*Halobacterium sp. NRC-1*	2605	1701
A	M	*Methanococcus jannaschii*	1786	1330
A	M	*Methanobacterium thermoautotrophicum*	1873	1388
A	P	*Thermoplasma acidophilum*	1482	1230
A	P	*Thermoplasma volcanium*	1499	1243
A	K	*Pyrococcus horikoshii*	1800	1378
A	K	*Pyrococcus abyssi*	1768	1456
A	Z	*Aeropyrum pernix*	1841	1178
E	Y	*Saccharomyces cerevisiae*	5955	2290
B	Q	*Aquifex aeolicus*	1560	1329
B	V	*Thermotoga maritima*	1858	1527
B	D	*Deinococcus radiodurans*	3187	2226
B	R	*Mycobacterium tuberculosis*	3927	2585
B	R	*Mycobacterium leprae*	1605	1134
B	L	*Lactococcus lactis*	2267	1618
B	L	*Streptococcus pyogenes*	1697	1211
B	B	*Bacillus subtilis*	4118	2870
B	B	*Bacillus halodurans*	4066	2878
B	C	*Synechocystis*	3167	2159
B	E	*Escherichia coli K12*	4275	3414
B	E	*Escherichia coli O157*	5315	3662
B	E	*Buchnera sp. APS*	575	568
B	F	*Pseudomonas aeruginosa*	5567	4392
B	G	*Vibrio cholerae*	3835	2820
B	H	*Haemophilus influenzae*	1714	1542
B	H	*Pasteurella multocida*	2015	1751
B	S	*Xylella fastidiosa*	2831	1589
B	N	*Neisseria meningitidis MC58*	2080	1506
B	N	*Neisseria meningitidis Z2491*	2065	1498
B	U	*Helicobacter pylori*	1576	1096
B	U	*Helicobacter pylori J99*	1491	1075
B	U	*Campylobacter jejuni*	1634	1302
B	J	*Mesorhizobium loti*	7275	4959
B	J	*Caulobacter crescentus*	3737	2628
B	X	*Rickettsia prowazekii*	835	697
B	I	*Chlamydia trachomatis*	895	631
B	I	*Chlamydia pneumoniae*	1054	648
B	T	*Treponema pallidum*	1036	716
B	T	*Borrelia burgdorferi*	1637	696
B	W	*Ureaplasma urealyticum*	614	406
B	W	*Mycoplasma pneumoniae*	689	425
B	W	*Mycoplasma genitalium*	484	381
		Total	104,101	74,059

Table 5.1: Genomes used for constructing the COG database. The first column shows the kingdom (A: archaea, B: bacteria, and E: eukarya) of the species in the third column. Each of the 26 rows corresponds to a distinct lineage considered in constructing COGs. The number of sequences in a genome and the number of sequences in the COG clusters are shown in columns 4 and 5, respectively.

Some of the protein sequences are multi-domain sequences; these are manually divided based on domains. It must be emphasized that the input data for ortholog clustering in COGs are the domain sequences and not the entire protein sequences. The process of dividing into domains, thus, increases the number of input sequences to 108,091 from the initial 104,101 sequences. Division into domains means that COGs contain 77,114 sequences derived from the the original 74,059 sequences.

The construction procedure for *COGs requires that orthologous sequences be present in three or more species*, so the 77,114 domain sequences represent only 71% of all sequences in the 43 complete genomes, belonging to 3,307 COGs in the COG database. More precisely, the COGs are constructed from a group of 26 "hyper-genomes" which are obtained by aggregating the "similar genomes" among the original 43 genomes [82]. The evolutionarily closely related species whose sequences are merged together prior to detecting ortholog clusters are shown in the same rows in the Table 5.1. In other words, all genes from genomes within a group are considered as genes within the corresponding hyper-genome. Furthermore, while constructing COGs similar genes from each of the 26 groups of genomes are grouped together as *paralogs* and considered as single units when producing ortholog clusters in COG. It should be noted that this paralog grouping procedure is not described in detail in COG references; and the intermediate results of such paralog groping between closely related genomes are also not made available. So, instead of using 26 groups (partite sets) for ortholog clustering by our procedure, we used 43 genomes, or 43 partite sets in our method. Since it was hard to find the phylogenetic tree relating the organisms, we estimated the tree based on their taxonomy. Further, we did not consider all the branch points in the taxonomical tree but restricted to five levels: kingdoms, phyla, classes, orders and organisms. This information is available from the COG website (`http://www.ncbi.nlm.nih.gov/COG/`). Furthermore, for computing the distance between the organisms, we considered only the topology of the tree induced by the taxonomy of the organisms and did not consider the actual evolutionary time scales for the organisms.

The sizes of each of the 3,307 COG clusters range from 3 to 806, although more than half (1,793 out of 3,307) of the clusters are small, containing 15 or fewer sequences

and only 277 clusters contain more than 50 sequences. A detailed histogram of COG sizes and the number of species in them will be deferred until the next chapter where we will compare their distributions to those for our ortholog clusters.

5.2 Dataset II : 5 plant genomes without known ortholog clusters along with Rice proteins as targets for annotation

We now describe the data from partially complete cereal genomes for which we do not have prior knowledge of ortholog clusters. They are the set of sequences from the partially sequenced genomes of maize (*Zea mays*), sorghum (*Sorghum bicolor*), wheat (*Triticum aestivum*), and barley (*Hordeum vulgare*). They were obtained from the plant genome database, PlantGDB [21] at `http://www.plantgdb.org/`, which is a publicly available resource for plant comparative genomics and acts as a portal for plant genomic data collected from various public resources.

The sequences available for the four incomplete cereal genomes are shown in Table 5.2. Compared to conservative estimates of 30,000 sequences in a cereal genome, the coverage of these genomes is very sparse. This will limit the number of ortholog clusters. As this could have an adverse impact on functional annotation, we included the complete set of proteins from the model plant *Arabidopsis thaliana* in the input data for ortholog clustering. The complete proteome of *Arabidopsis thaliana* was downloaded from MAtDB at MIPS [71].

Plant	Available number of sequences
Maize	3,318
Sorghum	478
Wheat	1,693
Barley	1,112
Arabidopsis	26,639
Rice	61,250

Table 5.2: Available sequences from the incomplete cereal genomes and *Arabidopsis thaliana*. The protein sequences from the Rice genome were used exclusively as targets for function annotation.

As targets for functionally annotating protein sequences by ortholog clusters, we

chose the Rice genome. The reason for choosing the Rice genome were: (i) being a cereal it is closely related to other genomes used for ortholog clustering, (ii) at the time of carrying out this study, the sequencing for the Rice genome had just been completed and being involved in collaboration with experts in Rice genomics, protein annotation was the next logical step. Sequences for the rice proteome were downloaded from TIGR (`ftp://ftp.tigr.org/pub/data/Eukaryotic_Projects/o_sativa/annotation_dbs/`) were used as targets for annotation. This website contained 61,250 protein sequences. It must be remarked that this is only a draft of the rice genome and very likely contains sequences that are not truly proteins.

For the evolutionary tree relating the above plant species, we focused on studies on evolutionary relationships between members of the grass family and the relationship of grasses to other plants. The species tree used in our study is shown in Fig. 5.1; it is the subtree extracted from a larger species tree relating different grasses as presented in [45].

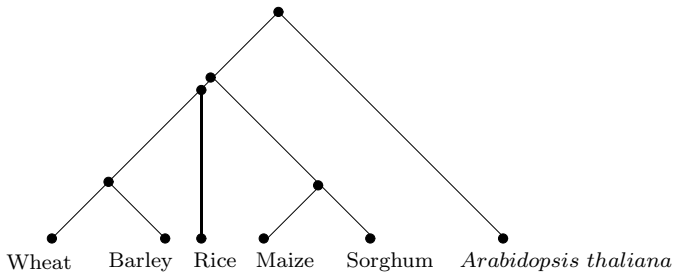

Figure 5.1: The species tree relating *Arabidopsis*, barley, maize, rice, sorghum, and wheat.

5.3 Computing pair-wise sequence similarity

Deciding the similarity edges and their weights requires estimating the pair-wise primary sequence similarity between the protein sequences. These weights can be determined using various sequence alignment or search tools. Accurate estimation of the similarity weights is always desirable, yet can incur high computational costs. So, often, there is a trade-off between accurately estimating these weights and computational resources. We used BLAST for determining the pair-wise sequence similarity which, being an excellent

compromise between accuracy and speed, has become a de-facto standard for large-scale sequence comparisons. We will now briefly describe the BLAST tool, following which the computation of actual similarity weights will be presented.

5.3.1 BLAST

BLAST (Basic Local Alignment Search Tool) [4] is a method for rapid searching of nucleotide and protein databases. It finds regions of local similarity between sequences. The program compares sequences to sequence databases and calculates the statistical significance of matches.

The BLAST algorithm finds similar sequences by breaking the query into short subsequences called "words". The program identifies the exact matches to the query words first (word hits). BLAST program then extends these word hits in multiple steps to generate the final gapped alignments. One of the important parameters governing the sensitivity of BLAST searches is the length of the initial words, also called the word size. Decreasing the word size, increasing the sensitivity at the cost of computational speed. We used the default word size (=3) in the program which has been chosen by experts.

Apart from computational issues in sequence comparison, the choice of the *substitution matrix* plays a critical role. In a substitution matrix, the value a_{ij} indicates a similarity measure for substituting the amino acid i with the amino acid j in an alignment. The values in the matrix are proportional to the probability that amino acid i changes into amino acid j during the biological evolution of protein sequences. Such matrices are constructed by assembling a large and diverse sample of verified pairwise alignments of amino acids. If the sample is large enough to be statistically significant, the resulting matrices should reflect the true probabilities of mutations occurring through a period of evolution. We use the BLOSUM62 (Block Substitution Matrix) substitution matrix. This substitution matrix is derived from the observations of mutations in the amino acids of alignments in the BLOCKS [37, 36] database; furthermore, only the alignments between sequences that share no more than 62% identity (exact amino acid matches) are considered so as to avoid over-weighting observations from

closely related sequences.

E-values and Bit scores

For large scale sequence comparisons, such as our case, BLAST is used in a database search mode. An all-against-all sequence comparison can be carried out by constructing a database of the given sequences, and then searching each sequence against those in the database. For computing the pair-wise similarities between sequences in our data, we used this mode of sequence comparison. The output from such a search contains a list of ranked sequence matches along with their alignments. Depending on a choice of parameter governing the output, the ranking is either based on the e-values or the bit scores. They both indicate the score for the alignment found by BLAST. The e-value (expectation value) measuring the number of different alignments, with scores equivalent to or better than the one observed between the two sequences, that are expected to occur in a database search by chance. The e-value for significant matches are usually very small numbers, close to zero, with lower e-values being strong indicators that the match found is truly significant. By contrast, the bit score is derived from the raw alignment by going over the alignment and adding for each aligned amino acid pair, the score taken from the chosen substitution matrix.

As seen above, the bit-score is not normalized with respect to the length of the sequence or other sequences in the databases. While the bit-score keeps growing with an increase in the length of the aligned fragments of a pair of sequence, the e-value reaches 0, indicating a very strong match, and cannot improve any further. Since e-value is an estimate of a match occurring by chance, it fails to discriminate between two excellent pair-wise alignments. On the other hand, the bit score, being the raw sum of substitution scores over the aligned positions, can discriminate between such alignments. Moreover, for short sequences the e-value never reaches 0, even for identical sequences, but the bit score does attain its best possible value for such pairs.

Due to the reasons discussed above, we used the e-value as a threshold for filtering out spurious matches. For constructing the multipartite graph of similarities between

the sequences, we used the bit-score as the weight of similarity [1]. Due to the nature of BLAST searches, it is possible to obtain asymmetrical scores during pair-wise sequence comparison, i.e., the similarity w_{ij} between sequence i and j may not be same as the value w_{ji}, the similarity between j and i. Since our multipartite graph is undirected, we forced the weights to be symmetric by using $w_{ij} = w_{ji} = \max(w_{ij}, w_{ji})$. As discussed in the previous chapter, we considered only a subset of the top hits for any given protein sequence. To be precise, if a sequence i in genome $g(i)$ has the sequence j as its best match in genome $g(j)$ with score θ, we considered all those matches for i from species $g(j)$ which had a bit score larger than $\theta/2$. As explained earlier, the idea behind this is to avoid low-scoring spurious hits (which would most likely be paralogs) without filtering out potential orthologs.

We used BLAST, mostly with the default parameters. Specifically, we used `blastall -p blastp -a 4 -d TARGETDB -e 1 -F "C;S" -i INPUT-SEQ-FILE`, where '`-a 4`' dictates blastp (the amino acid sequence comparison program) to be run on 4 processors, '`-e 1`' sets the e-value threshold to 1, '`-F "C;S"`' forces filtering out [2] coil-coil and low complexity regions using the SEG [92] filtering program while searching for sequences in the file INPUT-SEQ-FILE against those in the sequence database TARGETDB.

5.4 Data for validating ortholog clusters

The primary validation test for ortholog clusters extracted from Dataset I is the reference set of existing COG ortholog clusters. Additional independent sources of data used for evaluating the homogeneity of ortholog clusters extracted from input sequences data where we do not have knowledge of ortholog clusters are those used for functionally and structurally annotating the sequences in the input data. Since we will be using

[1] We have also used $\lceil -\log_{10} \text{e-value} \rceil$ as an integer similarity weight, but our experimental results using the bit score were better. The comparison of these results is not reported here and we will present results using the bit score as the measure of similarity weight.

[2] While comparing sequences, one has to watch out against high scores resulting from regions that are commonly present across various unrelated sequence families. These can be the coil-coil regions and/or the low-complexity regions including the runs of a single amino acids, short-period repeats, and more subtle over-representation of one or a few residues. To avoid spurious matches based on such regions, we used SEG [92] for masking the low-complexity regions and the coil-coil filter by Lupas et al. [50].

these annotations for evaluating ortholog clusters, the quality of the sources of such annotations is a key issue. We have selected Pfam and SCOP, which are popular and trusted sources for such annotations due to the expert curation on results of state-of-the art automatic methods and their updates reflecting the latest available information.

5.4.1 Pfam database

The Protein families, or Pfam, database is a collection of protein families and domains which is regularly updated [8, 9, 10, 11, 75, 76] and mirrored at various websites across the globe, including `http://www.sanger.ac.uk/Software/Pfam/`. It contains multiple protein alignments and profile-HMMs of Pfam families. Pfam is a semi-automatic protein family database, aiming to be comprehensive as well as accurate. According to the website "hopefully Pfams represent some evolutionary conserved structure which has implications for the protein's function".

Pfam is actually composed of two sets of families: Pfam-A which are families based on curated multiple alignments and Pfam-B families which are based on an automatic clustering of the rest of SWISSPROT and TrEMBL [13] derived from the PRODOM database [14]. The families in Pfam-A are regions of proteins that are predicted by the Pfam collection of hidden Markov models (HMMs) to belong to a family. These are strongly trusted matches to the family and are very unlikely to be false matches. As Pfam-A is more reliable, we will be focusing on annotations using Pfam-A only.

We have used the families and sequences from the Pfam-A release 12.0 (the latest available at the time of these studies). It contained 898,590 domain sequences divided into 7,316 Pfam functional families. The latest (as of Feb. 2006) release 19.0 of Pfam-A has 8,183 families which together contain 2,345,429 domain sequences.

5.4.2 The SCOP database

The Structural Classification Of Proteins (or, SCOP) [5] hierarchical classification of protein domains has been constructed manually by visual inspection and comparison of structures, but with the assistance of a battery of automated tools to make the task manageable and help provide generality. This database aims to provide a detailed and

comprehensive description of the structural and evolutionary relationships between all proteins with known structures. Such protein sequences are available at the Protein Data Bank, PDB [12]. SCOP provides a broad survey of all known protein folds, detailed information about the close relatives of any particular protein, and a framework for future research and classification.

SCOP Classification

Proteins are classified to reflect both structural and evolutionary relatedness. Many levels exist in the hierarchy, but the principal levels are family, superfamily and fold, described below (adapted from `http://scop.berkeley.edu/intro.html`).

1. **Family:** *Indicates clear evolutionarily relationship.* Proteins clustered together into families are clearly evolutionarily related. Generally, this means that pairwise residue identities between the proteins are 30% or higher. In some cases similarity of functions and structures provides even more definitive evidence of common descent in the absence of high sequence identity.

2. **Superfamily:** *Indicates probable common evolutionary origin.* Proteins having low sequence identity, but whose structural and functional features suggest the possibility of a common evolutionary origin are placed together in superfamilies.

3. **Fold:** *Indicates major structural similarity.* Proteins are defined as having a common fold if they have the same major secondary structures in the same arrangement and with the same topological connections. Different proteins with the same fold often have peripheral elements of secondary structure and turn regions that differ in size and conformation. Proteins placed together in the same fold category may not have a common evolutionary origin: the structural similarities could arise from just the physical and chemical properties of the proteins favoring certain topologies.

SCOP is available as a set of tightly linked hypertext documents which make this large database comprehensible and accessible. It can be found at the URL : `http://scop.mrc-lmb.cam.ac.uk/scop/`. We have used the version 1.63 which was released

in March 2003, and was built from 18,946 PDB entries; it contains 49,497 domains divided into 765 folds, 1,232 superfamilies and 2,164 families. The latest version 1.69, released in October, 2004, is based on the 25,973 PDB entries. It includes 70,8059 domains divided into 945 different folds divided into 1,539 superfamilies, further divided into 2,845 families.

5.4.3 Annotating sequences with Pfam and SCOP

The Pfam and SCOP annotations for sequences in the input data are used for validating the ortholog clusters. We now describe the method for associating annotations with the input sequences. Since we followed the same method of annotation using Pfam and SCOP, we will be describing the general method in this section. The Pfam and SCOP databases are not only collections of sequences but also contain the profile HMMs [24] for each family. Profile hidden Markov models (profile HMMs) are statistical models of the primary structure consensus for a sequence family. They are built from the multiple alignments of the members of the family and are statistical descriptions of the consensus of a multiple sequence alignment. They use position-specific scores for amino acids (or nucleotides) and position-specific scores for opening and extending an insertion or deletion. By doing so, profiles capture valuable statistics about the degree of conservation at various positions in the multiple alignment while varying the size of gaps and permitting insertions. This property gives them an edge over the traditional pairwise alignment techniques that use position-independent scoring parameters.

Traditional pairwise alignment (such as, BLAST, FASTA [64], or the Smith-Waterman algorithm [72]) uses position-independent scoring parameters. This property of profiles captures important information about the degree of conservation at various positions in the multiple alignment, and the varying degree to which gaps and insertions are permitted.

For annotating the input sequences, we searched them against the profile HMMs available from the Pfam and SCOP databases using the HMM package HMMER [24]. HMMER includes programs for creating HMM profiles, calibrating them, and searching against existing profile HMMs. For our case, the calibrated profile HMMs are already

available from the databases, so we used the `hmmsearch` program from this package. Like the pair-wise searches by BLAST, HMMER searches also indicate the strength of matches found using the e-value and the raw bit-score. We used the e-value cutoff 1.0 for our searches.

For multi-domain protein sequences, it is possible to get multiple matches from different families. In such cases, usually the different families correspond to the different regions of the primary sequence. So, we annotated each region of a sequence with the match corresponding to the strongest observed e-values. As the order in which different domains appear in the sequence is important, we stored this information also. Domain order (also called domain architecture) is important because orthologs are likely to be involved in the same function and hence should preserve the order in which different domains appear in the sequence.

5.5 Conclusion

We have described two data sets for ortholog clustering: one containing 43 complete genomes with known ortholog clusters and the other containing mostly the partially completed genomes from the cereals. Following the reasons for choosing these datasets, we described the procedure to calculate the similarities between sequences and the species tree for calculating the distance between the genomes. The sources for functionally and structurally annotating sequences - the Pfam and the SCOP databases, respectively, have been described. Finally , the methods used for associating the Pfam and SCOP annotations were discussed.

Chapter 6

Ortholog Clustering: Results

6.1 Introduction

We have tested and evaluated the multipartite graph clustering method on the two datasets described in chapter 5. This chapter presents our ortholog clustering results on these datasets and their validation using the criteria developed in chapter 4.

The organization of this chapter is as follows. Section 6.2 describes the ortholog clustering results on the 43 complete genomes on which the expertly curated ortholog clusters, COGs, are constructed. This section also presents detailed statistics and a comparative analysis of the results. In section 6.3 we show that using the evolutionary tree information for scaling the observed sequence similarities produces better ortholog clusters as seen by an increased correlation with COG ortholog clusters. The results of applying the method for ortholog clustering to the partially completed plant genomes and then using those clusters for functionally annotating the sequences from the rice genome are described in section 6.4.

6.2 Ortholog clusters for 43 complete genomes (COG data)

For the 43 complete genomes data, used for constructing the COGs, the multipartite graph clustering method produced 25,083 multi-gene sequence clusters and 13,202 singletons. According to their size, the clusters can be divided into 16,806 clusters of size 2 and the remaining 8,277 clusters that contain at least 3 sequences. As the singletons, by definition, cannot be ortholog clusters and are removed from further analysis.

We briefly characterize 16,806 simple ortholog clusters that contain 2 sequences and

Cluster 7 contains 1 COG			
Cluster has 46 genes (3 arcaeal, 43 bacterial, 0 eukaryotic) from 35 organisms (3 archaea, 32 bacteria, 0 eukaryote) spanning 20 lineages in 2 kingdoms. COG0187 has 61 genes (4 archeal, 56 bacterial, 1 eukaryotic) from 38 organisms (4 archaea, 33 bacteria, 1 eukaryote) spanning 23 lineages in 3 kingdoms. This COG is associated with Pfam family PF00204 and SCOP superfamily "e.11.1" which is annotated as: DNA gyrase (topoisomerase II) B subunit			
Kingdom	GroupId	Organism	COG0187
[A]	2	Halobacterium sp. NRC-1	1 (1)
[A]	4	Thermoplasma acidophilum	1 (1)
[A]	4	Thermoplasma volcanium	1 (1)
[B]	8	Aquifex aeolicus	1 (1)
[B]	9	Thermotoga maritima	1 (1)
[B]	10	Deinococcus radiodurans	1 (1)
[B]	11	Mycobacterium leprae	1 (1)
[B]	11	Mycobacterium tuberculosis	1 (1)
[B]	12	Lactococcus lactis	2 (2)
[B]	12	Streptococcus pyogenes	2 (2)
[B]	13	Bacillus halodurans	2 (2)
[B]	13	Bacillus subtilis	2 (2)
		⋮	

Table 6.1: An example ortholog cluster. The table shows the relationship of the cluster to COGs including its Pfam and SCOP annotations for member sequences.

analyze the remaining 8,277 clusters in detail. Some clusters that have 3 or more sequences contain sequences from only 2 organisms. These clusters were removed leaving 7,701 clusters, each of which contains sequences from at least 3 organisms. They are considered as the candidate ortholog clusters produced by our method and are compared with COGs.

The complete list of 25,083 clusters including various additional features used in their analysis and validation can be accessed at the URL `http://www.research.rutgers.edu/~vashisht/orthologClusters/`. An example (partial) cluster is shown in Table 6.1.

Every cluster is presented as a table that describes all COGs that overlap with the cluster. To facilitate a comparison with the COGs at the gene level, these tables also contain the distribution of organisms in a cluster, and also in all the COGs that overlap with our cluster. A rationale for providing this information is that orthologous

relationships among genes are based on speciation events, and therefore when analyzing ortholog clusters it is important to analyze clusters for the evolutionary relationship between different organisms present in a cluster. The grouping of organisms into 26 groups, provided in COG, is also shown, since it enables us to characterize clusters on the basis of relationships among organisms featured in a cluster. To facilitate an independent validation of the ortholog clusters, based on the functional family and 3D structure annotation of the member sequences, the Pfam family and SCOP superfamily annotation (based on HMM comparisons) are also shown.

Each table presents information about COGs that intersect with a cluster, including genes contributed by each organism. These tables also facilitate a detailed view of individual clusters and provide annotations of MPC_clusters based on COG annotations. Each table contains information related to the number of organisms/kingdoms that contribute genes to a cluster along with the same information for the COGs that intersect with the cluster. When a cluster contains sequences from a large number of organisms spanning multiple kingdoms, it makes sense to analyze clusters for the presence of various kingdoms and for the distribution of various lineages to give a more complete picture of evolutionary relationships. We also present the Pfam family and SCOP superfamily associated with the cluster.

6.2.1 Analysis of clusters containing 2 sequences

An interesting observation relating to 16,806 small clusters with only 2 sequences is that the number of sequences contributed by an organism has a wide range: such clusters contain sequences from all organisms but with a large variation in the number of genes contributed by organisms. The bacterium *Mesorhizobium loti* has 2195 sequences in these clusters, out of the 7,275 genes in its genome. Likewise, *Mycobacterium tuberculosis* (genome size: 3927 sequences) has 1,499 sequences in such clusters. At the other end of the range, *Buchnera APS* has only 85 sequences in these clusters from 575 genes in its genome. Some species with smaller genomes also contribute a significant fraction of their genes in clusters of size 2. For instance, *Mycoplasma pneumoniae* contributes 305 sequences from 689 genes in its genome. So, the number of sequences in these

clusters is not correlated to the genome size of the organism.

An interesting aspect of analysis of size 2 clusters has to do with the relationship between the two organisms that contribute sequences to a cluster. We considered the grouping of organisms, provided in COGs, to explore the correlation of relationship among organisms, and co-clustering of sequences from different organisms. There are 6,999 clusters (42% of clusters), each of which contains sequences from organisms in a single group from the 26 groups in COG. To assess conservation among sequences, in clusters of size 2, we used their Pfam and SCOP annotations, which were obtained through an HMM search performed using HMMER [24]. There are 3,633 clusters whose sequences are not associated with any Pfam family, and 6,410 clusters that do not contain sequences related to any SCOP superfamily. Among the 16,806 clusters of size 2, 10,052 clusters have both sequences associated with a single Pfam family. Similarly, 8,146 clusters contain sequences which have *the same SCOP* superfamily classification.

Every cluster of size 2 is an element of the family of clusters, C, and occupies a fixed position i in C determined by the iteration at which it was extracted (section 4.1). We found a strong correlation between the position of a cluster in C and how well the cluster is specified by Pfam family and SCOP superfamily: for a cluster with small index i, there is high probability that it is related to a single Pfam family or SCOP superfamily.

6.2.2 Analysis of clusters containing at least 3 sequences

For purposes of analysis, we divided the 8,277 clusters that contain at least 3 sequences, into two groups: 576 clusters that contain sequences from only 2 organisms, and 7,701 clusters that contain sequences from at least 3 organisms. The first group contains 2624 sequences. It is interesting that most clusters (487) from this group have size 4. There are 484 clusters (84% of the total) which contain sequences from organisms within a group in the 26 groups described in COGs. There are 442 (81%) clusters whose sequences belong to a single Pfam family. Comparison with SCOP reveals that 298 (51%) clusters contain sequences which belong to a single SCOP superfamily.

The largest cluster (cluster id 10861), in the set of 576 clusters, has 16 sequences

from bacterium *Mycobacterium tuberculosis* and 15 sequences from *E. Coli*. These are consistently associated with a single Pfam family and SCOP superfamily, both of which are related to the annotation *transposase*. Another cluster (cluster id 6974) contains 29 sequences, consistently annotated as *putative transposase*.

The third largest cluster (cluster id 452) has 18 sequences which belong to two very closely related organisms: the pathogenic bacteria *Helicobacter pylori 26695* and *Helicobacter pylori J99*, both of which cause peptic-ulcers. Sequences in this cluster are not associated with any Pfam family or any SCOP superfamily. The score value for this cluster is very high indicating that these sequences are highly similar. A further investigation using database searches reveals that this set of genes is highly related and enhance the infective power of these bacteria.

In the set of 7,701 clusters with sequences from at least 3 organisms, only 61 clusters contain sequences that are related to different Pfam families. There are 541 clusters, with no sequence associated with a Pfam family and 5,901 clusters with all their sequences annotated with a single Pfam family. Comparisons with SCOP annotations show that only 52 clusters contain sequences that are related to different SCOP superfamilies, and 4,946 clusters have all their sequences annotated by a single SCOP superfamily, while sequences in 1,884 clusters could not be annotated with any SCOP superfamily. As in the case of clusters with size 2, we found that clusters produced earlier in the iterative generating process are those highly correlated with a single Pfam family, or SCOP superfamily annotation.

6.2.3 Comparison with COGs

In the following analysis of results, we compare our clusters with COG clusters. As stated in chapter 5 earlier, since the construction procedure for COG clusters requires that ortholog clusters contain sequences from *at least three organisms*, we restrict the comparison to 7,701 of our clusters which contain sequences from at least 3 different organisms. We call these clusters MPC_clusters.

The 7,701 ortholog clusters contain 51,134 sequences and the relationship of these sequences to those in the COGs is shown in Fig. 3a. Among the 7,701 of our ortholog

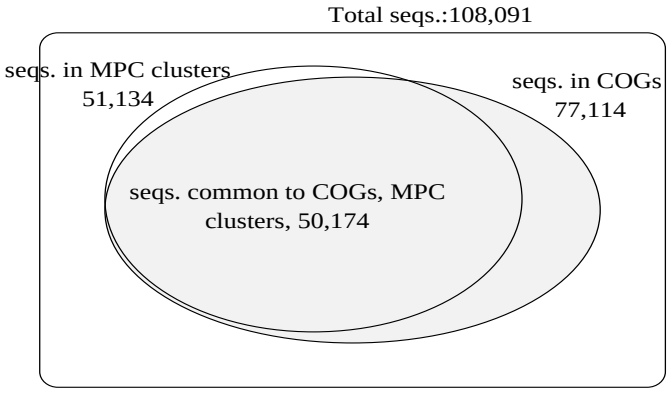

Figure 6.3: Relationship between sequences recognized as COGs and our ortholog clusters

clusters, 7,101 (92%) contain sequences from a single COG, but there are 7,474 (97%) clusters whose sequences are consistent in at least one of the annotations - Pfam, SCOP or COG membership. The set of all sequences in our ortholog clusters are mostly contained (98%) in the set of all sequences in COGs. There are 104 COGs that are found to match our clusters exactly.

As a consequence of the strong criterion used in the MPC method, the ortholog clusters produced are much smaller, and tighter. This is apparent from a comparison of the distribution of sizes of MPC_clusters and COGs shown in Fig. 6.4, which indicates that a large number of MPC_clusters have cardinality between 3 and 10. Smaller cluster sizes limit the number of organism in MPC_clusters, and on average MPC_clusters contain genes from fewer organisms compared to COGs (see Fig. 6.5).

The detailed results for our clusters are available at the URL `http://www.research.rutgers.edu/~vashisht/orthologClusters/` . Each MPC_cluster is described in a table that describes all COGs that overlap with the cluster. To facilitate comparison with COGs at the gene level, these tables also contain the distribution of organisms in a cluster, and in all COGs that overlap with an MPC_cluster. A rationale for providing this information is that orthologous relationships among genes are based on speciation events, and therefore when analyzing ortholog clusters it is important to analyze clusters for the evolutionary relationship between different organisms featured in a cluster. The grouping of organisms into 26 groups, provided in COG, enables us to characterize clusters on the basis of relationships among organisms featured in a cluster, so this

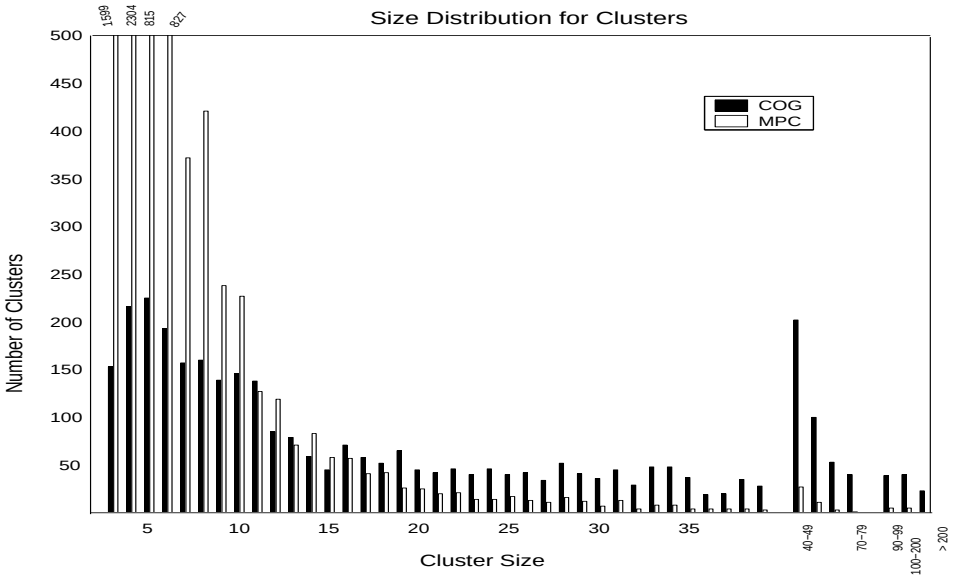

Figure 6.4: Cluster size distribution.

information is also provided in the tables. We also describe the Pfam family and SCOP superfamily annotation, based on HMM comparisons, for each COG that overlaps with a cluster.

To estimate the association between clusters in COG and clusters extracted by our method, we calculate several statistical parameters. We use the *rand index* [67] and its normalized variant the *adjusted rand index* [40] to quantify the relatedness of the two classifications. These indices involve counting the number of agreements between the two classes based on how each pair of elements in the underlying set is assigned to groups (clusters) in the two classification schemes. For comparison with COGs, we considered only the 7,701 clusters as ortholog clusters and obtained a value of 0.804 for the rand index which can be interpreted as : more than 80% of all pairs of sequences in our clustering results agree with ortholog clusters in COG. When the number of agreement pairs are normalized by the expected value of the number of agreement pairs in a random classification (the adjusted rand index) we obtain a value of 0.665, which again suggests a high degree of correlation between the two classifications.

For the average index measuring the fraction of the number of sequences in our clusters that do not belong to any COG cluster, we obtain a value of 0.021, suggesting that it is very rare that sequences in an MPC_cluster do not belong to any COG. The

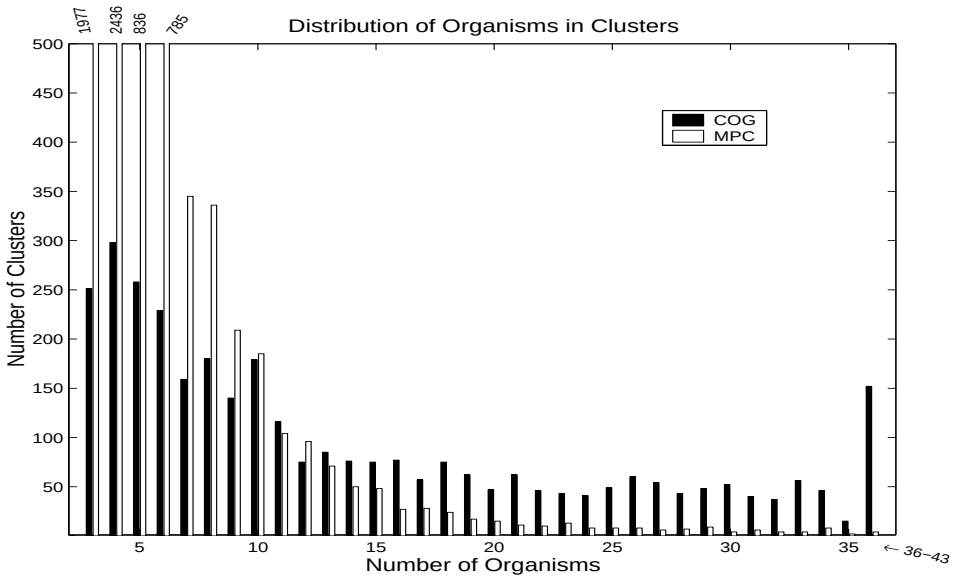

Figure 6.5: Number of organisms in clusters.

average number of different COGs that overlap with an MPC_cluster, is 1.087 which indicates that most our clusters contain sequences from a single COG cluster. The average of the number of MPC clusters overlapping with a COG clusters is 2.463. This is consistent with the fact that there are 7,701 MPC clusters and 3,307 COG clusters.

Discussion of Examples

When all members of a cluster belong to a single COG, it is clear that the MPC cluster successfully captures the orthologous relationship by COG. From this perspective, it is interesting to analyze the clusters that overlap with more than one COG, or contain sequences that do not belong to any COG. So, the subjects of the following discussion are, mostly, the clusters that contain sequences from more than one COG.

We begin our analysis with some of the largest MPC clusters. The largest cluster (cluster id 2352) contains 170 sequences and overlaps with 6 different COGs, yet all the sequences are annotated by the keyword *ABC transporter, ATPase component.* The score value for this cluster is large, and all sequences in these COGs are related to the Pfam family PF00005, so it can be said that these sequences share an orthologous relationship. The second largest cluster (cluster id 6975) has 134 sequences all of which are members of a single COG; they are identical in their Pfam and SCOP annotations.

The third largest cluster (cluster id 7182) contains 128 sequences which belong to two different COGs with annotations *GDEF domain* and *CheY-like receiver domain and GDEF domain*. The Pfam annotation, PF00990, is shared by all sequences in this cluster. Based on the annotation for COGs, it is clear that sequences in these COGs have a common domain, making it difficult to distinguish the sequences using pair-wise similarities alone. These two clusters are representative of other large clusters whose members belong to more than one COG, and demonstrate that sequences in such clusters are indeed homologous and capture important signals related to biological function.

There are 104 clusters that exactly match the COGs. Such clusters are mainly small clusters (90 clusters have size less than 10) and correspond to genes that are specific to a particular kingdom of life (80 clusters comprise genes from a single kingdom and 24 from 2 kingdoms). Annotation for clusters that contain genes from 2 kingdoms suggest the involvement of these genes in ecological niches in the corresponding organisms, regardless of their taxonomical position. The cluster with id 472, matches COG1980 that has annotation *Uncharacterized Archaeal conserved domain*, and contains 8 archaeal genes and 1 bacterial gene. Another such example is cluster id 1232, which matches with COG1031 (annotation: *Uncharacterized Archaeal conserved domain*), contains 6 archaeal genes and 1 gene from bacteria *Thermotoga maritima* which cohabits with other archaea in extreme conditions, and is known to have acquired its considerable genome through horizontal transfer [58].

MPC clusters that overlap with more than one COG might naively be considered as spurious, but a deeper analysis of such clusters proves otherwise - sequences in these clusters share important biological signals. There are 78 large MPC clusters (size $>= 20$) that correspond to 2 or more COGs. Despite this, the annotations for COGs are highly related, moreover, the Pfam and SCOP annotations for these COGs coincide. In many cases, these clusters correspond to COGs whose annotations have keywords like ABC transporters (ATPases and GTPases), dehydrogenases, regulatory genes for transcription, DNA Binding domains. These annotations are related to some of the largest groups of paralogs, and are known to be hard to distinguish based on pair-wise

sequence similarity scores [84]. An instance of these clusters (cluster id 98) contains 29 genes from COG0055 (annotation: *F0F1-type ATP synthase beta subunit*) and 29 genes from COG0056 (annotation: *F0F1-type ATP synthase alpha subunit*) - both of these COGs contain 30 genes and the two genes missing from our cluster are paralogs from the genome of *Ureaplasma urealyticum*.

There are 58 clusters that contain sequences from a single COG along with sequences that do not belong to any COG. The dense clusters (top 40%) in this group, contain mostly genes from 4 or more organisms representing 2 (or, at most 3) lineages. Furthermore, within each cluster, genes that are a part of COG belong to organisms in a single lineage; examples of such clusters include Clusters 4698, 4733, 4786 etc. The SCOP superfamily annotation for all sequences in cluster 4698 is *a.118.8* which is also the SCOP annotation associated with the COG that intersects with cluster id 4698. Similarly, the other two MPC_clusters are also consistent in their respective SCOP annotations. The structural conservation of genes present in these MPC_clusters strongly suggests an orthologous relationship within each cluster, which could not be captured by COGs.

In [1] it was discovered that some COGs may have discrepancies in orthologous relationships. One such COG mentioned in [1] is COG0050. This COG is related to transcription factors *tufA/tufB* in bacteria and its counterpart in Yeast mitochondria. However, the COG0050 contains 6 other genes from yeast, none of which can be considered as an ortholog - evolutionarily or functionally. Our cluster 1 which is related to COG0050 contains all the bacterial genes and one yeast gene - consistent with the findings in [1]. Other clusters also have been analyzed in detail in [1], unfortunately, we cannot compare our results with theirs because they studied clustering on an older version of COG based on only 21 genomes.

The MPC clusters have a tendency to recognize orthologous clusters from closely related organism. For instance, there are 10 MPC clusters that contain genes from all organisms in exactly 2 kingdoms, however, most of these (7) are fragments of COGs that contain genes from all the 3 kingdoms and among the remaining, 2 clusters match COGs exactly. An observation related to clusters which are proper subsets of COGs is

that such COGs contain many paralogs from an organism, but MPC clusters tend to recognize a fewer number of paralogous genes.

6.3 Effect of phylogenetic tree in constructing ortholog clusters

The results described in the previous section were obtained without using the phylogenetic tree information. We presented the motivation for using this information (see section 4.2 related to formulation of ortholog clustering problem) as the improvement of ortholog clustering. In this section, we present the ortholog clustering results obtained by using the phylogenetic tree (described in section 5.1) for rescaling the observed sequence similarities. To demonstrate the effect of using the phylogenetic information, a comparison of these clusters with those described in the previous section and obtained without using the phylogenetic tree is also presented.

Recall that in section 4.2, we had proposed the use of a gene-specific and genome-specific similarity threshold for removing the spurious (non-orthologous) gene similarities without removing the potential orthologs. As described earlier, this has the advantage of making the input graph sparse without adversely affecting the ortholog clustering results. This is likely to improve the clustering process as the modified input similarity graph will contain edges which are more likely to correspond to orthologous relationships. We compared the ortholog clusters obtained using this option with those obtained using the phylogenetic tree but without this threshold. In all of our comparisons, we used COGs as the underlying benchmark.

6.3.1 Ortholog clusters using phylogenetic tree

The multipartite graph clustering method produced 21,970 clusters, of which 10,332 were singletons and 5,153 contained 2 sequences. For our analysis and comparisons, we considered the 5,878 clusters that contain sequences from at least 3 organisms, and we call them MPC_Phyl clusters. To study the effect of using the phylogenetic tree, we compare the MPC_Phyl and MPC clusters and for analyzing overall quality of MPC_Phyl clusters we compared them with the COGs.

The 5,878 MPC_Phyl clusters together contain 77,467 sequences. Of these 77,467 sequences, 70,262 sequences are also classified as orthologs by COGs. In comparison, the 7,701 MPC clusters (those obtained without using the tree) contained 51,134 sequences of which 50,174 were also classified as orthologs in COGs. This simple comparison suggests that MPC_Phyl clustering was better at recognizing more COG orthologs. At the level of clusters also, the MPC_Phyl results matched the COGs better, as shown by comparisons in Fig. 6.6 below.

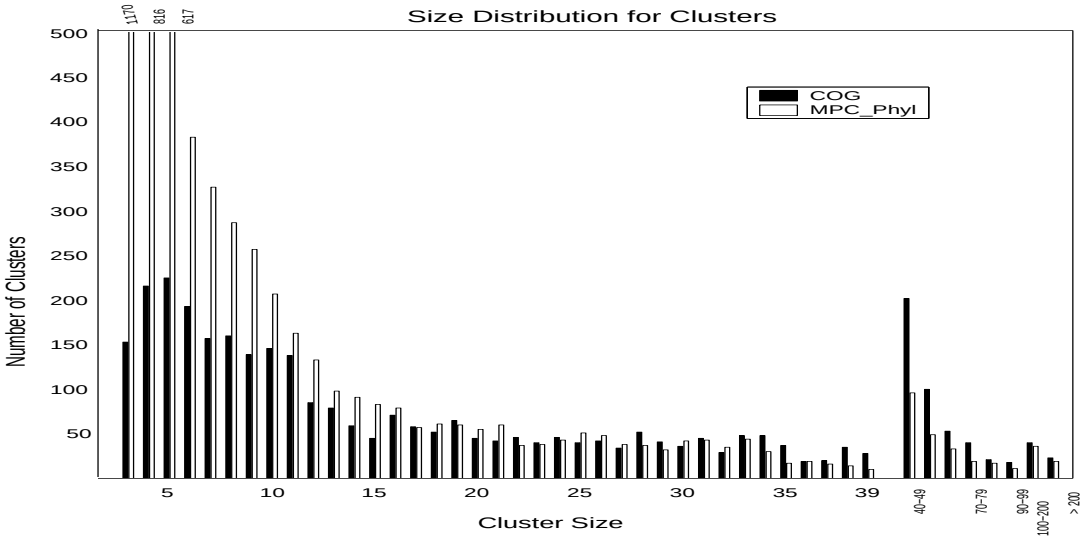

Figure 6.6: Cluster size distribution.

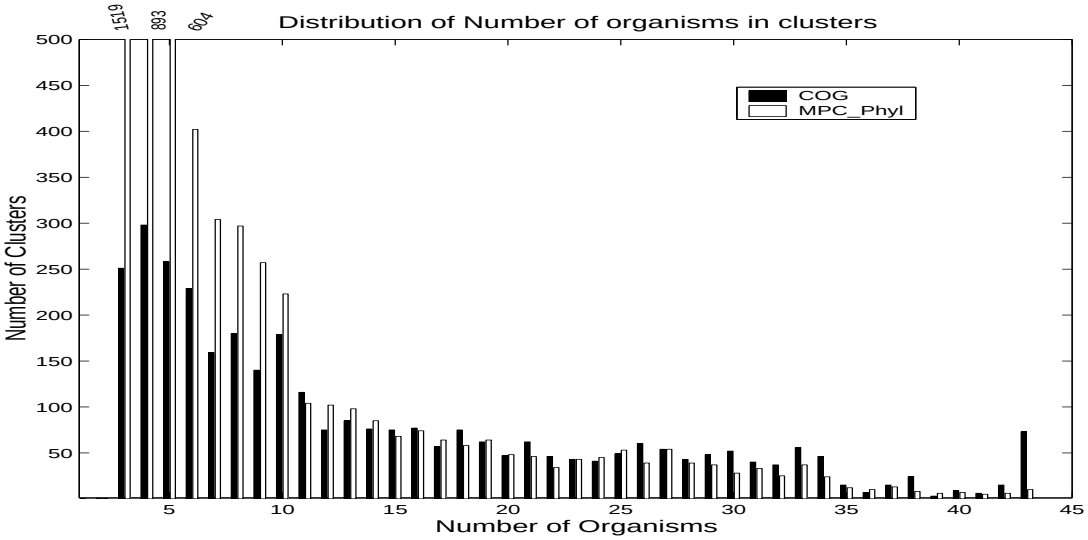

Figure 6.7: Distribution of organisms in clusters.

The distribution of size and the number of organisms in the MPC_Phyl clusters along against the same distributions for COGs are shown Fig. 6.6 and 6.7, respectively. A comparison with MPC clustering can be done by comparing Fig. 6.6 with Fig. 6.4 and Fig. 6.7 with Fig. 6.5. Both of these figures consistently indicate that MPC_Phyl clusters are larger in size compared to the MPC clusters. Such increase in size is a desired effect, since the distribution of MPC_Phyl clusters and COGs almost coincide for the medium range size of clusters (sizes 10 to 35 sequences), moreover, the number of small clusters is significantly less. There is a corresponding increased agreement in the distribution of the number of organisms in the clusters: the distribution for the number of organisms for clusters containing sequences between 10 and 35 organisms is almost the same for COGs and MPC_Phyl clusters. This again is a positive effect of biasing the observing sequence similarities by the phylogenetic tree, as the earlier MPC clusters consistently contained sequences from many fewer number of organisms.

In the following comparative analysis, we show, by using set-theoretic and statistical coefficients, that the above aggregation effect is indeed an improvement in recognizing the orthologous relationships. For a detailed comparison between the MPC and MPC_Phyl clusters, we relied on their relationship to the 'gold standard' COG clusters. Essentially, a clustering result which has a higher correlation with the COG clusters can be considered better. For inferring this, we used comparison measures described in section 4.4. The results for various set-theoretic comparisons, statistical coefficients and our indices are shown in Table 6.2.

The single most important comparison criterion is the number of clusters that exactly match the COG clusters. This number has increased from 104 for MPC clusters to 587 for MPC_Phyl clusters. The number of COGs *exactly partitioned* into MPC clusters is 221 while this number for the MPC_Phyl clusters is 1,324. For measuring homogeneity withing clusters, an important set-theoretic parameter is β_2, which is the average number of COGs in clusters. The value of β_2 for MPC clusters is 1.1 and that for MPC_Phyl clusters is 1.3. This indicates that, on average, MPC clusters overlap with fewer COG clusters.

We also computed statistical coefficients for estimating the overall agreement of the

Comparison Measure	COG	MPC	MPC_Phyl	MPC_Phyl_Cutoff (cutoff = 0.9)
Singletons		13,202	10,322	16,130
Pairs		16,806	5,153	8,269
Ortholog Clusters	3,311	7,701	5878	7,283
Seqs in Ortholog Clusters	77,114	51,134	77,467	65,913
Exact Match		104	587	540
COGs covered		2,590	3,231	3,200
COGs partitioned into clusters		221	1,324	894
Clusters overlapping 1 COG		7,203 (93%)	4,687 (80%)	5,859 (81%)
Clusters overlapping 2 COGs		396 (6%)	807 (13%)	980 (13%)
Clusters overlapping 3 COGs		70 (1%)	250 (4%)	330 (4%)
α_1		0.72	0.86	0.83
β_1 (Avg. frac. of sequences from a cluster contained in $C\bar{O}G$)		0.02	0.17	0.09
β_2 (Avg. #COGs in a cluster)		1.08	1.38	1.28
γ_1 (Avg. frac. of sequences in a COG that are not part of any cluster)		0.47	0.12	0.18
γ_2 (Avg. #clusters in a COG)		2.4	2.0	2.4
Cramer's V		0.50	0.746	0.76
Rand Index (frac. of pairs co-clustered with COGs)		0.80	0.945	0.912
Adjusted Rand Index		0.31	0.476	0.47

Table 6.2: Comparison of ortholog clusters obtained by applying our clustering method to different transformations of the input multipartite graph. The results shown in the second column were obtained by using the multipartite graph and ignoring the phylogenetic relationships. The third column shows ortholog clustering results obtained by rescaling inter-genome gene similarities according to the phylogenetic tree. The results in the fourth column were obtained where we considered the phylogenetic relationships as well as used the gene-specific genome-specific cutoffs in constructing the multipartite graph.

two clusterings with the COGs. For *Cramer's V* coefficient (which ranges from 0 to 1, with 1 implying an exact match) the value for MPC clusters was 0.50 while that for MPC_Phyl clusters was 0.75 a considerable improvement. For a more interpretive comparison we used the *Rand index*, which is the fraction of all pairs of sequences that are clustered identically in two different clustering. We found that MPC clustering and COGs agree in 80% of all pairs but MPC_Phyl clustering and COGs are in much higher pair-wise agreement with a value of 95%. Thus, the stastical coefficients suggest that rescaling the observed similarities based on the phylogenetic tree improves the ortholog clustering results making the more correlated with the manually curated results in COGs.

The MPC_Phyl approach is an improvement over the MPC method in most aspects as shown by the above comparison, however, it suffers from a drawback which is not captured by any of the coefficients above; it produced a few (5) very large clusters (size > 400) that overlapped with many ($10 - 111$) COGs. A manual inspection of relationships inside these clusters showed that this was an effect of considering weak pair-wise similarities between sequences. Recall that we have not used any threshold to filter out non-orthologous sequence similarities, but an ortholog problem specific cutoff was proposed in section 4.2.

6.3.2 Effect of gene-specific genome-specific cutoff

We present results of using gene-specific genome-specific similarity cutoff (described in section 4.2), while constructing the input multipartite graph for ortholog clustering. In this setting, for every gene we ignored those hits from a genome whose similarity value was less than 90% of the similarity value for the best hit in the same genome. We also rescaled the pair-wise gene similarities using the phylogenetic tree. The results are shown in column 4 (indicated by MPC_Phyl_Cutoff) of Table 6.2.

Using the phylogenetic tree and the gene-specific genome-specific cutoff (90%), the

multipartite graph clustering produced 7,283 clusters (called the MPC_Phyl_Cutoff clusters hereafter) containing sequences from at least 3 organisms. These clusters contain 65,913 sequences of which 63,113 sequences belong to COG clusters. The number of clusters that exactly matched COGs is 540, which is close to the 587 exact matches for the MPC_Phyl clusters. The number of COGs exactly partitioned into MPC_Phyl_Cutoff is 894 while this number for the MPC_Phyl clusters is 1,324 - a slightly better result. The average number of COGs that overlap with MPC_Phyl_Cutoff clusters is 1.28 which is an improvement over the 1.38 for MPC_Phyl clusters.

A comparison using the statistical coefficients also shows that the MPC_Phyl_Cutoff clustering is similar to the MPC_Phyl clustering. The values of *Cramer's V* coefficient, *Rand index* and the *Adjusted Rand index* are also similar for the two clusterings.

Essentially, the set-theoretic and statistical coefficients show that the two clusterings MPC_Phyl_Cutoff and the MPC_Phyl are largely in agreement. The MPC_Phyl_Cutoff clustering is better from the perspective of homogeneity within the clusters. We have seen that MPC_Phyl clustering contains one cluster with sequences from 111 different COGs; there are no such cases of high heterogeneity in the MPC_Phyl_Cutoff clusters. In fact, the maximum number of COGs that intersect with any cluster is 12, and only 8 clusters overlap with more than 5 COGs. Thus, MPC_Phyl_Cutoff is considerably more consistent with the COGs maintaining the advantage of using the phylogenetic tree without suffering from merging unrelated ortholog clusters.

6.4 Plant Genome Clustering

We applied the multipartite graph clustering method on the 33,227 protein sequences from the five plants genomes described in section 5.2. For this clustering, we used the known phylogenetic tree between these plants, also described in section 5.2. The ortholog clustering results and the results for annotating the rice protein sequences using the discovered orthologs clusters are described below.

Organisms in Clusters	Cluster size						
	2	3	4	5	6-10	11-25	>25
S+M		2		1			1
B+W	8	10	1	3		1	
S+M+B+W			1		1		
A+S+M+B+W				1	5	17	26
A + cereal(s)	251	358	180	132	234	131	39
others	20	10	1	2	5	4	
Total	279	375	183	139	244	155	66

Table 6.3: Joint distribution of size and species in clusters. S: sorghum; M: maize; B: barley; W: wheat; A: *Arabidopsis*; cereal: one of S,M,B,W.

6.4.1 Clustering Results

The clustering process took less than 20 minutes (discounting the BLAST comparison time) on a Pentium 2.8GHz machine, and produced 22,442 sequences as singletons (94 sorghum + 1,038 maize + 185 barley + 286 wheat + 20,839 *Arabidopsis*). The remaining 10,785 sequences were grouped into 1,440 clusters, which we call the candidate ortholog clusters.

The distribution by species in these candidate ortholog clusters is shown in Table 2. As is clear from this table, most clusters are small and contain only a few sequences from each species. The presence of 66 large clusters (size $\geq$ 25) is intriguing, as one would expect ortholog clusters to contain only a few sequences from each genome. We investigated these and found them to be related to various *transposable elements* and *retrotransposons* which are wide-spread in plants.

An evaluation of the quality of ortholog clusters requires comparing the clusters with benchmark (manually curated) ortholog clusters, which are not available for this data. So, an assessment of the homogeneity of clusters was performed using Pfam as an independent and relevant source of annotation. It should be emphasized that the same Pfam annotation for proteins from different species does not imply orthology. However, when proteins are orthologous, they are likely to be involved in similar functions, so they must have similar Pfam annotation. Moreover, when multiple Pfam families are associated with orthologous proteins, they must appear in the same (domain) order in all members of the ortholog family.

As shown in Fig. 6.8, almost 75% (1,063) of clusters contain sequences with the same set of Pfam annotations. Moreover, when multiple Pfam annotations were associated with sequences in a cluster, the order of their appearance (domain architecture) in different sequences was preserved, indicating their orthology. There are 1,239 (86%) clusters in which all sequences share at least one Pfam family annotation, while sequences in 185 (13%) clusters could not be annotated. There are 26 (1%) clusters with sequences from different Pfam families, but the keywords associated with those annotations are very similar. These statistics indicate that our ortholog clusters are homogeneous.

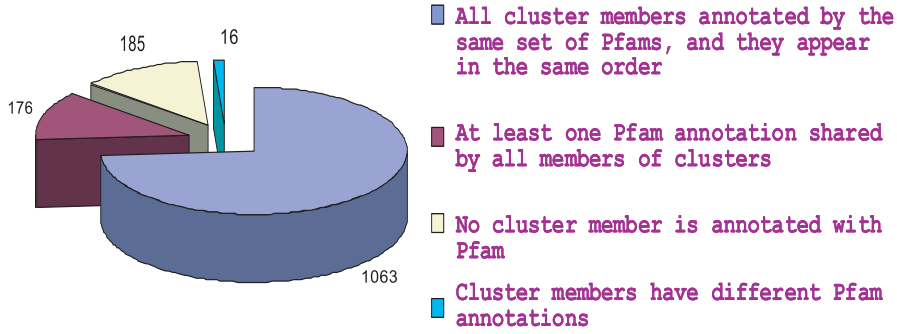

Figure 6.8: Consistency of members of ortholog clusters as shown by Pfam annotations.

There are 49 clusters with sequences from all the 5 species. Most (33 clusters) of these clusters are large (size $\geq$ 10) and contain multiple sequences from each species. An inspection of these clusters revealed that their members are consistently annotated and are related to vital processes, such as *transcription* and *RNA synthesis*, and to pathways that are specific to plants such as *disease resistance, chlorophyll, phytochromes, gibberlin, starch branching enzymes* etc . These annotations confirm an obvious fact: genomic regions known to contain sequences of interest are likely to have been the first sequenced, and so these well-studied proteins are critical for function annotation.

Our clustering results also show some lineage-specific expansion of novel protein families. We analyzed 23 clusters containing sequences exclusively from barley and wheat, and 4 clusters containing maize and sorghum sequences only. The barley-wheat

clusters contain genes specific to relatives of wheat, for instance, one cluster (with annotation *Hordoindoline*) containing 19 sequences is related to seed hardness in relatives of wheat [19] while another cluster is annotated as *Sucrose-phosphate synthase*, which has recently diverged significantly from orthologs in other plants [15]. Other barley-wheat and sorghum-maize clusters represent similar cases of family-specific proteins. For instance, a sorghum-maize cluster annotated as *kafirin* and *zein* has 22 zein-proteins from maize and 20 kafirin proteins from sorghum; the zein and kafirin proteins are known to be orthologous proteins found specifically in the sorghum-maize lineage [74].

6.4.2 Rice Sequence Annotation

We were able to annotate 25,540 protein sequences with 1,310 of the 1,440 ortholog clusters, and 130 clusters were not used for annotating any sequence. (Computationally, the annotation stage is inexpensive and was completed within 5 minutes on a Pentium 2.8 GHz machine.) Most of the clusters annotated a few sequences (see Fig. 6.9), for instance 241 clusters annotated 1 sequence each. However, 10,894 were annotated by 35 clusters alone. A manual inspection of annotations of members of these clusters revealed that these are related to large families of genetic elements, such as transposable elements (10 clusters), retrotransposons (17 clusters including a polyprotein related cluster that annotates 1728 sequences) and other repeat elements (such as PPR repeats) widely present in plants.

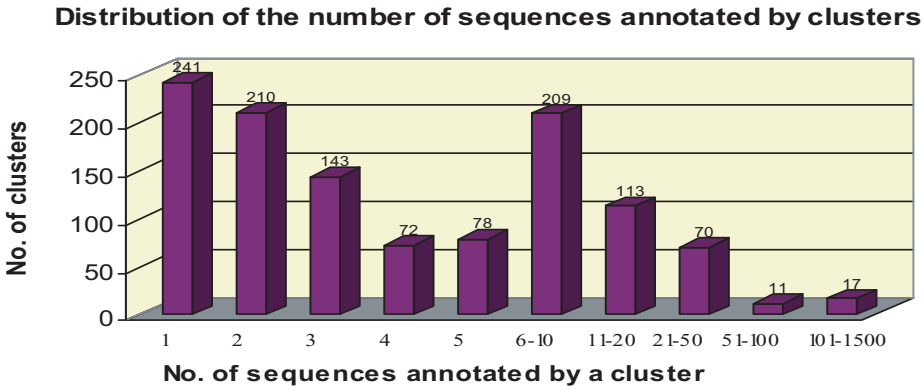

Figure 6.9: Distribution of sequences annotated by a cluster.

An evaluation of automatic annotation can be performed by comparison with the existing manually curated annotations available for rice sequences. Unfortunately, these annotations do not use standardized terminology and cannot be used for automated validation. So, for a systematic evaluation, we used the Pfam annotation for clusters. An ortholog cluster was annotated with the set of Pfam families shared by at least 80% of its member sequences. We found that varying this threshold between 50% to 95% has a negligible effect on the annotations for the clusters, as most clusters contain members with the same set of Pfam annotations. The 1,310 candidate ortholog clusters used for annotation can be divided into 1,090 clusters that are annotated with at least one Pfam family and 220 clusters that could not be annotated with any Pfam family.

The annotation results are summarized in Table 6.4. About 68% (17,356 of 25,540) of the annotated rice sequences exactly match the Pfam annotation of their corresponding clusters (including the organization of different Pfam families when multiple Pfams are present) and about 82% (20,844 of 25,549) share at least one Pfam annotation with their clusters. Among all the annotated sequences, 14% (3,656) rice sequences and the 243 clusters used to annotate them are not associated with any Pfam family. The existing annotations (available as part of the input data) for these clusters and rice sequences show that they are annotated as *hypothetical* or *putative proteins*. It must be emphasized that these are the cases where ortholog based methods provides more annotations (although not specific) while model based methods fail to provide any annotation at all.

Evaluation of performance for our annotation criterion: We compared our annotation results using the criteria (4.4) with those obtained using the best-Blast-hit (bBh) criteria (shown in Table 3). The bBh criterion annotated 33,144 rice sequences compared to 25,540 sequences annotated by our criterion. Although more sequences could be annotated using the bBh criterion, it used almost the same (1,310 vs 1,316) number of clusters for annotation. The usage of same clusters for annotation by both criteria further supports our earlier observation that the unused clusters are related to lineage-specific protein families not present in rice. As mentioned earlier, 68% of sequences annotated using our criterion exactly matched the Pfam annotation for their

Statistics	Annotation using our criterion	Annotation using bBh
Sequences Annotated	25,540	33,144
Exact match between Pfam annotation for rice sequence and corresponding ortholog cluster	17,356	20,224
Partial match between Pfam annotation for rice sequence and corresponding ortholog cluster	3,488	3,301
Clusters used for Annotation	1,310	1,316
Pfam annotated Clusters used for Annotation	1,014	1,030

Table 6.4: Results of annotation and comparison with best hit based annotation approach.

clusters compared to 61% for the bBh criterion. When we require that sequences share at least one Pfam family annotation with the corresponding cluster, results are 82% to 71% in favor of our criterion. Thus, our criterion for annotating sequences with ortholog clusters gives lesser but more sensitive coverage.

6.5 Conclusions

We have tested and evaluated the multipartite graph clustering method to the ortholog clustering problem for two different data sets. The first data comprises the 43 complete genomes, we found that the candidate ortholog clusters produced by the multipartite graph clustering method are consistent with the known expert curated ortholog clusters. This was shown by qualitative analysis of example ortholog clusters and also supported by various set-theoretic comparison measures and statistical coefficients for comparing our clusters with COGs. The validation of clusters using independent sources - Pfam and SCOP, demonstrated their homogeneity from the perspective of protein function and structure, respectively.

We also demonstrated that using a phylogenetic tree for rescaling observed similarity values between sequence pairs improved the results further. Essentially, rescaling was successful in aggregating the ortholog clusters that formed part of the same COG clusters. This was systematically shown by various statistical coefficients and

comparison-measures such as an increase in the number of exactly matched clusters, increased similarity in the distribution of cluster sizes and the number of organisms contained in clusters.

Discovering ortholog clusters in partially completed genomes is important but currently there are no methods for solving this problem. The multipartite graph clustering method was also successful in finding homogeneous ortholog clusters in the five plant genomes, 4 of which were very sparsely represented in the input data. Since there no known ortholog clustering results on this data, it was not possible to validate them directly. Their validation based on membership in Pfam and SCOP classes showed them to be quite homogeneous from the protein function and protein structure perspectives. Using the ortholog clusters, we were able to annotate about 40% of the rice genome. In this annotation most of the clusters annotated only a few sequences, as one would expect from orthologous relationships. However, there were a few clusters which were used for annotating a large number of rice sequences. Although this is different than orthology, these candidate ortholog clusters were found to be annotated as transposable elements of different classes which justifies a large number of sequences being annotated by these clusters.

Chapter 7

Conclusions and Future Work

We considered the problem of clustering a structured data set, where together with pair-wise similarities between objects, we are also given additional information about the structure of organization of objects. We focused on a simple structure - a partition model of data where the objects are a priori partitioned into groups. The goal was to find clusters of objects considering the pair-wise similarities across the partite sets. While clustering such data, we also considered an additional requirement of being able to differentiate between pair-wise similarities across different partite sets. Existing clustering methods do not solve this problem, since they are limited to finding clusters in input data which is treated as a collection of isolated objects. The abstract problem of clustering structured data is under-explored, and worthy of attention in its own right. We were also motivated by a concrete problem in genomics and wanted to explore the potential of developing a methodology that is applicable to various data analysis fields. So, we set out to investigate and find both the formulation and solution to the problem. Along side the abstract we had a well-fitting practical problem, the ortholog clustering problem. This constantly motivated us to keep in sight various practical issues such as efficiency of the algorithms as well as modeling specifics of the problem. This thesis aimed at contributing to all of the above aspects of the clustering problem.

7.1 Problem Formulation: Quasi-cliques in a multipartite graph

We considered two types of relationships in the data - the pair-wise similarity relationships and the relationships between the partite sets (subset defining the partition model). The problem required us to focus exclusively on the pair-wise similarities across the subsets specified in the input data and this naturally led to a multipartite graph as

a framework for representing both of these relationships. Furthermore, the requirement of differentially treating pair-wise relationships across different groups was modeled by constructing another graph, whose vertices are the partite sets and the weights on edges connecting the partite sets determine how we must differentiate between various pair-wise similarities.

The multipartite graph of pair-wise similarities along with the graph between the partite sets suitably represents the clustering problem for structured data characterized by a partition model. Also, we think it is the simplest, yet general, framework necessary for representing multiple levels of relationships in data. Although we limited our focus to a hierarchical relationship between the partite sets, the representation framework (and also all other aspects of the problem formulation) are applicable to more complex relationships. Having represented the problem, we addressed the issue of simultaneously considering two graphs while formulating the problem.

We opted for a sequential extraction of clusters (beginning with the strongest cluster) from the multipartite graph, enabling us to determine the number of clusters in the data automatically. So, the problem reduces to finding the strongest cluster or dense subgraph in the multipartite graph. This problem is usually formulated as the clique problem or as its relaxation quasi-clique problem. Since the latter is a more realistic model of a cluster in data, we opted for it. Traditionally, quasi-cliques are defined for unweighted graphs and their definitions, being based on cliques, require a predefined threshold. For our problem, we have weighted graphs, yet, in practice, one does not have an insight into the appropriate value of the threshold. We formulated a definition of quasi-cliques that does not involve any threshold, and as a consequence of the sequential extraction of clusters, the clustering method does not require specifying the number of clusters as an input.

The main idea in formulating quasi-cliques was to define a score for an arbitrary subset of elements. Such a score must reflect the density of the subset. In order to do this, three methodological problems were solved: (i) measuring similarity between objects considering the relationships between the partite sets, (ii) measuring similarity between an element and a subset, and finally (iii) measuring the score for the subset

using the element-to-subset similarity measure. In chapter 2, we described a method to combine the two levels of relationships. The score for an arbitrary subset was defined as the similarity of weakest (least-similar) object to all other objects in that subset. Using this, a quasi-clique was defined as the subset with the highest score, or the subset in which the least-similar object has highest similarity to all other objects in that subset. This allowed us to formulate the clustering problem as a combinatorial optimization problem. While designing all these, we considered the practical relevance and interpretation of the score and also efficiency of finding such a score.

7.2 Algorithm for Multipartite graph clustering

It was shown in chapter 2, that due to a specific property of the function we designed to score subsets, the globally optimal solution can be found efficiently. A straightforward algorithm for finding a single cluster in a multipartite graph takes $O(|E| + |V|^2)$ time, where E is the set of edges in the pair-wise similarity graphs and V is the set of objects. We analyzed the algorithm and found a particular step where the algorithm spends significant time. We found that using Fibonacci Heap data structure, it was possible to improve the run time of the algorithm to $O(|E| + |V| \log |V|)$. Furthermore, when the weights on the edges of the pair-wise similarity graph can be discretized into a constant number of intervals, we were able to further improve the runtime to $O(|E| + |V|)$ using the bucket sort algorithm.

The overall complexity of finding all clusters is $O(m(|E| + |V| \log |V|))$, where m is the total number of clusters in the data. The algorithm takes $O(|E|+|V|)$ storage space. There were practical considerations for efficiency as we wanted to cluster hundreds of thousands of objects. We also found heuristics which lead to significant performance gains in practice.

While in chapter 2 we focused on formulations that had direct practical applications, these formulations were generalized in chapter 3 and shown to be applicable to a wider class of problems. We found that the abstract properties of functions guaranteeing an efficient algorithm are very general. We further exposed those properties

and discussed examples of functions satisfying these properties. We also described a complementary class of score functions and their properties including algorithms for finding their optimal solutions.

7.2.1 Availability of the Software

The software is written in C++ and is available upon request.

7.3 Ortholog Clustering

We modeled the ortholog clustering problem on a multipartite graph. This allowed us to ignore pair-wise similarities between genes within genomes and focus on the inter-genome gene similarities. Traditionally, orthologs are extracted from a pair of genomes and then these ortholog pairs are extended to multiple genomes. The method we have used allows us to extract ortholog clusters simultaneously from multiple genomes. This was done by identifying the ortholog clusters with the multipartite clusters.

We adapted the multipartite graph clustering method for ortholog clustering by using various specific characteristics of the ortholog problem. Two important ones were rescaling the observed pair-wise sequence similarities and using gene-specific and genome-specific cut-offs for filtering out pair-wise similarities between non-orthologous genes. Ortholog genes, although conserved, diverge in sequence over time and the observed divergence between orthologs in a pair of organisms is correlated with the time elapsed since their last common ancestor. We used the information about the phylogenetic tree to correct for the observed differences between orthologs from different pairs of organisms. This correction was in the form of rescaling the observed similarities by an estimate of evolutionary distance between organisms. Since our focus was on functional orthologs, a requirement in ortholog clustering was to distinguish between the anciently duplicated paralogs and the recent ones. Since anciently duplicated paralogs are likely to be less conserved in sequence, such gene similarities can be ignored by removing the "less similar" genes. Different ortholog genes have different levels of similarity, so we used a gene-specific genome-specific threshold to remove anciently

duplicated paralogs without removing the potential orthologs.

7.3.1 Ortholog Results

We applied the multipartite graph clustering method to the ortholog clustering problem on two different data sets. In the first data comprising the 43 complete genomes, we found that ortholog clusters produced by the method are consistent with the known expert curated ortholog clusters. This was shown by qualitative analysis of example ortholog clusters and also supported by various set-theoretic comparison measures and statistical coefficients for comparing our clusters with COGs. The validation of clusters using the independent source - Pfam and SCOP, demonstrated their homogeneity from both functional and structural perspective.

We also demonstrated that the phylogenetic tree for rescaling the observed similarities improved the results further. Essentially, the rescaling was successful in aggregating the ortholog clusters that formed part of the same COG clusters. This was corroborated by various comparison measures such as an increase in the number of exactly matched clusters, increased similarity in the distribution of cluster sizes and the number of organisms contained in clusters, as well as by statistical coefficients.

Discovering ortholog clusters in partially completed genomes is important but currently there are no methods for solving this problem. The multipartite graph clustering method was also successful in finding homogeneous ortholog clusters in the five plant genomes, 4 of which were very sparsely represented in the input data. Since there are no known ortholog clustering results on this data, it was not possible to validate them directly. Nonetheless, their indirect validation based on membership in Pfam and SCOP classes showed them to be homogeneous from the perspective of protein function and structure. Using the ortholog clusters, we were able to annotate about 40% of the rice genome. In this annotation most of the clusters annotated only a few sequences as one would expect from orthologous relationships. However, there were a few clusters which were used for annotating a large number of rice sequences. Although this goes against the notion of orthology, these ortholog clusters were found to be annotated as transposable elements of different classes which are wide-spread plants, and this justifies

a large number of sequences being annotated by those clusters.

7.4 Problems in Computer Vision

As a proof of the broader applicability of multipartite graph clustering models to other real world problems, we adapted the developed multipartite graph clustering method to two problems in computer vision. In the first problem, we applied it to the gait recognition problem. The goal of this problem is to learn to identify a subject from a sequence of gait frames. We addressed the subproblem of finding gait frames that characterize a subject. These gait frames were later used for training a supervised classifier. We solved this problem as finding similar gait frames from multiple subjects, or uninformative gait frames. The uninformative gait frames were extracted as clusters from a multipartite graph where the vertices correspond to the gait frames and the edges correspond to similarities between the gait frames and the partite sets correspond to the subjects. The uninformative gait frames were excluded from the process of training a supervised classifier. We found that the classifier's performance improved when it was trained only on the informative gait frames (gait frames complementary to the uninformative frames). The recognition rates were competitive with state-of-the-art methods, and in most cases we were able to outperform existing methods.

We also applied the multipartite graph clustering to discriminative patch selection in patch based object recognition. We used a collection of images that contain the target object along with another collection of images that do not contain a target object. The saliencies of the target object can be captured in the patches coming from those images and these patches form a good representation for the target object. We excluded patches that are likely to be present in the images not containing the target object. So, the goal was to find clusters of patches that were present in most of the images containing the target object and at the same time very different from those in images not containing the target object. Furthermore, while finding the discriminative image patches that characterize the target object, we focused on similarities between patches across different instances of the target object, rather than similarities between patches from the image (although they may be very similar). These two informal

requirements can be conveniently expressed in the multipartite graph representation of the similarities between patches from different images [88].

The weighted multipartite graph was constructed using the two classes of images: the images containing the target object and those that do not contain it. The patches corresponding to images not containing the target object constituted a single negative class and we were interested in extracting patches that are dissimilar to patches in this negative class. Each image containing the target object constituted a class which contained vertices corresponding to patches from this image. In the multipartite graph, the similarities between patches contained within a class were ignored. For the discriminative subsets of patches, we extracted subsets of patches that were similar to patches from images containing the target object and distant from patches in the negative class. We applied this method for patch based object recognition to a database of objects and obtained results that are competitive with those from the current methods and in most cases show improved recognition over the existing methods [88].

7.5 Future Work

The thesis focused on two aspects of clustering in structural data: (a) problem formulation and the development of the multipartite graph clustering method, and (b) its applications. There are promising future directions in both of these aspects. We begin by outlining the possibilities for extending the conceptual framework for multipartite graph clustering and then describe how the proposed extensions would improve solutions to the problems we have studied.

7.5.1 Strengthening the problem representation and formulation

The multipartite graph framework can be extended to represent more complex types of data. The method already developed is suitable for structural input data characterized by a partition model, however, there are cases where structure in data is more complex and cannot be represented by a partition model. An interesting extension is to consider

relationships between the objects within the partite sets. Such relationships may correspond to roles played by the objects (in the larger system represented by the partite set). The role-level similarities capture the contextual or semantic similarity and are equally important in expressing the characteristics of the desired groups. For instance, in the ortholog clustering problem, it would be desirable to consider relationships between genes within an organisms; such relationships can represent the interactions between genes at various levels. In the gait recognition problem the modified graph can capture the frame sequences information, and in the patch selection problem it can represent the spatial organization of patches essential for modeling the target object. In essence, considering the additional structure in data allows us to incorporate increasing amounts of domain knowledge.

There are two aspects in extending the current framework - representation of the problem and its formulation. Representing multiple types of similarities will require extending the multipartite graph by allowing additional structures. The current framework focuses exclusively on the similarities between objects across different partite sets (classes). The role-level similarities between objects within a partite set can be represented by edges inside the partite sets. We can have multiple graphs on the same set of vertices with each graph expressing relationships of a particular kind. A combined use of different types of similarities will allow us to express the problem better and so guide us towards increasingly accurate solutions. At the first glance, considering relationships within a partite set may seem to contradict the current multipartite graph representation. It must be emphasized that the space of functional roles is different from that of physical sequence similarity and therefore, only adds another useful (semantic) dimension for reliably determining similarities across the partite sets.

Considering an additional structure in the data leads to a more complex problem formulation but allows us to incorporate critical domain knowledge. We consider the abstract clustering problem of finding groups of similar objects taking into account their pair-wise relationships, their roles within the partite sets and the relationships between the partite sets. Incorporating domain knowledge (by considering different types of similarities) involves analyzing heterogeneous data covering different aspects of the

same phenomena, and it can vastly improve the results from an automated analysis. In many instances, the different types of relationships are not comparable and are best used for finding the final solution by synergistically combining the individual solutions from individual types of relationships. Nonetheless, due to the lack of any suitable framework for enabling integration of heterogeneous data, such incomparable relationships are packed together into a single vector which is then used for formulating the optimization problem. Alternately, there are constrained optimization formulations where a single observation is used for determining the criterion while others play a subordinate role of constraints. In either case, the different observation types do not synergize or guide one another towards the final solution. One can develop an efficient and intuitive combinatorial optimization framework for such problems. This can be achieved by using separate objective functions for different types of relationships, which will then guide each other towards the "most consistent" final solution. Abstractly, this can be envisaged as multiple graphs on the same set of vertices with each graph expressing relationships between nodes in a different type of observation, and the objective is to find a subset of vertices which best reflects the relationships for all the observation types. A pleasing consequence of this is the freedom in designing objective functions appropriate for each observation type. The final solution can be a game theoretic equilibrium point between the multiple objective function such that any change in the solution adversely affects the individual objective values. This will guarantee a solution with a large optimal value without sacrificing values any of the objective function values. This will keep the heterogeneous and incomparable observation types separate until the very end while seeking the optimal solution based on cues from individual objective functions.

7.6 Improving Ortholog clustering

The proposed extension to the multipartite graph clustering framework will help improve the ortholog clustering method. The extensions will allow us to incorporate knowledge about genes beyond their pair-wise sequence similarity.

Some of the most difficult cases in extracting of groups of orthologous genes are related to multi-domain protein families. Our problem formulation currently ignores

this aspect of the problem and extracts clusters at the complete gene level. Due to increasing and reliable coverage of the protein domain space, it is increasingly possible to segment the protein sequences into domains. Using the domain sequences instead of the complete gene sequences has multiple advantages. It will increase the sensitivity of the clusters, since domains are the most basic functional units, and perhaps also the basic units of sequence conservation. Additionally, the multipartite clusters on domains will enable us to detect various other types of evolutionary events such as the domain shuffling and domain fusion events. In the current problem formulation, protein sequences with shuffled or fused domains are likely to pass as ortholog clusters, or at least prove to be hard cases for ortholog clustering. By extracting the domain level clusters, we would be able to better characterize the evolutionary relationships between protein sequences and also be able to provide more sensitive annotations for new protein sequences (proteins with shuffled or fused domains usually perform different biological functions).

It is widely recognized that objectively computed sequence similarity is not sufficient to accurately infer orthologous relationships [23]. Compelling evidence of this is that most evolutionary biologists prefer to manually extract true orthologs for their studies, despite the fact that using a large collection of orthologs from automated methods can greatly improve the statistical results of their findings. This is primarily because an expert can simultaneously consider multiple relationships between genes, such their positional location in the genome, exon-intron structure, domain architecture, and localization in the cell among others. It would be interesting to see how an automated method could simultaneously work with these different types of similarities. A possible formulation could be by setting up different optimization problems for different spaces of similarities and extracting an ortholog cluster based on an optimization problem using the solutions of individual smaller optimization problems. The critical issues in such an approach would the formulation of the optimization problem for extracting the final ortholog clusters and the computational efficiency of such a process. By including these diverse features, we hope to get closer to expert-based ortholog detection.

Appendix A

Fibonacci Heaps

A frequent set of operations in the algorithm to find the optimal solution in chapter 3 and 4, is finding the element(s) with the minimum value of the linkage function (`find-min`)and deleting it from the current list of active nodes (`delete-min`) and updating the linkage function value of the neighbors (`decrease-key`) which are affected by the deletion of the minimum value element. Due to a high frequency of this set of operations, the run time is critically tied to the overall time spent in carrying out these three operations. In this appendix we describe the *Fibonacci Heap* [30] data structure which has the amortized running time $O(1)$, $O(\log n)$, and $O(1)$ for the operations `find-min`, `delete-min`, and `decrease-key`, respectively.

A.1 Fibonacci Heaps: Overview

A Fibonacci heap is a collection of trees satisfying the *minimum-heap property*, that is, the key of a child is always greater than or equal to the key of the parent. This implies that the minimum key is always at the root of one of the trees. A Fibonacci heap is comprised on binomial heaps [17] (see Fig. A.1) but compared with binomial heaps, the structure of a Fibonacci heap is more flexible. The trees do not have a prescribed shape and in the extreme case the heap can have every element in a separate tree or a single tree of depth n. This flexibility is characteristic of the Fibonacci heaps and allows some operations to be executed in a "lazy" manner, postponing the work for later operations. For example, the union of two heaps is done simply by concatenating the two lists of trees, and the operation `decrease-key` sometimes cuts a node from its parent and forms a new tree.

By doing the operations in a lazy manner it saves on computations, and performs

effectively because the Fibonacci heap property is restored only when when enough work accumulates. This fine balance between postponing work and carefully choosing the time to restore the heap property enables an efficient amortized running time, although the worst case running time for individual operations can be comparable to other data structures. At some point some order needs to be introduced to the heap to achieve the desired running time. In particular, degrees of nodes (here degree means the number of children) are kept quite low: every node has degree at most $O(\log n)$ and the size of a subtree rooted in a node of degree k is at least F_{k+2}, where F_k is kth Fibonacci number. This is achieved by the rule that we can cut at most one child of each non-root node. When second children is cut, the node itself needs to be cut from its parent and becomes the root of a new tree. The number of trees is decreased in the operation `delete-min`, where trees are linked together.

As a result of a relaxed structure, some operations can take a long time while others are done very quickly. This is the key to obtain better amortized running time. In the amortized running time analysis we pretend that very fast operations take a little bit longer than they actually do. This additional time is then later subtracted from actual running time of slow operations. The amount of the time saved for later use at any given moment is measured by a potential function. The potential $\Phi(H)$ of a Fibonacci heap H is given by

$$\Phi(H) = t(H) + 2m(H) \tag{A.1}$$

where $t(H)$ is the number of trees in the Fibonacci heap, and $m(H)$ is the number of marked nodes in H. A node is marked if at least one of its children was cut since this node was made a child of another node (all roots are unmarked).

Thus, the root of each tree in a heap has one unit of time stored. This unit of time can be used later for linking this tree with another tree at amortized time 0. Also, each marked node has two units of time stored. One can be used to cut the node from its parent. If this happens, the node becomes a root and the second unit of time will remain stored in it as in any other root.

A.2 Implementation details

A Fibonacci heap is implemented as a circular doubly linked list, with a pointer to the minimum key, but the elements of the list are not single keys. Instead, the keys are collected together into structures called *binomial heaps*. Binomial heaps are trees that satisfy the heap property and have the following special structure as shown in Fig. A.1.

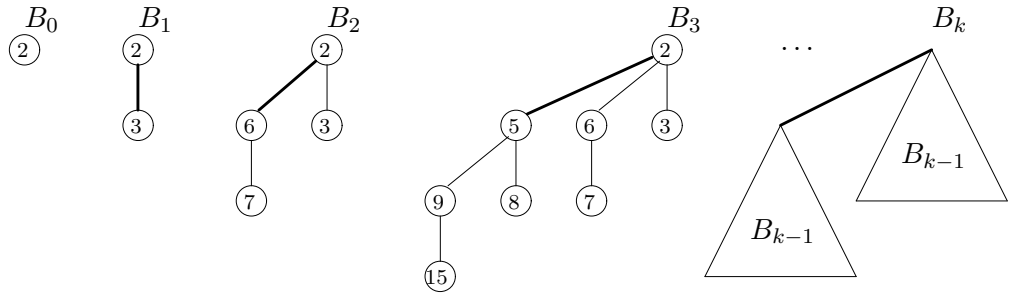

Figure A.1: Binomial trees of different orders. A binomial tree B_k of order $k(> 0)$ contains two copies of B_{k-1} as shown connected by the thick lines. The values contained in the nodes (circles) satisfy the heap property and are shown as numbers inside the circles.

Every node in a Fibonacci heap points to four other nodes: its parent, its 'next' sibling, its 'previous' sibling, and one of its children. The sibling pointers are used to join the roots together into a circular doubly-linked root list. In each binomial tree, the children nodes are also joined into a circular doubly-linked list using the sibling pointers as shown in Fig. A.2.

A.3 Operations

Although Fibonacci heaps have the best known amortized running times for different many operations, we will restrict this presentation to the three operations of interest here, viz. `find-min`, `delete-min`, and `decrease-key`. This section also presents a brief description of the amortized analysis for these operations.

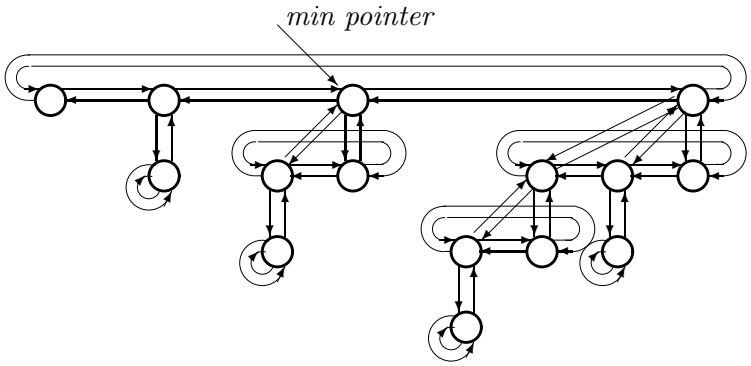

Figure A.2: A detailed view of a Fibonacci Heap showing the circular doubly linked list of binomial heaps. The minimum pointer and the four pointers associated with each node are shown. For clarity, the null pointers and the values contained inside the nodes are not shown.

A.3.1 `Find-min`

The heap property, coupled with the fact that we keep a pointer to the node containing the minimum key element makes finding the element with minimum key trivial. Furthermore, this operation does not change the potential of the heap, so both the actual and the amortized cost remain constant. Another operation similar to finding the minimum is taking the union of two Fibonacci heaps and is implemented simply by concatenating the lists of tree roots of the two heaps. This can be done in constant time and the potential does not change, leading again to constant amortized time. Likewise, the insert operation works by creating a new heap with one element and taking the union. This again takes constant time, however, the potential increases by one, because the number of trees increases. The amortized cost is thus still a constant.

A.3.2 `Delete-min`

The `delete-min` operation, also called the "extract minimum" operates in three phases. First we find the root containing the minimum element and remove it. Its children will become roots of new trees. If the number of children was d, it takes time $O(d)$ to process all new roots and the potential increases by $d - 1$. Therefore, the amortized running time of this phase is $O(d) = O(\log n)$. However to complete the "extract

minimum" operation, we need to update the pointer to the root with the minimum key. Unfortunately, there may by up to n roots we need to check. In the second phase we therefore decrease the number of roots by successively linking together roots of the same degree. When two roots u and v have the same degree, we make one of them a child of the other so that the one with smaller key remains the root. Its degree will increase by one. This is repeated until every root has a different degree. To find trees of the same degree efficiently we use an array of length $O(\log n)$, in which we keep a pointer to one root of each degree. When a second root is found to have the same degree, the two are linked and the array is updated. The actual running time is $O(\log n + m)$ where m is the number of roots at the beginning of the second phase. At the end we will have at most $O(\log n)$ roots because each has a different degree. Therefore the potential decreases by at least $m - O(\log n)$ and the amortized running time is $O(\log n)$.

In the third phase we check each of the remaining roots and find the minimum. This takes $O(\log n)$ time and the potential does not change. The overall amortized running time of extract minimum is therefore $O(\log n)$.

A.3.3 `Decrease-key`

The operation `decrease-key` takes the node, decreases the key, and if the heap property is violated (i.e., the new key is smaller than the key of the parent), the node is cut from its parent. If the parent is not a root, it is marked. If it was marked already, it is cut as well and its parent is marked. We continue upwards until we either reach the root or an unmarked vertex. In the process we create some number, say k, of new trees. Each of these new trees (except possibly the first one) was marked originally, but as a root it will become unmarked. One node can become marked. Therefore the potential decreases by at least $k - 2$. The actual time to perform the cutting was $O(k)$, therefore the amortized running time is constant.

Finally, the operation `delete` can be implemented simply by decreasing the key of the element to be deleted to minus infinity, thus turning it into the minimum of the whole heap. Then we call "extract minimum" to remove it. The amortized running time of this operation is $O(\log n)$.

Appendix B

Ortholog detection: A survey

B.1 Introduction

Orthologs are homologous genes that have evolved through vertical descent from a single ancestral gene in the last common ancestor of the considered species [29]. The ortholog detection problem is complicated due to the presence of another type of homologous genes in a genome, called paralogs [29]. Paralogs are results of duplication events where a segment of DNA gets duplicated leading to two copies of the gene in the duplicated segment. Paralogs are known to functionally diverge and adapt to new functions, although they maintain a high level of sequence similarity to the original gene. So, it is the comparable level of similarity between paralogs that complicates the detection of orthologs.

In this appendix, we survey methods for finding orthologous genes. We begin by briefly reviewing various uses of orthologous genes and looking at the evidence that suggests orthology. This is followed by a survey of methods and their analysis.

B.2 Purpose of ortholog detection

Being the trace of vertical evolution, orthologs are the basic source for studying molecular evolution. Distinguishing orthologs and paralogs is critical to describe evolution accurately. It is equally important for inferring gene function [Koonin '01]. Among others, the purposes of ortholog detection include:

- *Functional annotation:* Identification of orthologous sequences is necessary for transferring functional annotations and other knowledge from well-studied genes in model organisms to newly sequenced genes.

- *Target selection for experimental studies:* In order to increase our knowledge of new protein, we need to study novel families. Genes that do not have orthologs in other species are likely to be novel, and are valuable in prioritizing experimental studies [18] to enrich our knowledge of diversity in genes.

- *Anchors for determining other conserved genetic elements:* Along with the orthologous genes, other genetic elements are also inherited from ancestral genomes. However, these elements are weakly conserved and so do not carry enough signal by themselves for their recognition. Many of these (regulatory motifs, transcription start points etc.) are located in the vicinity of orthologous genes. So, orthologs are used as anchors to boost the signal for identification of the weakly conserved genetic elements [73].

- *Molecular evolution:* Correct identification of orthologous sequences is necessary for inferring the evolutionary history (phylogenetic trees) of the organisms.

- *Rates of protein-evolution:* When a phylogenetic tree for a set of organisms is known with confidence, orthologs are a reliable source for studying the rates of gene evolution and for estimating the divergence times for the species.

- *Medicinal purpose:* One of the purposes of studying genomics is to enhance our knowledge about disease-causing genes. Study of orthologs of genes implicated in causing diseases in microbes can help us to prevent epidemics by developing proper safe guards such as vaccines and other medicines. Also, we can find out more about the genes that can contribute to undesirable health conditions in humans, by studying them in closely related organisms.

B.3 Evidence for Orthology

The vertical inheritance of orthologs entails conservation of various features that can serve to improve the signal for ascertaining orthology. Although many of such signals also exist for paralogs and other kinds of homologous sequences, these signals are more prominent for orthologs.

- **Sequence Similarity :** The constraint of function retention for orthologs is usually reflected in the conservation of their primary sequence. The functional conservation of orthologs means that they can often be identified as the most similar sequences in a pair of genomes [83, 68] distinguishing them from paralogs and other such sequences. Primary sequence similarity, either in the nucleotide sequence or in the amino-acid sequence, is used in all ortholog detection methods.

  A shortcoming of using primary sequence similarity for ortholog detection is the weakening of this signal with increasing divergence of the species. In the absence of this signal, the secondary structure (involvement of residues in a particular shape) and the tertiary structure (the 3-dimensional structure) related to the sequence are helpful in identifying orthologous genes, as they may be more conserved than the primary sequence [49].

- **Gene Order :** Gene order information often provides additional evidence for orthology. Immediately following a speciation event, the gene order for orthologs is conserved and is likely to remain so until disturbed by random genome rearrangements [94]. Over time the rearrangement events are likely to disrupt the gene order for most genes unless the order provides functional benefits. So, conserved gene order serves as important evidence for orthology.

- **Domain organization :** When a protein contains multiple domains, these domains are involved in important roles necessary to carry out a function. Functional conservation requires conservation of each of the constituent domains [41], so orthologs with multiple domains are likely to be conserved in each domain and their organization.

- **Functional Similarity:** Functional conservation across genomes is the strongest evidence for orthology [23]. However, it is also the most difficult to uncover. This feature is valuable for ortholog detection by experts [59].

B.4 Survey of methods

Paralogs are closely related to orthologs, because if a duplication event follows a speciation event, orthology becomes a relationship between a set of paralogs [81]. Due to this complex relationship, most ortholog detection methods focus on identifying *ortholog clusters* rather than sets of "true orthologous genes". Ortholog clusters can contain multiple genes from single organisms. Paralogs related to ancient duplications are placed in different ortholog clusters whereas recently duplicated genes are placed in the same ortholog cluster, as has been done in COG [83], KOG [81], KEGG [31] and Inparanoid [68]. Such a definition of ortholog clusters is justified from a gene function perspective because anciently duplicated genes are most likely to have adapted to new functional niches.

All currently known techniques rely on sequence comparison using sequence alignment or database search techniques. Following the sequence comparison stage, various methods can be broadly divided into **expert-based approaches**, **clustering based approaches**, **phylogenetic tree based approaches**, and others. In the expert-based approaches, sequence comparison is followed by an expert interpretation of results. These approaches require an extensive understanding of genetic sequences and evolutionary mechanisms, and although most valuable, such studies cannot be scaled up to very large data sets. The effort in clustering-based methods is spent in designing (or choosing) an appropriate clustering method. In the phylogenetic tree based approaches, constructing correct a gene tree is the most critical step and inferring orthologs is based on the reconciliation of the species tree and gene tree [34, 62, 63]. We present a brief survey of representative studies of the three approaches.

B.4.1 Expert based methods

In [51], orthologs were defined by using a rigorous phylogenetic approach in HOVERGEN (database of homologous vertebrate genes) [22]. They extracted gene trees for human, mouse, and rat homologous genes from HOVERGEN and used a "triplet test"

for inferring confidence values on their orthology conclusions. They were able to extract a reliable set of 2,820 orthologous rodent and human sequences, which they used for estimating evolutionary parameters of the transcribed human genome. The study suggested an update to evolutionary rates in mammalian genes.

In [16], the complete set of protein sequences from worm and yeast were compared. They used simple sequence comparison and found that 57% of protein pairs contain just one worm and one yeast protein. Interspecies comparison of protein domains used in regulation and signal transduction shows that although there is a considerable sharing of domains, most of the proteins in which they appear are not orthologous. Increasing numbers of proteins during eukaryotic evolution are thought to have originated by domain shuffling [16, 42, 91]. They extensively analyze results for various protein families.

Though it may seem that they identified orthologs based on simple sequence comparison, it is not the case, since the onus of analysis was on the experts, which is apparent from the fact that they could detect many seemingly similar proteins sharing common domains are which turn out not to be orthologs. In essence, biologists rely on computational tools, especially sequence comparison mostly to filter-out pairs that definitely do not form ortholog pairs, instead of using computational tools to identify orthologs.

B.4.2 Clustering based Methods

Due to the availability of large amounts of genomic data, clustering based techniques for ortholog detection have become increasingly necessary. Most ortholog clustering methods are graph based clustering methods. To find ortholog clusters, they cluster the graph, whose vertices are the sequences and edges representing the pair-wise similarity values. Different methods vary in how they define ortholog clusters on the graph.

BBH concept

In many clustering-based techniques (and also in expert-based methods), the phylogenetic analysis phase of complete families is completely replaced by double checking

pairwise similarities, popularly known as Bidirectional best hit (BBH) [48, 61, 83]. The basic idea is that if a sequence i from one organism is most similar to sequence j from a second organism, and if the reverse relationship also holds (i.e., sequence j has i as its closest hit in the second organism), then i and j are probably orthologs. This concept appears by various names such as reflexive best hits [48], reciprocal best hits [89], mutually most similar hits [94]. BBH is central to these ortholog detection techniques and extracts the basic orthologs, which are then extended using heuristics to include other potential orthologs.

The definition of BBH requires finding the best match for a gene in another genome, so it requires availability of the complete set of genes in both the genomes. When the genomes being used for ortholog studies are not complete, this basic assumption is not satisfied, but nonetheless BBH or its variants are used. In such cases, BBH is usually combined with other supporting cues for increasing reliability of putative orthologs [59, 94].

An extension of the BBH concept is the pair of close bidirectional best hits, PCBBH [60]. Two pairs of genes are related by the PCBBH relationship if they satisfy the BBH relationship and their physical location, relative to other orthologs, in their genomes is preserved. The PCBBH relationship operates over multiple ortholog groups in a couple of genomes, and for inferring conservation relative to physical location it uses a predefined maximum acceptable distance (at the nucleotide level defined as the number of base pairs separating the ends of two genes) from other known orthologs.

Clusters of orthologous genes (COG)

One of the seminal works in ortholog detection is COG [26, 81, 82, 83]. COG is a database of clusters of orthologous genes (COGs) which are extracted based on BBH relationships. The ortholog clusters in this database are considered a "gold standard" and are extracted using a semi-automatic method followed by expert curation of the results.

Each COG consists of individual orthologous genes or orthologous groups of paralogs from three or more different lineages and these genes are assumed to have evolved

from an individual ancestral gene through a series of speciation and duplication events. Therefore, COG is an attempt to build ortholog clusters, which includes group of orthologs in different lineages along with paralogs in each lineage related to core orthologs in a COG and as such, it does not provide the true nature of relationship (ortholog/paralog) among individual members of a COG. Methodologically, building COGs involves, as a first step, collapsing obvious paralogs. This is followed by extraction of BBH relationships for genes in the set of given genomes, these relationships are used for identification of sets of at least three related sequences, from different organisms. These groups of three sequences (triangles) are merged if they have two sequences (vertices) in common. The resulting constructions of connected sequences are inspected visually for the detection of problematic multi-domain proteins and groups of paralogous sequences.

Besides providing insights into evolutionary relationships for the considered organisms, COGs are also meant to be a tool for functional annotation, so each COG is described by a functional category and specific function, which may be a broad function or may be uncharacterized. In order to facilitate working with this data, they provide a tool called COGnitor which can assign a query amino-acid sequence to a COG so that its functional or evolutionary relationship can be estimated. A detailed description of the clusters in the COG database is provided in section 5.1.

WIT

The WIT (What Is There?) [61] (`(http://wit.mcs.anl.gov/WIT2/)`) database of ortholog clusters defined orthologs using a variant of BBH, called the PCBBH. It also requires the orthologs to be recognizably similar based on a similarity threshold. The model does not capture the true nature of orthologous relationship. Each cluster contains at most one gene from any single organism, and the genes in the cluster are connected via the bidirectional best hit relation (but not all genes are bidirectional best hits of all other genes in the cluster). Rarely, a gene can appear in multiple clusters. As it uses the concept of positional coupling between genes defined by PCBBH, WIT also attempts to delineate the operon structure of genes in a genome.

TOGA

The TIGR orthologous gene alignments, TOGA, [48] is an attempt to characterize orthologous relationships on a massive scale. Unlike the other methods, TOGA uses EST sequences for orthologous relationships. Since orthologs are known to be conserved at a protein sequence level, they imply sufficient conservation at the DNA level too. This assumption is then translated into an operational definition of ortholog clusters whereby they are identified by requiring reflexive, high-stringency, transitive sequence matches across three or more species.

The TOGA database is complementary to COG as it delineates orthologs using EST (expressed sequenced tags) sequences. As an input it used more than 8 million ESTs from 28 eukaryotes (5 mammals, 10 plants, 7 eukaryotic parasites and 6 others). These were clustered yielding 4.5 million TCs (tentative consensus), singleton ESTs and ETs (expressed tags). Tentative ortholog groups (TOGs) were identified as transitive, reflexive best hits across at least three species with a maximum of blastn e-value of 10^{-5}. This initial clustering created 88K TOGs; in many cases sequences appeared in many TOGs. This problem was addressed by aggregating clusters in which more than a two thirds of sequences overlapped. This yielded 32K TOGAs. As an evaluation of ortholog clusters, 1,091 TOGs containing sequences from a minimum of 14 species were analyzed using GO (gene ontology) [7] assignments and found consistent. A phylogenetic tree analysis of selected TOGs also found the ortholog clusters to be consistent.

Kyoto Encyclopedia of Genes and Genomes (KEGG)

The KEGG database [44] is an attempt to computationally represent the complete knowledge about genes, information pathways of interaction biomolecules, cells and organisms. This database also consists of ortholog clusters organized into ortholog tables for the purpose of functionally annotating genes. So, members of an ortholog table consists of isofunctional genes. When the database was introduced, such genes were identified using a variety of gene-family specific procedures. Orthologs were extracted using a combination of computational methods and expert curation, so they do not

obey a single definition of orthology. In essence, the ortholog tables could be considered as a repository of orthologs detected by multiple researchers using methods they considered appropriate for the gene-family of their expertise.

In 2000, KEGG introduced a graph-based gene clustering algorithm [31], which was used to refine and augment the ortholog group tables. This clustering algorithm uses pair-wise sequence similarity along with positional coupling of genes in the genome. The conserved gene clusters are likely to represent functionally coupled genes, such as operons for co-expression, and those encoding physically interacting subunits. A conserved gene cluster is defined as a group of homologous genes located at contiguous positions in the genomes of multiple organisms. These are identified by constructing a pair-wise similarity (a cutoff of 100 was used for the Smith-Waterman local alignment algorithm) matrix of genes, with genes arranged in the matrix according to their position on the chromosome/genome. A diagonal stretch of 1's in the matrix corresponds to a conserved gene cluster. Gene clusters obtained by pairwise genome comparison are extended to multiple genomes. Similarity scores between two gene clusters is defined as the number of best hit gene pairs where multiple pairs involving the same node are combined in order to avoid counting paralogs more than once. To merge clusters they use a P-quasi complete linkage method which satisfies the condition that any member in one cluster has linkages to at least P% of all the members in the group [32]. This procedure was used on 17 genomes (12 bacteria, 4 bacteria and yeast).

Inparanoid

As described earlier, distinguishing the orthologs from the paralogs is a major issue in ortholog detection. A solution to this was proposed in Inparanoid [68]. It introduced the concept of in-paralogs and out-paralogs for distinguishing between genes whose duplicates are limited to one genome (in-paralog) from the out-paralogs which are results of duplication prior to the the speciation event between the two organisms. It relies on the BBH approach but interprets the results with a higher resolution. Beginning with two genomes, A and B, pairwise similarities for $A-A$, $A-B$, $B-A$, and $B-B$ are computed and those sequences which score above a certain threshold are retained. Besides

the BBH relationship, it requires that at least half of the larger sequence be involved in the pair-wise alignment. This is used for avoiding short domain level matches - ortholog sequences are expected to maintain similarity over the majority of their length. The in-paralogous sequences are identified as sequences from the same species that are more similar to the main ortholog than to any other sequence from other species. Since in-paralogs are likely to retain the same biological function, they are added to the same group as their main orthologs. Furthermore, overlapping ortholog clusters are resolved based on how the scores for neighbors for competing candidate clusters overlap. This work focused on only two genomes, and fly and worm genomes were used for analysis.

B.4.3 Phylogenetic tree based approaches

Phylogenetic tree based computational methods used for finding orthologs are based on reconciled trees [34, 62, 63], where a new tree is constructed that reconciles the gene tree with the species tree. These methods assume the knowledge of the species tree and construct the gene tree from the sequence data. So, the phylogenetic tree based approach requires solving three subproblems - obtaining a collection of homologous sequences for determining orthologous and paralogous relationships, constructing the optimal gene tree, and finally reconciling the gene tree with the species tree. These methods assume that homologous sequences are provided as an input. The latter two problems are computationally inefficient, so most methods for determining orthologous relationships, resort to estimating the confidence intervals for a given topology of the gene tree using the bootstrap method [28], keeping the one with the highest confidence interval.

OrthoStrapper

OrthoStrapper [79] uses bootstrapped trees to detect orthologous relationships in two groups, which may be two species or two groups of species. The algorithm takes a multiple alignment of sequences as an input and uses it as data for computing the support values for orthology assignments. The bootstrap trees are inferred from the original multiple alignment by sampling with replacement. For each bootstrap alignment a tree

is generated and analyzed for orthologs giving a support value to those ortholog assignments. The support values for all ortholog assignments are added up as new trees are calculated from bootstrap alignments. When the process terminates, all possible pair-wise orthology assignments are given a score and the one with the highest score is picked. Graphical results of OrthoStrapper are presented using OrtholoGUI [39].

OrtholStrapper is tested on simulated data generate by the program Rose [78], which generates sequence families using a substitution matrix for amino acids and the default parameters for insertion-deletion. Rose simulates evolution by evolving a sequence following a guidance tree. In this experiment, with a cut off of 50% support cut off, the program identified around 95% of true orthologs and detected upto 10% false positives. OrthoStrapper was also tested on real data comprising of 114 families containing worm-mammalian orthologs, where 1000 bootstrap trees were generated per group and the program performed better than tree reconciliation techniques.

HOPs

Hierarchical grouping of Orthologous and Paralogous Sequences (HOPs) [77] is a database of orthologous protein domains in Pfam [76, 75, 9]. Orthology is inferred in a hierarchic system of phylogenetic subgroups using ortholog bootstrapping [79]. To avoid frequent errors stemming from horizontally transferred genes in bacteria, analysis in HOPs is limited to ortholog detection in eukaryotes, and it performs ortholog detection at the level of groups of species, or lineages (treating them as pseudo-species). Furthermore, orthology analysis is carried out only between species (pseudo-species) at the same level. The eukaryotic level is split into clades of Metazoa, Viridiplantae and Fungi. Some species which do not belong into finer levels are assigned to upper levels, for instance a species which does not belong to any of the metazoan divisions considered would be considered at the metazoan level for analysis.

Orthology assignments are calculated using OrthoStrapper [79] discussed above. It analyzes a set of bootstrap trees instead of the optimal tree of orthologs. The algorithm detects orthologous relationships in two (groups of) species; the frequency of orthology assignments in the bootstrap trees can be interpreted as a confidence value for the

possible orthology of two proteins.

RIO

RIO (Resampled Inference of Orthologs) [96] is a procedure for automated phylogenomics using explicit phylogenomic inference. Its analyses are performed over bootstrap resampled phylogenetic trees to estimate reliability of orthology assignments. RIO is based on an algorithm to infer speciation and duplication events on a gene tree by comparison to a trusted species tree by Zmasek and Eddy [95].

The protein family database Pfam [10] is the starting point of orthologous analysis. The multiple sequence alignments and profile HMMs in Pfam together with the neighbor joining tree method of [69] are used to estimate orthologous relationships through bootstrap resampling. A query sequence is aligned with existing multiple alignments in Pfam using hmmalign from the HMMER package [http://hmmer.wustl.edu/] and resulting multiple-alignment is resampled with replacement. Maximum likelihood pairwise distance matrices are calculated for each of the resampled multiple alignments using a model of amino acid substitution. Using these resampled multiple alignments, unrooted gene trees are inferred using the neighbor joining method, which are then converted to rooted trees using [95]. For each node of a rooted gene tree a speciation or duplication event is inferred by comparing it to a trusted species tree. Using the tree and events estimated on each gene tree, each subjected sequence is designated as ortholog or paralog of the query sequence and a final confidence value is output for relationships based on consistencies in relationships obtained from the rooted trees.

The effectiveness of this method has been evaluated on the *Arabidopsis thalina* and *Caenorhabditis elegans* genomes. Almost half of each of the proteome could be analyzed based on matches found on Pfam and the results were found to be found to confirm the existing annotations for sequences. A caveat with this approach is the bias in detecting the orthologs from families whose members are available in Pfam.

B.5 Analysis

B.5.1 Expert based approaches

Expert based ortholog detection methods are useful to improve our understanding of new organisms, but they require manual work which cannot match with the increasing amounts of genomic data. The other issue is related to subjectivity on part of researchers which is valuable for ortholog families involving experts foci which allow considering diverse evolutionary rates and recent complex evolutionary knowledge for relevant organisms. On the other hand, it could lead to inaccurate results for other protein families. So, expert based studies are feasible and valuable only for limited studies, but impractical not only for large scale but also for distantly related organisms. Nonetheless, such studies can serve as excellent benchmarks for automated ortholog detection methods.

B.5.2 Clustering based approaches

The completely automatic clustering methods are suitable for screening of ortholog clusters in large amounts of genomic data. One of the critical issues with these methods is the reliability of the resulting ortholog clusters. Most clustering methods incorporate the ortholog model in the form of the BBH relationship. The BBH relationship is appropriate for a couple of simple genomes which do not contain multi-domain proteins of large number of duplicates. This concept is discussed below in detail.

The concept of BBH recognizes only one-to-one relationship among orthologs where as the true orthologous relationships can be many-to-many. One-to-one relationships may exist for closely related organisms which diverged recently, but for distantly related organisms, the orthologous relationships are complicated by the interspersed nature of speciation and duplication events. The BBH is appropriate for detecting pair-wise orthologous relationships in the prokaryotes where paralogs are not high in number. Even if consider the distantly related prokaryotes ortholog detection, using BBH is still possible for most gene families. For eukaryotes, the BBH concept is not robust for detecting orthologous relationships, because of frequent presence of high numbers of

paralogs. In the presence of paralogs, the ortholog detection may not only be incorrect but this concept can fail to detect a relationship when one exists. Another pitfall with BBH is that it requires complete genomes.

Apart from these conceptual deficiencies in BBH, there arise problems when this concept is used in practice. Most methods based on BBH use Blast to estimate similarity among sequences. Blast often returns as the highest scoring hit a protein that is not the nearest phylogenetic neighbor of the query sequence [47]. By the BBH definition, if the the forward blast yields a paralogous best hit but the reciprocal blast recovers an actual ortholog, both pairs will be excluded. A solution to this problem is to use rsd (reciprocal smallest distance) algorithm that preserves the safeguard of reciprocal genome queries but is less vulnerable to exclusion of orthologs due to identification of a paralog in one blast hit direction [89]. The rsd algorithm relies on global sequence alignment and maximum likelihood estimation of evolutionary distance to detect orthologs between two genomes; rsd finds many putative ortholog misssed by BBH because it is less likely to be misled by the presence of a close paralog.

B.5.3 Phylogenetic tree based approaches

Phylogenetic tree based techniques compare the phylogenetic tree and gene tree to discover orthologs. This approach suffers because, in many cases, good estimates of species trees are not available, or the species may be completely unresolved, for instance, species tree for human, worm and fly is unresolved. It also requires a set of homologous sequences which can be best determined by clustering of similar sequences. The most critical part is the construction of the gene tree, which is a computationally expensive problem.

Appendix C

Applications in Computer Vision

We describe two specific problems in computer vision where our multipartite graph clustering approach has been applied. Both of these were formulated as classification problems in which the multipartite graph framework was used for representing the relationships in the input data and for extracting data relevant for effectively training a supervised classifier. Since the choice and details of the classification process are beyond the scope of multipartite graph clustering, the presentation here is limited to the motivation leading to the problem formulation and how the method developed in chapters 2 and 3 was adapted for solving the problems. The the description for the gait recognition problem in section C.1 is adapted from [93] while that for the discriminative patch selection, described in section C.2, is adapted from [88].

C.1 Gait Recognition

In human subject identification using gait analysis, an individual's gait is captured by cameras as a sequence of image frames. The goal is to construct a supervised classifier that can be trained using the gait sequences, and later use it for identifying subjects using their gait frame sequence. Although, the entire sequence of gait frames is important, in many cases, some particular stages in the gait cycle can uniquely characterize the subject. So, for constructing a supervised classifier, the gait frames characteristic of the subject are more informative [93]. Such informative frames can be identified by comparing the target subject's gait frames to those of others. But this may lead to extraction of noisy frames which may not be representative of the target subject. An alternative approach is to exclude gait frames from a subject that are similar to those from many other subjects. Such an exclusion will retain gait frames

specific to the subject including those that characteristic of the gait style.

The problem of finding gait frames that characterize and identify a subject is solved as an inverse problem where we identify clusters of similar frames from multiple subjects. Frames in such clusters are uninformative for constructing a supervised classifier for identifying an individual subject, and are therefore removed from training the classifier.

The input for the gait recognition problem is a sequence binary silhouette images which do not include texture and color information. This additional information is deliberately left out so that the recognition process can be based entirely on the gait information. The binary silhouette images are transformed into feature space(s) where the recognition task is carried out. Then, given the representation of frames in a feature space, our goal is to divide the training data frames from multiple subjects into two classes: a set with subject-specific frames, and another with non-subject-specific ones (shared by many subjects). It is likely that filtering out frames that are similar (or, uninformative) across subjects would increase the accuracy of classification. Finding such uninformative frames can be cast as a clustering problem on a graph representing similarity between frames from different subjects.

In a traditional clustering method, similar frames will be grouped into a cluster. However, in the gait recognition problem we would like to ignore similarity between frames from the same subject as we are interested in finding clusters containing similar frames from different subjects. Such a similarity structure is captured by a multipartite graph, where we denote a subject as a partite set and frames from a subject as vertices in the corresponding partite set. The multipartite graph representation is suitable because it only considers the similarity relationships between frames from different subjects. A cluster in such a graph is a subset of frames from multiple subjects. The presence of clusters in the multipartite graph brings out similarities between frames from different objects, so clusters containing very similar frames from a large number of subjects can be considered as uninformative.

C.1.1 Determining Uninformative Clusters

For extracting gait frames that are similar in gait cycles of k subjects, we represent the problem as a k-partite graph. The frames from the subject i are represented by the vertices in the partite set $V_i,\ i \in \{1, 2, \ldots, k\}$. We construct an undirected weighted multipartite graph $G = (V, E, W)$, where $V = \cup_{i=1}^{k} V_i$, and $E \subseteq \cup_{i \neq j} V_i \times V_j$ is the set of weighted, undirected edges connecting vertices from different partite sets. The weight $w_{ij} \in W$ on the edge $e_{ij} \in E$ represents the similarity between the vertices i and j. Having done so, the problem of finding groups of similar frames from multiple subjects can be modeled by the multipartite quasi-clique problem, described in Chapter 2.

We used two criteria for determining which clusters should be labeled as uninformative: (a) frames within such clusters should be highly similar to each other, (b) these clusters should contain frames from most of the subjects. Recall that the clustering procedure also outputs the score value $F(H^*)$ for the cluster H^*. Since the score value is the aggregated similarity between a frame and frames from all the other subjects in the cluster (see Chapter 2 for details), it would, in general, be larger for larger clusters. So for extracting the clusters of very similar frames, we used the average similarity value (density) of a cluster defined as $F(H^*)/|H^*|$, instead of $F(H^*)$ for estimating homogeneity within a cluster. Furthermore, clusters with large average similarity values, but containing frames from just a few subjects can be informative as they can provide hints for classification by narrowing down the possibilities for recognition. Consequently, for the extracting the uninformative frames, we used only those clusters whose average similarity was above a certain threshold, and which contained frames from most of the subjects.

C.1.2 Classification using Informative Frames

Clusters obtained in the previous section were used for constructing a hierarchical structure for the entire input data, based on which a probabilistic classifier was built (see [93] for details). The first level focused on determining whether a probe frame was informative, whereas identification within each informative cluster was specified in the

second level. In addition, for a group of frames from an unknown subject, we aggregate the identification results from all informative ones to reach a final subject identification.

The method was applied to the silhouette images from Human ID Gait Challenge Database collected at the University of South Florida (USF) in May and November 2001. The comparison recognition rates with those from other methods, reported previously showed the proposed approach considerably outperformed the baseline algorithm [70] in 10 out of the 12 cases, while matching in another one. Also, the method achieved a much higher recognition rate for some of the most difficult experimental settings in the challenge set. In comparison to the results from the HMM approach [80], our method performs better or equivalently in 5 of the 7 experimental conditions.

Our method significantly outperformed all other previously reported methods in cases when the subject walks on a different surface or at a different time of the compared to that in the training data. This shows that our method has more prediction power for unknown walking conditions. Eliminating the common frames across subjects enables us to avoid training on noisy (i.e., uninformative from the classification perspective) frames, thereby resulting in a higher recognition rate. In contrast, approaches that attempt to capture strict dynamics of a walking style, such as the HMM based methods, cannot isolate the uninformative frames, and hence may perform poorly.

C.2 Discriminative Patch Selection

In an object recognition task where an image is represented as a constellation of image patches, many patches often correspond to the cluttered background. If such patches are used for object class recognition, they can adversely affect the recognition rate. We describe the use of our multipartite graph clustering method for selecting image patches which characterize the target object class and are capable of discriminating between the positive images containing the target objects and the complementary negative images.

C.2.1 Problem Description

An important subtask in object recognition lies at the interface between feature extraction and its use for recognition. It involves deciding which extracted features are most suitable for improving recognition rates [90], because the initial set of features is large, and often features are redundant or correspond to the clutter in the image. Finding such actual object features reduces the dimensionality of the problem and is essential to learn a representative object model to enhance recognition performance. So, we focus on selecting the "best" features from the already extracted image features that are both exclusive and well represented in different images of the target object.

The problem we address can be stated as: Given a pool of local features (patches) extracted from a set of labeled training images containing positive and negative images of the target class, how can we choose (in an unsupervised way) the best features representing the object.

Naively, it seems plausible to select patches from both the negative images and positive images, and classify a test image in the class to which it is closest. However, the space of negative images, devoid of any instance of the target image, is prohibitively large to allow any generalization on the negative class. So, one should rather train the classifier on the positive images using patches which are common to most of the positive images. This is based on the assumption that salient features of the target object will be present and captured from most of the positive images, and form a good representation for it. However, a potential side effect of focusing entirely on the positive images is the selection of undesirable patches corresponding to the background. To avoid this one can simultaneously consider the positive and negative images for selection of the image patches as representing both the saliencies of the target object while at the same time being exclusive or discriminative for the positive class. We have used two approaches for this purpose and the combinatorial approach based on a multipartite graph is described below.

C.2.2 Patch selection using Multipartite graph

Suppose we are given a set $V^+ = \{V_1^+, V_2^+, \ldots, V_p^+\}$ of p images (positive class) containing the instances of the target object, and a set $V^- = \{V_1^-, V_2^-, \ldots, V_n^-\}$ of n images (negative class) which do not contain the target object. Recall that any arbitrary image is represented as a set of m salient image patches, so the image i^{th} from the positive class can be denoted as $V_i^+ = \{v_{i1}^+, v_{i2}^+, \ldots, v_{is}^+, \ldots, v_{im}^+\}$, where v_{is}^+ is the s^{th} image patch. Further, we also use V^+ to denote the set of all patches in V_1^+ through V_p^+, i.e. $V^+ = \cup_{\ell=1}^{p} V_\ell^+$; similarly, $V^- = \cup_{\ell=1}^{n} V_\ell^+$. The usage will become clear from the context.

We are interested in finding the subset of image patches from the set V^+ which are very similar to each other and, at the same time, distant from those in the set V^-. Furthermore, while finding image patches that characterize the target object, it is best to focus on similarities between image patches across different instances of the target object, rather than similarities between patches from the same image although they may be very similar. These two informal requirements can be conveniently expressed in a multipartite graph representation of the similarities between image patches from different images, as shown in Fig. C.1. The right part of this figure shows an undirected edge weighted vertex weighted multipartite graph, $G = (V^+, E, W, N)$, with p partite sets V_1^+ through V_p^+ so that, as described earlier, $V^+ = \cup_{\ell=1}^{p} V_\ell^+$. The edges in the set $E \subseteq \cup_{i \neq j} V_i^+ \times V_j^+$, represent similarity between the image patches from different images while the weight w_{ab} on the edge connecting the vertices corresponding to the patches a and b represents the strength of their similarity. Each vertex in V^+ is also associated with a weight $N : V^+ \rightarrow \mathbb{R}^+$ which reflects its aggregated similarity to images patches in V^-. For any vertex $i \in V^+$, its vertex weight $N(i)$ is calculated as $N(i) = \sum_{s \in V^-} m_{is}^2$, where m_{is} is the similarity between image patch i and the image patch s from a negative image.

We consider the situation where the negative images in the training set do not contain any instance of the target object, and the positive images contain exactly one instance of the target object. Of course, it is possible to model more complex situations

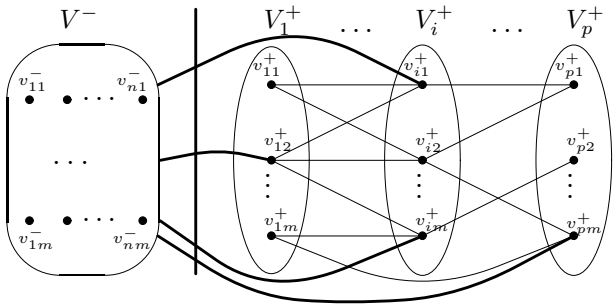

Figure C.1: A multipartite graph representation for expressing similarity relationships between the image patches. Ellipse corresponding to V_i^+ represents the i^{th} instance of target image, and the m points inside this ellipse represent the image patches from this image. The patches from the images that do not contain the target object are represented inside the oval V^- without distinguishing between the images of those patches. The straight lines connecting the images patches across different instances of images represent the weighted similarity between them, while the thick curved lines represent the aggregated (weighted) similarity between an image patch from a positive image to all image patches in the negative class. For visual clarity, weights are not shown on the edges.

where the positive images contain multiple instances of the target object. However, we have focused on modeling the simpler situation.

We would like to find the subset of image patches which are characteristic of positive images and distant from patches in the negative images. In other words, we want to find a subset $H \subseteq V^+$ (so, $H = \cup_{\ell=1}^{p} H_\ell$, where $H_\ell \subseteq V_\ell^+$) of image patches from the positive images in which patches are very similar to each other and at the same time different from image patches in the negative images. To achieve this, any subset H is assigned score the $F(H)$ which measures the degree of similarity between the patches from different images in H and also their distinction from patches in V^-. This score is designed to be higher, as described later, for desirable subsets. The best subset, H^* is the globally optimal solution for the criterion formulated below (see chapter 2 for a detailed description). The crucial step in the formulation of the optimization problem is the definition of the linkage function. The linkage function used for solving this problem is given next.

Note that we only have pairwise similarities between the image patches from different images and we must design the element to set similarity function $\pi(i, H)$ using only these. Also, since H is a multipartite subset, i.e., $H = \cup_{\ell=1}^{p} H_\ell$ where $H_\ell \subseteq V_\ell^+$ is a

subset of patches from the image V_ℓ^+. If w_{ij} is the similarity value between the image patch from i from the image $I(i)$ and the image patch j from the image $I(j)$, then we can define the linkage function as:

$$\pi(i, H) = \sum_{\substack{\ell=1 \\ \ell \neq I(i)}}^{p} \left(\sum_{j \in H_\ell} w_{ij}^2 - \sum_{k \in V_\ell^+ \setminus H_\ell} w_{ik}^2 \right) - \beta N(i) \tag{C.1}$$

where $\beta \in \mathbb{R}^+$ is a constant factor for scaling $N(i)$, the weight associated with the vertex (i), defined as the aggregated similarity of i to the patches from the negative images. This scaling factor β serves to account for imbalance between the number of positive and negative instances of the target object. The first term ($\sum_{j \in H_\ell} w_{ij}^2$) in the linkage function aggregates the similarity of patch i from image $I(i)$ to patches from other images present in H. The second term ($\sum_{k \in V_\ell^+ \setminus H_\ell} w_{ik}^2$) estimates how patch i is related to patches not included in H_ℓ. A large positive value of the linkage function $\pi(i, H)$ indicates that i is very similar to patches in H and different from the patches in the negative images, or the patches from the positive images not included in H. According to this definition of linkage function, the optimal solution, H^* corresponds to a collection of image patches from different positive images, each of which is highly similar to each other (as the least similar patch is highly similar to other patches) and very different from the patches in the negative images. So, such a formulation indeed serves our purpose of selecting characteristic from the positive class which also discriminative the negative class.

This problem formulation gives us one subset of similar image patches from the positive images, and likely corresponds to some characteristic in the target object in those images. However, often an object has multiple salient characteristics, and these disjoint subset of patches corresponding to different characteristics of the target object can be found by sequentially extracting clusters until we get optimal solutions with large values.

C.2.3 Classification based on discriminative patches

A statistical formulation was also used for selecting discriminative patches as those patches from the positive images which consistently appear in multiple instances of the positive images but only rarely appear in the negative images. Intuitively, if an individual image patch from a positive image performs well in recognizing the images of the target object, a combination of a number of such image patches is likely to enhance the overall performance [88]. Following the selection of characteristic image patches from the positive images, a probabilistic method was used for object class recognition. The selected image patches were used, simultaneously, to build a probabilistic model for the object class and the object reference frame.

The results of using the above sequence of methods with the results obtained from state of the art methods on the images in the Caltech database (`http://www.vision.caltech.edu/html-files/archive.html`) showed that a sequential combination of the combinatorial patch selection method followed by the statistical patch selection method yielded the best results [88]. The results were competitive with the best reported results, and the use of patch selection methods improved the recognition rates of the classifier by a significant margin, and often surpassed the recognition rates reported by using the other methods.

References

[1] Abascal, F., and Valencia, A. Clustering of proximal sequence space for identification of protein families. *Bioinformatics 18*, 7 (2002), 908–921.

[2] Abascal, F., and Valencia, A. Automatic annotation of protein function based on family identification. *Proteins 53* (2003), 683–692.

[3] Abello, J., Resende, M. G., and Sudarsky, S. Massive quasi-clique detection. *LATIN'02* (2002).

[4] Altschul, S., Madden, T., Schaffer, A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res. 25*, 17 (1997), 3389–3402.

[5] Andreeva, A., Howorth, D., Brenner, S. E., Hubbard, T. J. P., Chothia, C., and Murzin, A. G. SCOP database in 2004: refinements integrate structure and sequence family data. *Nucleic Acids Res. 32*, 1 (2004), 226–229.

[6] Arabie, P., and Boorman, S. Multidimensional scaling of measures of distance between partitions. *J Math Psychol 10* (1973), 148–203.

[7] Ashburner, M., Ball, C., Blake, J., Botstein, D., Butler, H., Cherry, J., Davis, A., Dolinski, K., Dwight, S., Eppig, J., Harris, M., Hill, D., Issel-Tarver, L., Kasarskis, A., Lewis, S., Matese, J., Richardson, J., Ringwald, M., Rubin, G., and Sherlock, G. Gene ontology: tool for the unification of biology. *Nat Genet. 25* (2000), 25–29.

[8] Bateman, A., Birney, E., Cerruti, L., Durbin, R., Etwiller, L., Eddy, S. R., Griffiths-Jones, S., Howe, K. L., Marshall, M., and Sonnhammer, E. L. L. The Pfam protein families database. *Nucleic Acids Res. 30*, 1 (2002), 276–280.

[9] Bateman, A., Birney, E., Durbin, R., Eddy, S. R., Finn, R. D., and Sonnhammer, E. L. Pfam 3.1: 1313 multiple alignments and profile hmms match the majority of proteins. *Nucleic Acids Res. 27*, 1 (1999), 260–262.

[10] Bateman, A., Birney, E., Durbin, R., Eddy, S. R., Howe, K. L., and Sonnhammer, E. L. L. The Pfam protein families database. *Nucleic Acids Res. 28*, 1 (2000), 263–266.

[11] Bateman, A., Coin, L., Durbin, R., Finn, R. D., Hollich, V., Griffiths-Jones, S., Khanna, A., Marshall, M., Moxon, S., Sonnhammer, E. L. L., Studholme, D. J., Yeats, C., and Eddy, S. R. The Pfam protein families database. *Nucleic Acids Res. 32*, 1 (2004), 138–141.

[12] Berman, H., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T., Weissig, H., Shindyalov, I., and Bourne, P. The protein data bank. *Nucleic Acids Res. 28* (2000), 235–242.

[13] Boeckmann, B., Bairoch, A., Apweiler, R., Blatter, M.-C., Estreicher, A., Gasteiger, E., Martin, M. J., Michoud, K., O'Donovan, C., Phan, I., Pilbout, S., and Schneider, M. The SWISS-PROT protein knowledgebase and its supplement TrEMBL in 2003. *Nucleic Acids Res. 31*, 1 (2003), 365–370.

[14] Bru, C., Courcelle, E., Carrère, S., Beausse, Y., Dalmar, S., and Kahn, D. The ProDom database of protein domain families: more emphasis on 3d. *Nucleic Acids Res. 33*, 1 (2004), 212–215.

[15] Castleden, C. K., Aoki, N., Gillespie, V. J., MacRae, E. A., Quick, W. P., Buchner, P., Foyer, C. H., Furbank, R. T., and Lunn, J. E. Evolution and function of the sucrose-phosphate synthase gene families in wheat and other grasses. *Plant Physiology 135* (2004), 1753–1764.

[16] Chervitz, S. A., and Aravind, L. Comparison of the complete protein sets of worm and yeast: orthology and divergence. *Science 282* (1998), 2022–2028.

[17] Cormen, T., Leiserson, C., Rivest, R., and Stein, C. *Introduction to Algorithms, Second Edition.* The MIT Press, 2001.

[18] Cort, J. R., Koonin, E. V., Bash, P. A., and Kennedy, M. A. A phylogenetic approach to target selection for structural genomics: solution structure of YciH. *Nucleic Acids Res. 27* (1999), 4018–27.

[19] Darlingto, H., Rouster, J., Hoffmann, L., Halford, N., Shewry, P., and Simpson, D. Identification and molecular characterization of hordoindolines from barley grain. *Plant Mol Biol. 47* (2001), 785–794.

[20] Dawande, M., Keskinocak, P., Swaminathan, J. M., and Tayur, S. On bipartite and multipartite clique problems. *J. Algorithms 41* (2001), 388–403.

[21] Dong, Q., Schlueter, D., and Brendel, V. PlantGDB, plant genome database and analysis tools. *Nucleic Acids Res. 32* (2004), D354–D359.

[22] Duret, L., Mouchiroud, D., and Gouy, M. HOVERGEN, a database of homologous vertebrate genes. *Nucleic Acids Res. 22* (1994), 2360–2365.

[23] Dutilh, B. E., Huynen, M. A., and Snel, B. A global definition of expression context is conserved between orthologs, but does not correlate with sequence conservation. *BMC Genomics 7* (2006).

[24] Eddy, S. R. A review of the profile HMM literature from 1996-1998. *Bioinformatics 14* (1998), 755–763.

[25] Eisen, J., and Wu, M. Phylogenetic analysis and gene functional predictions: phylogenomics in action. *Theor Popul Biol. 61* (2002), 481–487.

[26] et al., K. Beyond complete genomes: from sequence to structure and function. *Curr. Opin. Struct. Biol. 8* (1998), 355–363.

[27] Everitt, B. S., Landau, S., and Leese, M. *Cluster Analysis, 4th edition.* Oxford University Press Inc., 2001.

[28] Felsenstein, J. Confidence limits on phylogenies: an approach using the bootstrap. *Evolution 39* (1985), 783–791.

[29] Fitch, W. M. Distinguishing homologous from analogous proteins. *Systematic Zoology 19* (1970), 99–113.

[30] Fredman, M. L., and Tarjan, R. E. Fibonacci heaps and their uses in improved network optimization. *J. ACM* (1987), 596–615.

[31] Fujibuchi, W., Ogata, H., Matsuda, H., and Kanehisa, M. Automatic detection of conserved gene clusters in multiple genomes by graph comparison and p-quasi grouping. *Nucleic Acids Res. 28*, 2 (2002), 4096–4036.

[32] Fujibuchi, W., Ogata, H., Matsuda, H., and Kanehisa, M. Automatic detection of conserved gene clusters in multiple genomes by graph comparison and P-quasi grouping. *Nucleic Acids Res. 28*, 2 (2002), 4096–4036.

[33] Galperin, M. Y., and Koonin, E. V. Who's your neighbor? new computational approaches for functional genomics. *Nature Biotechnology 18* (2000), 609–613.

[34] Goodman, M., Czelusniak, J., Moore, G., Romero-Herrera, A., and Matsuda, G. Fitting the gene lineage into its species lineage, a parsimony strategy illustrated by cladograms constructed from globin sequences. *Systematic Zoology 28* (1979), 132–163.

[35] He, X., and Zhang, J. Rapid subfunctionalization accompanied by prolonged and substantial neofunctionalization in duplicate gene evolution. *Genetics 169* (2005), 1157–64.

[36] Henikoff, J. G., Greene, E. A., Pietrokovski, S., and Henikoff, S. Increased coverage of protein families with the blocks database servers. *Nucleic Acids Res. 28* (2000), 228–230.

[37] Henikoff, S., and Henikoff, J. G. Automated assembly of protein blocks for database searching. *Nucleic Acids Res. 19* (1991), 6565–6572.

[38] Hochbaum, D. S. Approximating clique and biclique problems. *J. Algorithms 29* (1997), 174–200.

[39] Hollich, V., Strom, C. E., and Sonnhammer, E. L. OrthoGUI: graphical presentation of orthostrapper results. *Bioinformatics 18*, 9 (2002), 1272–1273.

[40] Hubert, L. J., and Arabie, P. Comparing partitions. *Journal of Classification 2* (1985), 193–218.

[41] Itoh, M., Nakaya, A., and Kanehisa, M. Identification of orthologous groups in KEGG/SSDB by considering domain structure. *Genome Informatics 13* (2002), 342–343.

[42] Kaessmann, H., Zllner, S., Nekrutenko, A., and Li, W.-H. Signatures of domain shuffling in the human genome. *Genome Res. 12* (2002), 1642–50.

[43] Kamvysselis, M., Patterson, N., Birren, B., Berger, B., and Lander, E. Whole-genome comparative annotation and regulatory motif discovery in multiple yeast species. In *RECOMB* (2003), pp. 157–166.

[44] Kanehisa, M., Goto, S., Hattori, M., Aoki-Kinoshita, K., Itoh, M., Kawashima, S., Katayama, T., Araki, M., and Hirakawa, M. From genomics to chemical genomics: new developments in kegg. *Nucleic Acids Res. 34* (2006), D354–357.

[45] Kellogg, E. A. Relationships of cereal crops and other grasses. *Proceedings of National Academy of Science 95* (1998), 2005–2010.

[46] Koonin, E. V. How many genes can make a cell: The minimal-gene-set concept. *Annu. Rev. Genomics Hum. Genet. 1* (2000), 99–116.

[47] Koski, L. B., and Golding, G. B. The closest BLAST hit is often not the nearest neighbor. *Journal of Molecular Biology 52* (2001), 540–542.

[48] Lee, Y., Sultana, R., Pertea, G., Cho, J., Karamycheva, S., Tsai, J., Parvizi, B., Cheund, F., Antonescu, V., White, J., Holt, I., Liang, F., and Quackenbush, J. Cross-referncing eukaryotic genomes: TIGR orthologous gene alignments (TOGA). *Genome Research 12*, 3 (2002), 493–502.

[49] Lin, Y., Case, J., Pao, H.-K., and Burnside, J. Predicted secondary structure slightly enhances ortholog detection. *RECOMB* (2004).

[50] Lupas, A., Van Dyke, M., and Stock, J. Predicting coiled coils from protein sequences. *Science 252* (1991), 1162–1164.

[51] Makalowski, W., and Boguski, M. Evolutionary parameters of the transcribed mammalian genome: An analysis of 2,820 orthologous rodent and human sequences. *Proceedings of National Academy of Science 95* (1998), 9407–9412.

[52] Matula, D. W., and Beck, L. L. Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM 30*, 3 (1983), 417–427.

[53] Mirkin, B., and Muchnik, I. Induced layered clusters, hereditary mappings, and convex geometries. *Appl. Math. Lett. 15* (2002), 293–298.

[54] Mirkin, B., and Muchnik, I. Layered clusters of tightness set functions. *Appl. Math. Lett. 15* (2002), 147–151.

[55] Mullat, J. E. Extremal subsystems of monotone systems i. *Automation and Remote Control* (1976), 758–766.

[56] Mullat, J. E. Extremal subsystems of monotone systems ii. *Automation and Remote Control* (1977), 1286–1294.

[57] Murzin, A., Brenner, S. E., Hubbard, T., and Chothia, C. SCOP: A structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology 247* (1995), 536–540.

[58] Nelson, K., and et. al. Evidence for lateral gene transfer between archaea and bacteria from genome sequence of thermotoga maritima. *Nature 399* (1999), 323–329.

[59] Ouyang, M., Case, J., and Tirunagaru, V. 565 triplets of chicken, human, and mouse candidate orthologs. *Journal of Molecular Biology 57* (2003), 271–281.

[60] Overbeek, R., Fonstein, M., D'Souza, M., Pusch, G. D., and Maltsev, N. The use of gene clusters to infer functional coupling. *Proceedings of National Academy of Science 96* (1999), 2896–2901.

[61] Overbeek, R., Larsen, N., Pusch, G. D., D'Souza1, M., Jr, E. S., Nikos Kyrpides, Michael Fonstein, N. M., and Selkov, E. WIT: integrated system for high-throughput genome sequence analysis and metabolic reconstruction. *Nucleic Acids Res. 28*, 1 (2000), 123–125.

[62] Page, R. D. Maps between trees and cladistic analysis of historical associations among genes, organisms, and areas. *Systematic Biology 43* (1994), 58–77.

[63] Page, R. D. GeneTree: comparing gene and species phylogenies using reconciled trees. *Bioinformatics 14*, 9 (1998).

[64] Pearson, W. R. Rapid and sensitive sequence comparison with fastp and fasta. *Methods in Enzymology 183* (1990), 63–98.

[65] Peeters, R. The maximum edge biclique problem is NP-complete. *Discrete Applied Mathematics 131* (2003), 651–654.

[66] Pei, J., Jiang, D., and Zhang, A. On mining cross graph quasi-cliques. *KDD'05* (2005).

[67] Rand, W. M. Objective criterion for the evaluation of clustering methods. *J. Am. stat. Assoc. 66* (1971), 846–850.

[68] Remm, M., Strom, C. E., and Sonnhammer, E. L. Automatic clustering of orthologs and In-paralogs from pairwise species comparisons. *Journal of Molecular Biology 314* (2001), 1041–1052.

[69] Saitou, N., and Nei, M. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Mol. Biol. Evol. 4* (1987), 406–425.

[70] Sarkar, S., Phillips, P., Liu, Z., Vega, I., Grother, P., and Bowyeri, K. The HumanID gait challenge problem: Data sets, performance, and analysis. *IEEE PAMI 27* (2005), 162–177.

[71] Schoof, H., Zaccaria, P., Gundlach, H., Lemcke, K., Rudd, S., Kolesov, G., Mewes, R. A. H., and Mayer, K. MIPS arabidopsis thaliana database (MAtDB): an integrated biological knowledge resource based on the first complete plant genome. *Nucleic Acids Res. 30* (2002), 91–93.

[72] Smith, T. F., and Waterman, M. S. Identification of common molecular subsequences. *Journal of Mol Biol. 147* (1981), 195–197.

[73] Solovyev, V., and Shahmuradov, I. PromH: promoters identification using identification orthologous genomic sequences. *Nucleic Acids Res. 31*, 13 (2003), 3540–3545.

[74] Song, R., Llaca, V., Linton, E., and Messing, J. Sequence, regulation, and evolution of the maize 22-kD alpha zein gene family. *Genome Res. 11* (2001), 1817–1825.

[75] Sonnhammer, E. L., Eddy, S. R., Birney, E., Bateman, A., and Durbin, R. Pfam: multiple sequence alignments and hmm-profiles of protein domains. *Nucleic Acids Res. 26*, 1 (1998), 320–322.

[76] Sonnhammer, E. L., Eddy, S. R., and Durbin, R. Pfam: A comprehensive database of protein domain families based on seed alignments. *PROTEINS: Structure, Function and Genetics 28* (1997), 405–420.

[77] Storm, C. E., and Sonnhammer, E. L. Comprehensive analysis of orthologous protein domains using the HOPS database. *Genome Research 0* (2003), 2353–2362.

[78] Stoye, J., Evers, D., and Meyers, F. Rose: generating sequence families. *Bioinformatics 14* (1998), 157–163.

[79] Strom, C. E., and Sonnhammer, E. L. Automated ortholog inference from phylogenetic trees and calculation of orthology reliability. *Bioinformatics 18*, 1 (2002), 92–99.

[80] Sundaresan, A., RoyChowdhury, A., and Chellapa, R. A hidden markov model based framework for recognition of humans from gait sequences. In *ICIP* (2003).

[81] Tatusov, R., Fedorova, N., Jackson, J., Jacobs, A., Kiryutin, B., Koonin, E., Krylov, D., R, R. M., Mekhedov, S., Nikolskaya, A., Rao, B., Smirnov, S., Sverdlov, A., Vasudevan, S., Wolf, Y., Yin, J., and Natale, D. The COG database: an updated version includes eukaryotes. *BioMed Central Bioinformatics* (2003).

[82] Tatusov, R. L., Galperin, M. Y., Natale, D. A., and Koonin, E. V. The COG database: a tool for genome-scale analysis of protein functions and evolution. *Nucleic Acids Res. 28*, 1 (2000), 33–36.

[83] Tatusov, R. L., Koonin, E. V., and Lipman, D. J. A genomic perspective on protein families. *Science 278* (1997), 631–637.

[84] Tomii, K., and Kanehisa, M. A comparative analysis of ABC transporters in complete microbial genomes. *Genome Research 8*, 10 (1998), 1048–59.

[85] Vashist, A., Kulikowski, C., and Muchnik, I. Ortholog clustering on a multipartite graph. In *Proc. 5th Workshop on Algorithms in Bioinformatics (WABI) Lecture Notes in Computer Science series* (2005), vol. 3692, pp. 328–340.

[86] Vashist, A., Kulikowski, C., and Muchnik, I. Screening for ortholog clusters using multipartite graph clustering by quasi-concave set function optimization.

In *Proc. 10th Intl. Conf. on Rough Sets, Fuzzy Sets, Data Mining and Granular Computing (RSFDGrC) Lecture Notes in Computer Science* (2005), vol. 3642, pp. 409–419.

[87] Vashist, A., Kulikowski, C., and Muchnik, I. Screening for ortholog clusters using multipartite graph clustering by quasi-concave set function optimization. In *Proc. of the 10th Intl. Conf. Research in Computational Molecular Biology (RECOMB), Lecture Notes in Computer Science* (2006).

[88] Vashist, A., Zhao, Z., Kulikowski, C., Muchnik, I., and Elgammal, A. Discriminative part selection by combinatorial and statistical methods for part-based object recognition. IEEE CVPR Workshop on Beyond Patches, 2006.

[89] Wall, D., Fraser, H., and Hirsh, A. Detecting putative orthologs. *Bioinformatics 19*, 13 (2003), 1710–1711.

[90] Weber, M., Welling, M., and Perona, P. Unsupervised learning of models for recognition. In *ECCV (1)* (2000), pp. 18–32.

[91] Weiner, J., Beaussart, F., and Bornberg-Bauer, E. Domain deletions and substitutions in the modular protein evolution. *FEBS J. 273* (2006), 2037–47.

[92] Wootton, J., and Federhen, S. Statistics of local complexity in amino acid sequences and sequence databases. *Computers and Chemistry 17* (1993), 149–163.

[93] Zhang, R., Vashist, A., Kulikowski, C., Muchnik, I., and Metaxas, D. A new combinatorial approach to supervised learning: application to gait recognition. In *Proc. 2nd Workshop on Analysis and Modeling of Faces and Gestures, Lecture Notes in Computer Science* (2005), vol. 3723, pp. 55–69.

[94] Zheng, X. H., Lu, F., Wang, Z.-Y., Zhong, F., Hoover, J., and Mural, R. Using shared genomic synteny and shared protein functions to enhance the identification of orthologous gene pairs. *Bioinformatics 21* (2005), 703–710.

[95] Zmasek, C. M., and Eddy, S. R. A simple algorithm to infer gene duplication and speciation events on a gene tree. *Bioinformatics 17*, 9 (2001), 821–828.

[96] Zmasek, C. M., and Eddy, S. R. RIO: Analyzing proteomes by automated phylogenomics using resampled inference of orthologs. *BMC Bioinformatics 3*, 14 (2002).

Vita

Akshaya Kumar Vashist

2007 Ph. D. in Computer Science, Rutgers University

1994-98 M. E. Computer Science & Engg., Indian Institute of Science, INDIA.

1991-94 B. Sc. (Hons.) Physics, University of Delhi, INDIA.

1999-2001 Teaching Assistant, Department of Computer Science, Rutgers University

2001-2003 Graduate Fellow, Waksman Institute Fellowship, Rutgers University

2003-2005 Teaching Assistant, Department of Computer Science, Rutgers University

Publications

2006 Protein function annotation using ortholog clusters extracted from incomplete genomes using combinatorial optimization. Akshay Vashist, Casimir Kulikowski and Ilya Muchnik. *10^{th} Intl. Conf. on Research in Computational Molecular Biology (RECOMB 2006), LNCS Vol.3909, pp.99-103.*

2005 Screening for ortholog clustering by Quasi-concave set function optimization. Akshay Vashist, Casimir Kulikowski and Ilya Muchnik. *Proceedings of the Tenth Intl. Conf. on Rough Sets, Fuzzy Sets, Data mining and Granular Computing (RSFDGrC 2005), LNCS Vol.3642, pp.409-419.*

2005 Ortholog Clustering on a Multipartite Graph. Akshay Vashist, Casimir Kulikowski and Ilya Muchnik. *Proc. Workshop on Algorithms in Bioinformatics (WABI 2005), LNCS, Vol.3692, pp.328-340.* & IEEE/ACM Trans. on Computational Biology and Bioinformatics (2006).

2005 A new combinatorial approach to supervised learning: application to gait recognition. Rong Zhang, Akshay Vashist, Casimir Kulikowski, Ilya Muchnik and Dimitris Metaxas. *Proc. IEEE Analysis of modeling of faces and gestures (AMFG 2005), LNCS Vol.3723, pp.55-69*