ABSTRACT

ELLISON, VICTORIA MARIE. Using Linear Programming based Exploratory Techniques in Gene Expression Consensus Clustering. (Under the direction of Yahya Fathi.)

There exist many options when it comes to clustering data including: what clustering algorithm to use, what preselected parameters of certain clustering algorithms to use, and which results of certain clustering algorithms to use. Consensus clustering is useful in generating a clustering which has the most consensus with multiple differing clusterings of the same data set. However, most consensus clustering algorithms do not provide an exact optimal solution, provide the user with options as to how many clusters are in the returned consensus clustering, or reflect the nested nature of the clusters, which is a feature inherent to many data sets. These features are especially important in the analysis of gene expression profile data.

To meet the needs of these features we propose a divisive hierarchical consensus clustering algorithm, which we call CCA-β. This consensus clustering algorithm makes use of a special type of cut, which we also propose, called a β-ratio cut. We demonstrate how this β-ratio cut is a natural extension of the cluster ratio cut that incorporates finding the nested nature of a graph.

We also propose a heuristic for CCA-β, which we call CCA-ROPPPA. This heuristic applies a proposed parametric programming algorithm to a proposed parametric linear programming formulation. We create this proposed parametric linear programming formulation by modifying the BILP formulation of the Median Partition Problem (MPP) proposed by Grotschel and Wakabayashi. We create this proposed parametric programming algorithm by modifying the optimal partition parametric programming algorithm (OPPPA) proposed by Berkelaar et al.

We prove that under certain conditions a.) our CCA-ROPPPA heuristic returns a consensus clustering that is equivalent to a solution to the minimum cluster ratio problem and b.) our CCA-ROPPPA heuristic returns a set of consensus clusterings that could have been returned by our CCA-β algorithm. Using real gene expression profile data sets and artificial data sets we create sets of clusterings for which we would like to find a consensus clustering(s). We then apply CCA-ROPPPA to these sets of clusterings and assess the frequency with which these aforementioned conditions were met.

We also compare CCA-ROPPPA to two other consensus clustering algorithms. We use the same real gene expression profile data sets and artificial data sets and the same sets of clusterings created from these data sets. We then apply CCA-ROPPPA and the two other consensus clustering algorithms to these sets of clusterings and compare a.) the nature and quality of the consensus clusterings returned and b.) the execution times of the algorithms.

Finally, we use an argument that is based on linear programming duality and similar to

the argument that Yeh et al. use to demonstrate the rationale of their cluster ratio cut to: a.) show that the β-ratio cut is a natural extension of the cluster ratio cut and b.) demonstrate the rationale of our β-ratio cut.

Using Linear Programming based Exploratory Techniques
in Gene Expression Consensus Clustering

by
Victoria Marie Ellison

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Operations Research

Raleigh, North Carolina

2017

APPROVED BY:

Steffen Heber	Michael Kay
Amy Langville	Osman Ozaltin

Yahya Fathi
Chair of Advisory Committee

BIOGRAPHY

Victoria was born and raised in Southern Maryland on the shores of the Chesapeake Bay. An early passion for problem solving combined with growing up so close to a center of change, progress, and civic engagement like Washington D.C. lead her to pursue a career path in applied mathematics where she hoped to be able to apply her problem solving skills to solving real world problems.

She completed a B.S. in Mathematics with a minor in Statistics at James Madison University in 2008. During this time she conducted undergraduate research applying and developing statistical techniques to applications in finance and healthcare. Also during this time she applied statistical techniques to applications in transportation and population studies via her internships at the Washington Metropolitan Area Transit Authority and the United States Census Bureau respectively.

Wanting to further her studies, she then completed a M.S. in Mathematics at the College of Charleston in 2010. While Victoria enjoyed statistics, she found herself desiring a career path more focused in applied deterministic modeling and having more opportunities for student outreach. It was during this time that she discovered the field of Operations Research and her passion for teaching. She found that she enjoyed the challenge of breaking down complex topics and making it readily understandable for anyone to understand. It was then she decided to purse a Ph.D. in Operations Research at North Carolina State University.

The topic for this dissertation started from research which began in her master's program at the College of Charleston. Victoria really enjoyed the particular combination of Linear Programming and data mining techniques such as consensus clustering, so she pursued an extension of this research topic for her doctoral dissertation.

Victoria worked in industry for five years while pursuing her doctoral degree. At Centice Corporation she developed signal processing algorithms using statistical and mathematical techniques to identify presence and concentration of narcotics in an unknown substance by analyzing the substance's unique Raman spectral fingerprint. At MaxPoint Interactive she developed mathematical algorithms to recommend websites for online advertisements and to predict the probability an internet user will purchase an advertised product. At TiVo Research and Analytics she built automated analytics tools to assess TV ad campaign performance.

ACKNOWLEDGEMENTS

The path toward as large an undertaking as a PhD can become narrow and easy to slip from at times. I would like to thank my family, friends, and mentors for helping to widen that path and push me forward more than I could have ever done alone.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Gene Expression Analysis

With the advent of new technology, physicians and researchers have increasingly been able to extract and gain access to a great plentitude of gene related data. Analyzing such data has proven to be extremely useful to researchers in diagnosing various diseases and in discovering gene targeted disease therapies.

The most common type of data analyzed for these purposes is gene expression profile data. Gene expression is the process by which a gene expresses itself in protein synthesis. In genetics, it is how a genotype gives rise to an organism's phenotype. The expression of a gene within a certain cell is typically analyzed by first measuring the amount of mRNA corresponding to that gene in the cell, and then using these mRNA levels to infer the expression of the gene. Gene expression analysis has become a very important tool in the study of diseases. For example, the expression levels of certain genes in a diseased patient may be higher or lower than the gene expression levels of a patient without this disease. The amount of pertinent genetic data in an organism or sample can range from 30,000 to 3 billion. Therefore data mining techniques are essential in decoding this overabundance of data.

In selecting the appropriate data mining technique to analyze gene expression data, specific properties about the data must be taken into consideration. First as goes the curse of dimensionality with all large data sets, the ideal data mining algorithm will maintain a fast computation time while providing meaningful results to the user. Since disease related research is not usually needed for real time analysis, some allowances can be made in the way of computation time to allow for better and more meaningful results. Furthermore, as is the case with a lot of data mining applications the researcher does not often know ahead of time what their exact goal for analysis is. That is why exploratory techniques which convey different facets of information about the data are ideal as well.

More specifically the underlying structure of gene expression data has been shown to exhibit a high level of clusterability. In a broad sense clusterability means that similar objects tend to group together in relatively homogenous groups called clusters. In terms of samples, this means that similar samples will group together in clusters displaying relatively similar gene expression data profiles. In terms of genes, this means that similar genes will group together in clusters having similar sample data. Because of this, cluster analysis is one of the most often used methods in extracting information about samples and genes.

Gene expression profiles are typically arranged in a real-valued expression matrix $\mathbf{\Phi} = [\phi_{ia}]_{i=1,a=1}^{n,F}$ where ϕ_{ia} is the expression level of the a^{th} gene of sample i. The expression matrix can be used to either cluster the samples, using the F genes as features, or cluster the genes using the n samples as features. Clustering the samples, for instance, can assist in the diagnosis of disease and may even reveal disease subtypes, as studies have shown that clustering gene expression profiles of patients with the same disease can often reveal two or more sub-types of this disease [Tam99]. Here it is assumed, that each patient has only one of these disease sub-types. Being able to identify and study the gene expression profiles of these disease sub-types can be helpful in the development of drugs or therapies aimed at eradicating these diseases. Alternatively, clustering the genes can help determine which genes are responsible for the development and progression of certain diseases and thus also play a pivotal role in the development of more intelligent drugs or therapies. Since the number of genes often greatly outnumbers the number of samples, we choose to focus on clustering the samples because our proposed methods, while useful for extracting pertinent additional information, are not as well equipped for handling data sets with a large number of objects as other methods are.

There have been many different clustering methods proposed to classify gene expression data. Due to the nature of gene expression data, there are special features that a good gene expression clustering algorithm must consider. Jiang et al. [Jia04] describe these as the following. First, there is much about gene expression patterns that is still unknown to scientists. Therefore, a good gene expression clustering algorithm will require as little prior knowledge about the data set as possible. Second, a good gene clustering algorithm should be able to handle highly connected graphs and embedded (nested) clusters. Next, gene expression data is often wrought with an excessive amount of noise, thus a good clustering algorithm will be well equipped at cleaning out any unnecessary information from the data. Finally, a good gene clustering algorithm will reveal other information about the clusters, such as how close clusters are to one another. We aim to especially satisfy the first two of these requirements in our research.

1.2 Gene Expression Clustering

There have been many suggested clustering algorithms that try to satisfy these gene expression related requirements. The k-means clustering algorithm [Llo82] has often been used due to its computational efficiency and simplicity. However, the initial placement of the centroids of the k-means algorithm will often produce different clustering results each time the algorithm is run. In addition, the user must preselect the number of clusters he or she wants in advance, which would thus require a preexisting knowledge of the data set. Finally, the k-means algorithm may be sensitive to noise [Sme02] [She01].

Self Organizing Map (SOM) [Koh84] approaches to clustering are also commonly used. SOM methods are beneficial in that they can provide insightful 2-d and 3-d visualizations of clustering results of a high dimensional data set. Herrero et al. [Her01] and Tamayo et al. [Tam99] suggest that SOM clustering methods are less sensitive to noise, however they are more sensitive to irrelevant data points. If there are too many irrelevant data points, such as genes with invariant patterns, then these data points will pervade many of the clusters.

Hierarchical clustering methods are also used very frequently in gene expression analysis. These methods are well adapted for data sets with nested clusters and can provide a dendrogram or tree graph that visually displays the results. Eisen et al. [Eis98] proposed an agglomerative hierarchical clustering algorithm known as UPGMA (Unweighted Pair Group Method with Arithmetic Mean) and used a particularly effective way of graphically displaying the clustering results. The drawbacks of hierarchical clustering methods are their high compuational complexity [Tam99] and their sensitivity to small perturbations [Tam99].

Graph-theoretic clustering algorithms are also commonly used. In respect to gene expression profile sample clustering, which is what we discuss in this paper, each sample is represented as a node in a graph. In some methods, a weighted edge is placed between each pair of samples. The weight between the samples is determined by a similarity metric. One such similarity metric would be the Euclidean distance between the gene expression levels of two samples. That is, the weight of the edge between samples i and j would be the Euclidean distance between $\mathbf{\Phi}_{i*}$ and $\mathbf{\Phi}_{j*}$ [SS00] [XK01]. Other methods, simply place a 0 or a 1 weight on each edge based on whether the selected similarity metric surpasses a specified threshold [BD99] [HS00]. Shamir and Sharan [SS00] propose their graph-theoretic clustering algorithm known as CLICK (CLuster Identification via Connectivity Kernels), which is based on finding the minimum edge weight cut in a graph representation of the weighted adjacency matrix. The down side of using graph-theoretic clustering algorithms is that information is lost in converting the gene expression profile data into a weighted adjacency matrix.

Although using a clustering algorithm to cluster gene expression data has proven to be very useful in unlocking hidden connections and information about disease and biological functions, there is no single algorithm which has proven to work best for all data sets as evidenced by the abundance of new clustering algorithms proposed each year. As shown above, each algorithm has its own strengths and weaknesses in their ability to meet the specialized needs of a comprehensive gene expression data analysis. Furthermore the exact goal of cluster analysis is at best a fuzzily defined notion. As there are a great number of clustering algorithms proposed in the literature, there are almost an equal amount of proposed objective functions aimed at optimizing and assessing the results of a clustering algorithm. However, defining the goal of clustering as one fixed objective function to optimize may in and of itself leave out a facet of information about the data that may also prove useful.

To further complicate the matter, many clustering algorithms like the k-means algorithm result in only a local optimal solution to the objective function they are trying to optimize. Many of these algorithms are initialized with random starting points and will thus produce different results because of this. Again, to say one local optimal solution is better than another is not necessarily the right mindset to take on in clustering as one local optimal result may provide some useful information that the other may not, and vice versa. Furthermore the question of how many clusters to consider is also a confounding factor in choosing the 'right' clustering algorithm for gene expression analysis. While some algorithms require the user to preselect the resulting number of clusters in the clustering returned by the algorithm, other algorithms will return a clustering with a number of clusters chosen by the model. In either case, the question arises as to whether a resultant clustering with k clusters is the most informative. Would a resultant clustering with $k-1$ or $k+1$ clusters also provide useful information? In many cases the answer may be yes.

Because of all these confounding factors in conducting gene expression cluster analysis it is prudent for a researcher to consult the results of several different clustering algorithms run with different initializations and input parameters (if the algorithm allows it). However, the question of how to best amalgamate the results of all these algorithms is a difficult problem in and of itself.

1.3 Consensus Clustering

Consensus clustering is a recently proposed approach to answering this question. Also called cluster aggregation or ensemble clustering, it takes as input the clustering results of several clustering algorithms and/or different clustering results of the same clustering algorithm. We will call these differing clustering results the **input clusterings** of a consensus clustering algorithm. Typically all the input clusterings are attained by clustering the same initial data set. Using the input clusterings as input, a consensus clustering algorithm returns a **consensus clustering** which achieves the most amount of consensus amongst the input clusterings.

Consensus clustering can be particularly beneficial in analyzing gene expression data as it has the power to create a consensus clustering result which may emphasize the strong input clustering results while downplaying weak or unwanted input clustering results. For instance, as we discussed in section 1.2, gene expression profile data is often noisy and the k-means algorithm is often sensitive to noise. Executing the k-means algorithm multiple times and using a consensus clustering algorithm to combine the results has the power to minimize the effect of this noise.

One of the drawbacks of using a consensus clustering algorithm, particularly for gene expression analysis, is that it adds even more computational time to the analysis. Furthermore, since most consensus clustering algorithms only make use of the input clusterings and not the original data (used to create the input clusterings), information is inevitably lost in further processing. Useful information is also inevitably lost as most consensus clustering algorithms return only a single consensus clustering with no additional information such as the nested nature of the clusters.

In general, consensus clustering methods are comprised of two steps: 1. the generation step and 2. the consensus function step [VPRS11]. The generation step generates a set of input clusterings $\{\pi^1, ..., \pi^T\}$ by executing one or more clustering algorithms a total of T times. Important questions to consider in this step are listed below.

1. Which clustering algorithms should be used?

2. If a clustering algorithm can produce different results, how many times should this algorithm be run?

3. If a clustering algorithm requires a preselected number of clusters k, how many clusters should be preselected?

4. Which subset of objects should be included in a given clustering algorithm (if not all)?

5. Which subset of object features should be used in the given clustering algorithm (if not all)?

These considerations are important as the consensus clustering returned by a consensus clustering algorithm is influenced by the quality and diversity of the input clusterings.

The second step of a typical consensus clustering method uses a consensus function to find a consensus clustering that achieves the most consensus among the input clusterings. In the literature there are two main types of consensus function approaches: object co-occurrence and median partition [VPRS11].

1.3.1 Co-Occurence Consensus Clustering

The object co-occurrence approach is based on assigning an object to a cluster using a voting method which considers a.) the number of times a given object is assigned to a given cluster over all the input clusterings or b.) the number of times two given objects are assigned to the same (and different) clusters over all the input clusterings.

A common co-occurence consensus clustering approach first creates an $n \times n$ coassociation matrix $\boldsymbol{S} = [s_{ij}]$ where:

$$s_{ij} = \frac{1}{T} \sum_{t=1}^{T} \delta(\pi^t(o_i), \pi^t(o_j)), \tag{1.1}$$

where $\Pi = \{\pi^1, ..., \pi^T\}$ is the set of input clusterings, $\delta(\pi^t(o_i), \pi^t(o_j)) = 1$ if o_i and o_j are assigned to the same cluster in the input clustering π^t, and $\delta(\pi^t(o_i), \pi^t(o_j)) = 0$ if o_i and o_j are assigned to different clusters in π^t. After forming this matrix, typically an additional clustering algorithm is performed upon $\boldsymbol{S}$, where s_{ij} represents the similarity between o_i and o_j.

While there exists a great deal of literature surrounding co-occurence consensus clustering algorithms, we highlight the two following works in particular because a.) they use co-occurence consensus clustering algorithms for gene expression profile data and b.) they use a hierarchical clustering algorithm to recluster the co-association matrix. This is important for the purpose of comparison as the consensus clustering algorithm that we propose in this thesis is also a co-association consensus clustering algorithm, also uses hierarchical clustering algorithms, and also used on gene expression profile data.

In the first work, by Monti et al. [Mon03], a consensus based clustering algorithm is proposed for gene expression analysis that uses both hierarchical clustering and SOM clustering as the primary algorithms and perturbs the input data to create differing clustering results. They create a normalized weighted adjacency $\boldsymbol{S} = [s_{ij}]$ matrix between the samples, where s_{ij} is created using equation 1.1. They define their consensus score as $\sum_i \sum_j s_{ij} x_{ij}$ where $x_{ij} = 1$ if samples o_i and o_j are placed in the same cluster in the consensus clustering algorithm $x_{ij} = 0$ otherwise. They then perform a hierarchical clustering algorithm on the $\boldsymbol{S}$ to recluster the data. They use the adjusted Rand score [Ran71] [HA85] (which we will define in chapter 2) to assess the quality of their algorithm. This consensus clustering algorithm is more so for the purpose of assessing clustering quality of the original clustering algorithm and whether the correct number of clusters were picked by this original algorithm. Furthermore, the results are subject to the the sampling method and the clustering algorithm used to recluster $\boldsymbol{S}$.

In the second work, by Kashef and Kamel [KK08], the researchers combine gene expression clustering results from both the k-means algorithm and the bisecting k-means algorithm to produce a hierarchical clustering dendogram. Their proposed algorithm generates input clusterings as part of the consensus clustering algorithm so no time or information is lost and greater degree of consensus is reached. However, this consensus clustering algorithm only uses clustering results from the k-means and the bisecting k-means algorithms. It is not flexible in being able to provide a way to combine input clusterings generated from different clustering algorithms. They also do not provide a metric for assessing the consensus reached by the returned consensus clustering.

Other common consensus clustering methods that use object co-occurence consensus functions include relabeling and voting methods, coassociation matrix methods, graph and hypergraph methods, information theory methods, finite mixture models, LAC algorithms, and fuzzy methods [VPRS11]. As with clustering algorithms, each consensus clustering algorithm has its merits. The benefit of object co-occurence methods is that they typically have a low computational complexity, thereby allowing for the consensus clustering of larger data sets. However, they are lacking in mathematical rigor as they are not attempting to find an exact solution to an optimization problem. An exact optimization problem can give a quantitative score to assess how much overall consensus the consensus clustering had with the input clusterings, thereby yielding extra information about the data. Furthermore, its rich mathematical structure has the capability of being further exploited to yield extra information such as the nature of nested clusters and cluster closeness.

1.3.2 Median Partition Problem

The second main consensus clustering function is more mathematically rigorous as it selects a consensus clustering by solving an optimization problem which we define below.

Definition Let $O = \{o_1, ..., o_n\}$ be a set of objects and let $\{\pi^t\}_{t=1}^T$ be a set of partitions (input clusterings) of O. Let $\mathbf{\Pi}(O)$ be the set of all possible partitions (input clusterings) that can be made from O. Let $\Gamma : \mathbf{\Pi}(O) \times \mathbf{\Pi}(O) \mapsto R$ be a similarity function. Then the **Median Partition Problem (MPP)** [Lec94] returns a partition $\pi^* \in \mathbf{\Pi}(O)$ such that:

$$\pi^* = \underbrace{argmin}_{\pi \in \mathbf{\Pi}(O)} \sum_{t=1}^{T} \Gamma(\pi, \pi^t) \tag{1.2}$$

Researchers have proposed many functions for Γ, however one of the most widely used and studied functions is the symmetric difference distance function (or the Mirkin distance function) [Mir96].

Definition Let $O = \{o_1, ..., o_n\}$ be a set of objects and let π and π' be two partitions (input clusterings) of O. The **Mirkin distance** between π and π' is defined as:

$$\Gamma(\pi, \pi') = N_{01} + N_{10}, \tag{1.3}$$

where N_{01} is the number of object pairs (o_i, o_j) for which o_i and o_j are clustered together in partition π', but not in partition π and similarly N_{10} is the number of object pairs (o_i, o_j) for which o_i and o_j are clustered together in partition π, but not in partition π'.

In other words, the Median Partition Problem that uses the Mirkin distance seeks to minimize the total disagreements between the each input clustering and the consensus clustering. Wakabayashi [Wak86] and Krivanek and Moravek [KM86] both proved in different ways that the Median Partition Problem with the Mirkin distance is NP-hard.

Filkov and Skiena [FS03] use heuristics of the Median Partition Problem to consensus-cluster gene expression profile data. They are able to consensus-cluster up to 1000 objects using their algorithms. They use the Rand index [Ran71] and proposed p-value to assess the agreement of the consensus clustering and all input clusterings. However, their algorithms are limited in providing only one clustering solution to the user.

Other common consensus clustering approaches which use the Median Partition Problem include Non-negative Matrix Factorization (NMF) methods, genetic algorithms, and kernel methods [VPRS11]. Unfortunately these methods rely upon a heuristic to solve the problem. Thus the theoretical rigor of the Median Partition Problem is lost to make way for higher data crunching capabilities.

1.4 IP Formulation of the Median Partition Problem with Mirkin Distance

Grotschel and Wakabyashi [GW89] introduced the IP formulation shown below. They prove that it yields the exact optimal solution to the Median Partition Problem with Mirkin distance function.

$$\begin{array}{lll}
\min & \sum_{1\le i<j\le n} w_{ij}x_{ij} & \\
\text{st} & x_{ij}+x_{jk}-x_{ik}\le 1 & \text{for } 1\le i<j<k\le n \\
 & x_{ij}-x_{jk}+x_{ik}\le 1 & \text{for } 1\le i<j<k\le n \\
 & -x_{ij}+x_{jk}+x_{ik}\le 1 & \text{for } 1\le i<j<k\le n \\
 & x_{ij}\in\{0,1\} & \text{for } 1\le i<j\le n
\end{array}$$

$$x_{ij} = \begin{cases} 1 & \text{if } o_i \text{ and } o_j \text{ are clustered together in the consensus clustering} \\ 0 & \text{otherwise} \end{cases}$$

The objective function coefficients are defined as $w_{ij} = w_{ij}^- - w_{ij}^+$ where w_{ij}^+ is the number of input clusterings that place o_i and o_j in the same cluster and w_{ij}^- is the number of input clusterings that place o_i and o_j in different clusters.

Note, another way to define w_{ij} is as follows:

$$w_{ij} = \sum_{t=1}^{T} \epsilon(\pi^t(o_i), \pi^t(o_j)), \tag{1.4}$$

where $\epsilon(\pi^t(o_i), \pi^t(o_j)) = -1$ if o_i and o_j are assigned to the same cluster in π^t and $\epsilon(\pi^t(o_i), \pi^t(o_j)) = 1$ if o_i and o_j are assigned to different clusters in π^t.

We call this the **Grotschel Wakabayashi IP formulation (GWIP)**. We refer to the **Grotschel Wakabayashi LP formulation (GWLP)** as GWIP with relaxed binary constraints.

The benefit of using the GWIP for consensus clustering is that it solves for an exact solution to the Median Partition Problem. The model has a rigorous mathematical structure, which in chapter 8 allowed us to discover a theoretical relationship to another existing problem in the literature in which there exist several fast heuristics.

However, to the best of our knowledge no researchers have attempted to extract additional information about the data using the mathematical structure of the model. Instead they choose to either solve the IP to optimality or accept a heuristic solution to the IP and conduct no further analysis.

A drawback of using the GWIP for the purpose of consensus clustering is that it has a relatively large number of variables and constraints, specifically $3\binom{n}{3}$ constraints and $\binom{n}{2}$ decision variables. Furthermore there is no guarantee that GWLP will yield a binary optimal solution. Thus without using any specialized IP solving techniques for this problem (such as cutting plane methods or branch and bound techniques), instances of this problem for $n > 50$ becomes intractable.

Because of this, several researchers have proposed various techniques to increase problem tractability for larger n. Tushaus [Tüs83] introduces a branch and bound method that only works for small values of n. Wakabayashi and Grotschel [GW89] give a cutting plane method which is able to solve GWIP for n in the hundreds. These methods only discuss ways in which to make cuts or the branching schema more effective in the sense that as few LPs as possible have to be solved in order to get a binary optimal solution. However, what these methods do not take into account is that GWLP itself has a large number of constraints without any additional cuts or branch and bound restrictions and thus solving GWLP even once may prove to be an overly costly operation in and of itself.

Downing et al. [Dow10] propose a variety of constraint reduction techniques which increase efficiency in solving GWLP. Because the consensus clustering heuristic we propose in chapter 3 involves solving multiple linear programming formulations that are only slight modifications of GWLP, we employ some of these techniques in our proposed consensus clustering heuristic as well (chapter 3). These techniques are summarized below.

- Solving the linear relaxation using a relaxed feasible region and then iteratively adding *all* of the constraints violated by the current optimal solution to the problem. An optimal solution to the original linear relaxation is reached when there are no more constraints violated by the current optimal solution to the problem.

- Solving the linear relaxation using a relaxed feasible region and then iteratively adding *a random sample* of the constraints violated by the current optimal solution to the problem. An optimal solution to the original linear relaxation is reached when there are no more constraints violated by the current optimal solution to the problem.

- Iteratively deleting constraints that correspond to basic variables in the dual of the current relaxed feasible region, provided that the optimal objective function value at the current iteration increased from the value of the optimal objective function in the previous iteration. They explain why this method works in the literature.

In their numerical tests, the majority of the objective function coefficients vectors $[w_{ij}]$ they use in GWLP do not produce an optimal solution to the GWLP. We note here that we also conduct numerical tests using the GWLP in chapter 7, however the majority of the objective function coefficients vectors $[w_{ij}]$ they use in GWLP *do* produce an optimal solution to the GWLP. Although it is not within the scope of this dissertation to explore why this discrepancy occurred, this could be due to the fact that the input clusterings we use to create $[w_{ij}]$ are the results of clustering the same original data set multiple times whereas the input clusterings Downing et al. to create $[w_{ij}]$ are independent sets of categorical data. Thus our input clustering sets may have had an overall higher degree of consensus with one another than those used by Downing et al.

In addition, these researchers choose to focus less on producing an exact optimal solution and instead introduce a rounding schema which approximates the binary optimal solution in the event that the optimal solution to the linear relaxation is fractional. This rounding schema greatly reduces execution time in comparison to solving the GWIP by using branch and bound or cutting plane techniques. By using these efficiency improvements and approximate rounding techniques, Downing et al. [Dow10] were able to solve a consensus clustering data set of size $n = 2048$ within a reasonable amount of time. However, due to the fact that a rounding schema had to be introduced, this method is now only a heuristic to the MPP.

As discussed earlier, all of the research regarding the GWIP is concentrated at improving efficiency and reducing computational time in producing an optimal or near optimal solution to the GWIP. That is, the researchers produce one consensus clustering from GWIP and no further analysis is conducted on the data. This singular result can be a drawback to the purpose of gene expression analysis for the following reasons.

1. **Not the Desired Number of Clusters** First, the number of clusters obtained via the GWIP cannot be specified. Depending on the purpose of the analysis, researchers may have a predefined idea as to how many clusters they desire in the consensus clusterings. For instance, a data set we use in our numerical tests in chapter 7 consists of gene expression profiles of four well established types of central nervous system tumors as well as gene expression profiles of normal cerebella tissue. For the purpose of diagnosis and classification it befits the researcher to desire each of the input clusterings as well as the consensus clustering to have five clusters. However, a set of input clusterings (each clustering having five clusters) was generated and GWLP returned a consensus clustering with eight clusters. While for the purpose of research and development this may actually provide useful information about further divisions of disease subtypes, for the purpose of quick disease classification this clustering with an excess of clusters may prove to be confusing. Thus it befits the researcher to have more consensus clustering result options, specifically consensus clustering results which yield fewer or greater than the number of clusters returned by GWIP.

2. **No Reflection of Nested Clusters** As discussed as a special consideration for gene expression analysis, the clusters formed by gene expression profiles are often nested in

nature. A singular clustering result returned by the GWIP does not reflect this nested nature. A hierarchical model which could convey how and when certain objects split away from other objects based on some metric of *consensus cluster* similarity would be beneficial in reflecting this, however no such models exist for this which use the GWIP.

1.5 Thesis Summary

In this thesis we slightly modify GWIP such that it still maintains the rich mathematical structure and reduced computation time techniques discussed in 1.4, but also allows for the discovery of nested consensus clusterings. In doing so, this thesis has five main contributions.

1. First we introduce a new cut, which we call a β-ratio cut. Using linear programming duality theory, we demonstrate a.) how our β-ratio cut is a natural extension of the cluster ratio cut [Yeh95] and b.) the rationale for why this cut is useful for the purpose of hierarchical consensus clustering.

2. Next we introduce a divisive hierarchical consensus clustering algorithm, which we call CCA-β that uses this β-ratio cut. Instead of one consensus clustering, this algorithm produces a *set* of nested consensus clusterings.

3. Then we introduce two heuristics for CCA-β, which we call CCA-OPPPA and CCA-ROPPPA. We call our methods CCA-OPPPA and CCA-ROPPPA heuristics of CCA-β because they will not always produce a set of nested consensus clusterings solutions that would have been a possible output of CCA-β. However, we show that when *certain conditions* are satisfied by the results of our CCA-OPPPA and CCA-ROPPPA heuristics, they will produce a set of solutions that would have been a possible output of CCA-β. Our CCA-OPPPA and CCA-ROPPPA heuristics apply a parametric programming algorithm to a parametric LP formulation P_λ that we introduce (which is a slight modification of GWIP). While CCA-OPPPA uses an existing parametric programming algorithm [Ber96], CCA-ROPPPA uses a proposed parametric programming algorithm that is a slight modification of this existing parametric programming algorithm.

4. Because our CCA-ROPPPA heuristic does not always return a set of nested consensus clusterings that could have been returned by CCA-β, our fourth contribution is to test how often they do, using sets of input clusterings generated from real gene expression profile data sets and artificial data sets.

5. Also using these sets of input clusterings generated from real gene expression profile data sets and artificial data sets, our fifth contribution will be to compare the CCA-ROPPPA heuristic to two other consensus clustering algorithms.

In chapter 2 we give some background needed in order to discuss our proposed consensus clustering algorithm. We define existing graph theoretic terms such as partitions, cuts, and a specialized type of cut, known as the minimum cluster ratio cut [Yeh95]. We also introduce and define our proposed cut, the β-ratio cut. We formally define the clustering and consensus clustering problem. We also discuss a widely used class of clustering algorithms, namely divisive hierarchical clustering algorithms. We discuss a specific type of divisive hierarchical clustering algorithm (DHCA) which uses our β-ratio cut. We call this DHCA-β and use it in our consensus clustering algorithm that we introduce in chapter 3. Finally in chapter 2 we define the Uniform Multicommodity Flow Problem [LR99], which we will use to show the rationale for β-cuts in chapter 8.

In chapter 3 we introduce our proposed consensus clustering algorithm CCA-β and its proposed heuristics, CCA-OPPPA and CCA-ROPPPA. Section 3.1 details how our proposed divisive hierarchical consensus clustering algorithm (CCA$-\beta$), uses DHCA-β (discussed in 2.3). In section 3.2 we introduce CCA-OPPPA and CCA-ROPPPA, our proposed heuristics for CCA-β.

CCA-OPPPA and CCA-ROPPPA apply a parametric programming algorithm to a parametric LP formulation. In chapter 4 we introduce this parametric LP formulation, which we call P_λ, and discuss various properties of it. In chapter 5 we introduce the parametric programming algorithms we apply to P_λ in CCA-OPPPA and CCA-ROPPPA. In section 5.1 we introduce notation and show various properties regarding P_λ. This terminology, which includes finding the dual of P_λ, is needed in explaining the parametric programming algorithms that we discuss in 5.2 and 5.3. In section 5.2 we discuss a parametric programming algorithm found in the literature [Ber96], which we call OPPPA in this paper. Although the proof for why the algorithm works is too long to place in this dissertation, we briefly summarize how OPPPA works and give a few results related to the algorithm. The way that we present OPPPPA in 5.2 is made specific to our particular LP P_λ over the range $[0, \binom{n}{2})]$, however OPPPPA as described in the literature can be applied to any parametric LP formulation.

In 5.3 we introduce another parametric programming algorithm, which we call ROPPPA, that is a modification of OPPPA. In each iteration of OPPPA two LPs are solved. In 5.3.3 we discuss that when using our specific parametric LP P_λ, the second LP in each iteration has many more constraints than variables and the first LP in each iteration has many more variables than constraints. Thus we explain how we can use row generation techniques to more quickly solve the second LP in each iteration of OPPPA and use column generation techniques to solve the first LP in each iteration. By using the row generation techniques we introduce ROPPPA in 5.3.2 (using the column generation techniques are outside the scope of this dissertation, but are available for future research). Section 5.3.1 discusses the specific row generation techniques that we apply to the first LP in every iteration of ROPPPA. Section 5.4 discusses important proofs regarding OPPPA and ROPPPA.

In chapter 6 we show how our the results of our CCA-OPPPA and CCA-ROPPPA heuristics will produce CCA-β solutions if and only if the results satisfy certain conditions. The proofs we show in chapter 6 will be useful in chapter 8 when we demonstrate a.) how our β-ratio cut is a natural extension of the cluster ratio cut [Yeh95] and b.) the rationale for why this cut is useful for the purpose of hierarchical clustering.

In chapter 7, using real gene expression profile data sets and artificial data sets we create sets of clusterings for which we would like to find a consensus clustering(s). We then apply CCA-ROPPPA to these sets of clusterings and assess the frequency with which CCA-ROPPPA returns a set of consensus clusterings that could have been a result of CCA-β. Also in chapter 7, we compare CCA-ROPPPA to two other consensus clustering algorithms. We use the same real gene expression profile data sets and artificial data sets and the same sets of input clusterings created from these data sets. We then apply CCA-ROPPPA and the two other consensus clustering algorithms to these sets of clusterings and compare a.) the nature and quality of the consensus clusterings returned and b.) the execution times of the algorithms.

Specifically, in section 7.1 we discuss the two consensus clustering algorithms we will compare to CCA-ROPPPA. In section 7.2 we will discuss how we create input clustering sets that will be used as input for CCA-ROPPPA and the two other consensus clustering algorithms we test. In section 7.3 we give the specific computer, software, and LP solving options we used to execute all algorithms. In the appendix, for each of the input clustering sets that we created, we graphically display the results of the three consensus clusterings algorithms that used these input clustering sets as input. In section 7.4, we discuss how to interpret these graphs in the appendix. In section 7.5 we discuss the frequency with which CCA-ROPPPA returned a set of consensus clusterings that could have been a result of CCA-β. Finally in section 7.6 we compare CCA-ROPPPA to the two other consensus clustering algorithms we tested.

Finally, in chapter 8 we use an argument that is based on linear programming duality and similar to the argument that Yeh et al. [Yeh95] use to demonstrate the rationale of their cluster ratio cut to: a.) show that the β-ratio cut is a natural extension of the cluster ratio cut and b.) demonstrate the rationale of our β-ratio cut.

Chapter 2

BACKGROUND

In this chapter we introduce some important background to the problem which we intend to solve.

2.1 Graph Theory

2.1.1 Existing Graph Theoretic Terms

We first define some well-established graph theory terms. A **graph** $G = (V, E)$ is defined by a set of vertices (nodes) V and a set E of pairs of distinct vertices called edges. We say that a set of disjoint subsets $\{V_1, ..., V_r\}$ of V is a **partition** (**clustering**) of a graph $G = (V, E)$ if and only if $V_1 \cup ... \cup V_r = V$ and $V_s \cap V_t = \phi$ for all $s \neq t$. We use the term **cluster** to refer to a disjoint subset V_s in a given partition (clustering) $\{V_1, ..., V_r\}$.

A **complete graph** is a graph in which all pairs of nodes are linked by an edge. We denote a complete graph with n nodes as K_n. For a graph $G = (V, E)$ where $U \subset V$, $F \subset E$, and the vertices of each edge in F are also in U, then we say that $H = (U, F)$ is a subgraph of G. A **clique** of a graph $G = (V, E)$ is a subgraph of G that is a complete graph.

Given a graph $G = (V, E)$ we call a subset of edges $E(V_1, ..., V_r) \in E$ a **clique partition** if there is a partition $\{V_1, ..., V_r\}$ of V such that $E(V_1, ..., V_r) = \cup_{s=1}^{r}\{(i, j) \in E | i, j \in V_s\}$ and the subgraph induced by V_s is a clique for $s = 1, ..., r$. We say that the clique partition $E(V_1, ..., V_r)$ defines the partition $\{V_1, ..., V_r\}$. We denote a clique partition that defines the partition $\{V_1, ..., V_r\}$ as $E(V_1, ..., V_r)$.

We define a **cut** C **of the graph** $G = (V, E)$ as the set of edges in E whose removal would divide G into disjoint subgraphs $G_1(V_1, E_1), ..., G_r(V_r, E_r)$. We denote such a cut C as follows:

$$C(V_1, ..., V_r) = \{(i, j) \in E | i \in V_s, j \in V_t, s \neq t\} \tag{2.1}$$

where $\{V_1, ..., V_r\}$ is a partition of V, for $r \geq 2$.

Note that a clique partition defines a cut $C(V_1..., V_r) = E/E(V_1, ..., V_r)$.

If the graph $G = (V, E)$ has edge weights $c : E \to R$, then we denote the **value of the cut** $C(V_1, ..., V_r)$ as the following (for $r \geq 2$):

$$|C(V_1, ..., V_r)| = \sum_{(i,j) \in C(V_1,...,V_r)} c_{ij} \tag{2.2}$$

For notation purposes of this dissertation, we say that when $r = 1$, $C(V_1) = C(V)$ and $|C(V_1)| = |C(V)| = 0$.

Yeh et al. [Yeh95] define the **cluster ratio** of a cut $C(V_1, ..., V_r)$ as:

$$R_C(V_1, ..., V_r) = \frac{|C(V_1, ..., V_r)|}{\sum_{1 \leq s < t \leq r} |V_s||V_t|} \tag{2.3}$$

for $r \geq 2$.

They define the **minimum cluster ratio** among all possible cuts $C(V_1, ..., V_r)$ of $G(V, E)$ for all $r \geq 2$ as:

$$R_C^* = min_{r \geq 2}\{R_C(V_1, ..., V_r)\}. \tag{2.4}$$

A cut $C^*(V_1, ..., V_r)$ is a **minimum cluster ratio cut** of the graph $G(V, E)$ if the cluster ratio of $C^*(V_1, ..., V_r)$ is equal to the minimum cluster ratio of $G(V, E)$.

Yeh et al. [Yeh95] [SM90] also define the **weighted cluster ratio** of a cut $C(V_1, ..., V_r)$ as:

$$W_C(V_1, ..., V_r) = \frac{\sum\limits_{1 \leq s < t \leq r} w_{st}|C(V_s, V_t)|}{\sum\limits_{1 \leq s < t \leq r} w_{st}|V_s||V_t|} \tag{2.5}$$

for $r \geq 2$, where w is a symmetric distance function. That is, $w_{st} > 0$ for all $1 \leq s < t \leq r$ and $w_{st} + w_{tu} \geq w_{su}$ for all $1 \leq s < t < u \leq r$.

They define the **minimum weighted cluster ratio** among all possible cuts $C(V_1, ..., V_r)$ of $G(V, E)$ for all $r \geq 2$ as:

$$W_C^* = min_{r \geq 2}\{W_C(V_1, ..., V_r)\}. \tag{2.6}$$

A cut $C^*(V_1, ..., V_r)$ is a **minimum weighted cluster ratio cut** of the graph $G(V, E)$ if the weighted cluster ratio of $C^*(V_1, ..., V_r)$ is equal to the minimum weighted cluster ratio of $G(V, E)$.

2.1.2 Proposed Graph Theoretic Terms

We say that a partition (clustering) $\{V_1, ..., V_r\}$ of a graph $G(V, E)$ is **nested in a partition (clustering)** $\{U_1, ..., U_s\}$ if $\{V_1, ..., V_r\}$ is the result of splitting one or more clusters in $\{U_1, ..., U_s\}$ into two or more clusters.

Given a graph $G(V, E)$, a partition $\{U_1, ..., U_s\}$, and a partition $\{V_1, ..., V_r\}$ that is nested in $\{U_1, ..., U_s\}$, we define the **cut difference of** $\{V_1, ..., V_r\}$ **and** $\{U_1, ..., U_s\}$ as

$$|C(V_1, ..., V_r|U_1, ..., U_s)| = |C(V_1, ..., V_r)| - |C(U_1, ..., U_s)| \tag{2.7}$$

We propose the following terms which we can think of as an extension of the cluster ratio terms defined in 2.1.1. We discuss further in chapter 8 the rationale for why these terms can be thought of as extensions of the cluster ratio terms.

Definition Given a graph $G(V, E)$, a partition $\{U_1, ..., U_s\}$ of G, the cut $C(U_1, ..., U_s)$, a partition $\{V_1, ..., V_r\}$ of G that is nested in $\{U_1, ..., U_s\}$, and the cut $C(V_1, ..., V_r)$ we define the **β-ratio of** $C(V_1, ..., V_r)$ **given** $C(U_1, ..., U_s)$ as:

$$\beta_C(V_1, ..., V_r|U_1, ..., U_s) = \frac{|C(V_1, ..., V_r)| - |C(U_1, ..., U_s)|}{\sum_{1 \leq t < t' \leq r} |V_t|\dot{|}V_{t'}| - \sum_{1 \leq w < w' \leq s} |U_w|\dot{|}U_{w'}|}. \tag{2.8}$$

for $r > s \geq 2$.

Note that for $s = 1$, $|C(U_1, ..., U_s)| = 0$. The denominator of the ratio which is (the sum of the the multiplied cardinality of all pairs of clusters in $\{V_1, ..., V_r\}$) - (the sum of the the multiplied cardinality of all pairs of clusters in $\{U_1, ..., U_s\}$). To remain consistent with this structure, when $s = 1$ we say that $\sum_{1 \leq w < w' \leq s} |U_w|\dot{|}U_{w'}| = |U_1|\dot{|}\phi| = |U|\dot{|}\phi| = 0$.

Therefore when $s = 1$, the **β-ratio of** $C(V_1, ..., V_r)$ **given** $C(U_1, ..., U_s)$ is

$$\beta_C(V_1, ..., V_r|U_1, ..., U_s) = \frac{|C(V_1, ..., V_r)|}{\sum_{1 \leq t < t' \leq r} |V_t|\dot{|}V_t'|}. \tag{2.9}$$

Thus this shows that when $s = 1$, the β-ratio of $C(V_1, ..., V_r)$ given $C(U_1, ..., U_s)$ is the cluster ratio of $C(V_1, ..., V_r)$, and vice versa.

We define the **minimum β-ratio given** $C(U_1, ..., U_r)$ among all possible cuts $C(V_1, ..., V_r)$ where $\{V_1, ..., V_r\}$ is nested in $\{U_1, ..., U_r\}$ as:

$$\beta_C^* = min_{r \geq s}\{\beta_C(V_1, ..., V_r|U_1, ..., U_s)\}. \tag{2.10}$$

We say that a cut $C^*(V_1, ..., V_r)$ is a **minimum β-ratio cut** given $C(U_1, ..., U_s)$ of a graph $G = (V, E)$ if and only if

1. $\{V_1, ..., V_r\}$ is nested in $\{U_1, ..., U_s\}$ and
2. the β-ratio of $C^*(V_1, ..., V_r)$ given $C(U_1, ..., U_s)$ is equal to the minimum β-ratio cut given $C(U_1, ..., U_s)$.

Note it also follows that when $s = 1$, a minimum β-ratio cut given $C(U_1, ..., U_r)$ is a minimum cluster ratio cut, and vice versa.

2.2 Clustering and Consensus Clustering

As summarized in the introduction, a the goal of a clustering problem is to find structure in an unlabeled set of objects. There are many different types of clustering algorithms which yield a variety of different output structures. A partition of objects can be one such type of output from a clustering algorithm. We use the phrase a **partition-based clustering algorithm** to refer to a clustering algorithm which yields a partition of the unlabeled set of objects. Our research deals with the question of how to best combine the partitions that are yielded as a result of executing one or more partition based clustering algorithms on the *same set of objects* $O = \{o_1, ..., o_n\}$. With this in mind, we define the following notation.

We define an **input partition** (or **input clustering**) π of a set of objects $O = \{o_1, ..., o_n\}$ as a partition yielded by performing a partition-based clustering algorithm on $O = \{o_1, ..., o_n\}$. We define a **partition-based consensus clustering algorithm** as an algorithm which takes as input a set of input partitions (clusterings) $\{\pi^t\}_{t=1}^T$ (each created from the same set of unlabeled objects $V = O = \{o_1, ..., o_n\}$) and outputs a new partition (clustering) $\pi^* = \{V_1, ..., V_r\}$. We refer to π^* as a **consensus clustering** of the set of input partitions (clusterings) $\{\pi^t\}_{t=1}^T$. We use the term **consensus cluster** to refer to a disjoint set V_s in $\pi^* = \{V_1, ..., V_r\}$.

Finally we can measure the similarity between two clusterings π, π' in a variety of different ways. One such way used by the literature is the RAND Index (RAND Score) [Ran71].

Definition Let π and π' be two partitions of a set of objects $O = \{o_1, ..., o_n\}$. We define the

RAND Index (RAND Score) of π and π' as

$$RAND(\pi, \pi') = \frac{N_{11} + N_{00}}{\binom{n}{2}} \tag{2.11}$$

where N_{11} is the number of object pairs in O that are in the same cluster in both π and π' and N_{00} is the number of object pairs in O that are in different clusters in both π and π'.

2.3 Divisive Hierarchical Clustering Algorithm

In our research we introduce a consensus clustering algorithm which uses a class of clustering algorithms known as **divisive hierarchical clustering algorithms** (DHCA) [Man08]. A DHCA takes a set of objects $O = \{o_1, ..., o_n\}$ and begins with a partition (clustering) $p^1 = O$ with all n objects in the same cluster. In each iteration at least one cluster in the current partition (clustering) p^{r-1} is selected and cut into at least two clusters creating a new partition (clustering) p^r. This process is repeated until all objects are partitioned into singleton clusters. The result of a DHCA is a set of partitions (clusterings) $\{p^1, ..., p^R\}$ where p^1 is the set of all objects and where p^r is nested in p^{r-1} for $r = 2, ..., R$.

Which clusters are split and how they are split is dependent on the type of DHCA. A special type of DHCA is described as follows.

Divisive Hierarchical Clustering Algorithm with β-ratio cuts (DHCA-β)

Input: Set of objects $O = V = \{o_1, ..., o_n\}$ and $S = [s_{ij}]$ where s_{ij} represents the similarity between objects o_i and o_j. Here we assume that as s_{ij} increases, similarity between o_i and o_j increases.

Output: A set of nested partitions of V, $\{p^1, ..., p^R\}$.

1. Create a complete graph $G = (V, E)$ with edge weights $s : E \rightarrow R$.

2. Begin with a partition $p^1 = V$.

3. For each iteration $r = 2, ..., R$ create a partition p^r such that $C(p^r)$ is a minimum β-ratio cut given $C(p^{r-1})$.

We call this algorithm **DHCA-β**. There could be different ways to calculate a minimum β-ratio cut in this algorithm, thus there exist different *implementations* of the DHCA-β. For instance, one way to find a minimum β-ratio cut would be to use a brute force method in each iteration. However, our goal is to find more efficient ways at determining a minimum β-ratio cut, thus we would like to utilize a more efficient implementation of DHCA-β. Note that if for a given graph $G = (V, E)$ and similarity matrix S it is the case that there exists only one minimum β-ratio cut in each iteration, then all implementations of DHCA-β will return the same result for this $G = (V, E)$ and similarity matrix S.

2.4 Relevant Optimization Problems

Finally we introduce the following optimization problems found in the literature. They will be used in chapter 8 to a.) show how our β-ratio cut is a natural extension of the cluster-ratio cut and b.) demonstrate the rationale for our β-ratio cut.

Definition Let $G = (V, E)$ be a graph, ρ be a set of commodities, $C : E \mapsto R^+$ be a capacity function on the edges, $S : V \times \rho \mapsto R^+$ be a supply function, and $D : V \times \rho \mapsto R^+$ be a demand function. The **multicommodity flow problem (MFP)** [LR99] tries to construct flows for the commodities that satisfy the demand for each commodity at each vertex without violating the constraints imposed by the supply function and the capacity function.

Definition A **uniform multicommodity flow problem (UMFP)** [LR99] is a MFP where:

1. there is a separate commodity for each ordered pair of vertices ($\rho = V \times V$);
2. f half-units of commodity (i, j) must flow from i to j and f half-units of commodity (i, j) must flow from j to i for each $(i, j) \in E$; and
3. f is maximized

We give two linear programming (LP) formulations for the UMFP in chapter 8. Finally we restate the Median Partition Problem with Mirkin distance here.

Definition Let $O = \{o_1, ..., o_n\}$ be a set of objects and let $\{\pi^t\}_{t=1}^T$ be a set of partitions (input clusterings) of O. Let $\mathbf{\Pi}(O)$ be the set of all possible partitions (input clusterings) that can be made from O. Let $\Gamma : \mathbf{\Pi}(O) \times \mathbf{\Pi}(O) \mapsto R$ be a similarity function. Then the **Median Partition Problem (MPP)** [Lec94] returns a partition $\pi^* \in \mathbf{\Pi}(O)$ such that:

$$\pi^* = \underbrace{argmin}_{\pi \in \mathbf{\Pi}(O)} \sum_{t=1}^{T} \Gamma(\pi, \pi^t) \tag{2.12}$$

Researchers have proposed many functions for Γ, however one of the most widely used and studied functions is the symmetric difference distance function (or the Mirkin distance function) [Mir96].

Definition Let $O = \{o_1, ..., o_n\}$ be a set of objects and let π and π' be two partitions (input clusterings) of O. The **Mirkin distance** between π and π' is defined as:

$$\Gamma(\pi, \pi') = N_{01} + N_{10}, \tag{2.13}$$

where N_{01} is the number of object pairs (o_i, o_j) for which o_i and o_j are clustered together in partition π', but not in partition π and similarly N_{10} is the number of object pairs (o_i, o_j) for which o_i and o_j are clustered together in partition π, but not in partition π'.

Chapter 3

PROPOSED CONSENSUS CLUSTERING ALGORITHM AND HEURISTICS

3.1 CCA-β

In this dissertation we introduce a co-occurence type of consensus clustering algorithm which uses DHCA-β. We call this proposed consensus clustering algorithm **Divisive Hierarchical Consensus Clustering Algorithm with minimum β-ratio cut (CCA-β)**. The algorithm is as follows:

Divisive Hierarchical Consensus Clustering Algorithm with a minimum β-ratio cut (CCA-β)

Input: Let $\Pi = \{\pi^1, ..., \pi^T\}$ be a set of input clusterings of the set of objects $O = \{o_1, ..., o_n\}$.

Output: A set of nested consensus clusterings $\{p^1, ..., p^R\}$.

1. Create a $n \times n$ co-association matrix $\boldsymbol{C} = [c_{ij}]$, where

$$c_{ij} = \sum_{t=1}^{T} \delta(\pi^t(o_i), \pi^t(o_j)) \tag{3.1}$$

   $\delta(\pi^t(o_i), \pi^t(o_j)) = 1$ if o_i and o_j are assigned to the same cluster in π^t and $\delta(\pi^t(o_i), \pi^t(o_j)) = 0$ if o_i and o_j are assigned to different clusters in π^t.

2. Run DHCA-β, using $V = O = \{o_1, ..., o_n\}$ and $\boldsymbol{S} = \boldsymbol{C}$ as the similarity matrix.

There are different implementations of DHCA-β, thus there are different *implementations* of CCA-β.

Definition Let $O = \{o_1, ..., o_n\}$. We say that a set of nested consensus clusterings $\Pi = \{p^1, ..., p^R\}$ is a **CCA-β solution for** $\{\pi^1, ..., \pi^T\}$ if $\{p^1, ..., p^R\}$ could have been returned by CCA-β where $\{\pi^1, ..., \pi^T\}$ was the input.

In other words, $p^1 = O$ and $C(p^r)$ is a β-ratio cut given $C(p^{r-1})$ for $r = 2, ..., R$.

Note that if in each iteration of CCA-β given an input $\{\pi^1, ..., \pi^T\}$ there exists only one $\beta-$cut, then all implementations of CCA-β will return the same result. In other words, there exists only one CCA-β solution for $\{\pi^1, ..., \pi^T\}$.

Clearly the most computationally taxing part of CCA-β is finding the β-ratio cut in each iteration. Our goal is to find efficient ways to determine a minimum β-ratio cut in each iteration. In section 3.2 we propose two heuristics for CCA-β that use a parametric linear program model (PLP) that we introduce in chapter 4 and parametric programming algorithms which we introduce in chapter 5.

3.2 Heuristics for CCA-β

In this thesis we introduce two heuristics for CCA-β, given below. Both heuristics involve using a parametric LP formulation, P_λ, which we introduce in chapter 4 and one of two linear parametric programming algorithms (which we call OPPPA and ROPPPA) which we introduce in sections 5.2 and 5.3 respectively.

CCA-OPPPA Heuristic

Input: Let $\Pi = \{\pi^1, ..., \pi^T\}$ be a set of input clusterings of the set of objects $O = \{o_1, ..., o_n\}$.
Output: A set of vectors $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ where $\boldsymbol{x}^*(\lambda_r)$ is an optimal solution of P_{λ_r} for $r = 1, ..., R$.

1. Create a co-association matrix $C = [c_{ij}]$, where

$$c_{ij} = \sum_{t=1}^{T} \delta(\pi^t(o_i), \pi^t(o_j)) \tag{3.2}$$

   $\delta(\pi^t(o_i), \pi^t(o_j)) = 1$ if o_i and o_j are assigned to the same cluster in π^t and $\delta(\pi^t(o_i), \pi^t(o_j)) = 0$ if o_i and o_j are assigned to different clusters in π^t.

2. Use $C = [c_{ij}]$ in the objective function of the proposed parametric LP P_λ (chapter 4) and

apply the parametric linear programming algorithm OPPPA (chapter 5.2) to P_λ over the range $[0, \binom{n}{2}]$.

CCA-ROPPPA Heuristic

Input: Let $\Pi = \{\pi^1, ..., \pi^T\}$ be a set of input clusterings of the set of objects $O = \{o_1, ..., o_n\}$.

Output: A set of vectors $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ where $\boldsymbol{x}^*(\lambda_r)$ is an optimal solution of P_{λ_r} for $r = 1, ..., R$.

1. Create a co-association matrix $C = [c_{ij}]$, where

$$c_{ij} = \sum_{t=1}^{T} \delta(\pi^t(o_i), \pi^t(o_j)) \tag{3.3}$$

   $\delta(\pi^t(o_i), \pi^t(o_j)) = 1$ if o_i and o_j are assigned to the same cluster in π^t and $\delta(\pi^t(o_i), \pi^t(o_j)) = 0$ if o_i and o_j are assigned to different clusters in π^t.

2. Use $C = [c_{ij}]$ in the objective function of the proposed parametric LP P_λ (chapter 4) and apply the parametric linear programming algorithm ROPPPA (chapter 5.3) to P_λ over the range $[0, \binom{n}{2}]$.

One thing to note is that the output of CCA-OPPPA and CCA-ROPPPA is a set of *vectors* $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$, not a set of nested consensus clusterings $\{p^1, ..., p^R\}$ like the output of CCA-β. We will discuss in chapter 4 that when the set of vectors $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ satisfy certain conditions, they *define* a set of nested consensus clusterings $\{p^1, ..., p^R\}$. However, when they do not meet these conditions they do not define a set of nested consensus clusterings. Thus this is the reason why we call CCA-OPPPA and CCA-ROPPPA *heuristics* for CCA-β in that they will not always return CCA-β solutions. We will discuss this more in chapters 4, 6, and 7.

Chapter 4

PARAMETRIC LP MODEL

In this chapter we present our proposed parametric LP formulation P_λ that is used in our CCA-OPPPA and CCA-ROPPPA heuristics. In section 4.1 we present the parametric LP formulation. In section 4.3 we discuss how to interpret P_λ.

Because the feasible region of the GWLP (discussed in 1.4) is closely related to the feasible region of P_λ, many our results in section 4.3 rely upon results proven by Grotschel and Wakabayashi [GW89] about the GWLP. Thus in section 4.2 we rewrite the GWLP, discuss how it is related to P_λ, and give these results.

4.1 Parametric LP Model P_λ

The proposed parametric LP formulation that is used in our CCA-OPPPA and CCA-ROPPPA heuristics is given below. Because the formulation is very similar to GWLP, we call it the **Modified Grotschel Wakabayashi Linear Program (P_λ)**. We will denote this parametric LP formulation as P_λ for the remainder of this dissertation. In addition, we will denote the feasible region of P_λ as (P_λ). We note that this parametric LP formulation came as a result of a research collaboration with Yoshitsugu Yamamoto at the University of Tsukuba and Amy Langville at the College of Charleston.

Modified Grotschel Wakabayashi Linear Program P_λ

Model

$$
\begin{array}{lll}
\max & \sum_{1\leq i<j\leq n} c_{ij}x_{ij} & \\
\text{st} & x_{ij}+x_{jk}-x_{ik}\leq 1 & \text{for } 1\leq i<j<k\leq n \\
& x_{ij}-x_{jk}+x_{ik}\leq 1 & \text{for } 1\leq i<j<k\leq n \\
& -x_{ij}+x_{jk}+x_{ik}\leq 1 & \text{for } 1\leq i<j<k\leq n \\
& \sum_{1\leq i<j\leq n} x_{ij}=M(\lambda) & \\
& x_{ij}\geq 0 & \text{for } 1\leq i<j\leq n
\end{array}
$$

Variables $x_{ij} = \begin{cases} 1 & \text{if } o_i \text{ and } o_j \text{ are clustered together in the consensus clustering} \\ 0 & \text{otherwise} \end{cases}$

Objective Function Coefficients

$$c_{ij} = \sum_{t=1}^{T} \delta(\pi^t(o_i), \pi^t(o_j)), \tag{4.1}$$

where $\Pi = \{\pi^1, ..., \pi^T\}$ is a set of consensus clusterings, $\delta(\pi^t(o_i), \pi^t(o_j)) = 1$ if o_i and o_j are assigned to the same cluster in π^t and $\delta(\pi^t(o_i), \pi^t(o_j)) = 0$ if o_i and o_j are assigned to different clusters in π^t.

Parametric RHS $M(\lambda) := \binom{n}{2} - \lambda$

Furthermore we will use the notation $\boldsymbol{x}(\lambda)$ to denote a feasible solution of (P_λ).

4.2 Related GWLP Model P

Our proposed P_λ is related to the GWLP. We give the GWLP again in section 4.2.1. In section 4.2.2 we discuss how the GWLP is related to our P_λ. In section 4.2.3 we give several results that Grotschel and Wakabayashi prove about the GWLP which will help us in proving results about P_λ in section 4.3.

4.2.1 GWLP Model P

The GWLP (also in section 1.4) is shown again below.

Grotschel Wakabayashi Linear Program P (GWLP)

Model

$$\begin{array}{lll} \min & \sum_{1\le i<j\le n} w_{ij}x_{ij} & \\ \text{st} & x_{ij}+x_{jk}-x_{ik}\le 1 & \text{for } 1\le i<j<k\le n \\ & x_{ij}-x_{jk}+x_{ik}\le 1 & \text{for } 1\le i<j<k\le n \\ & -x_{ij}+x_{jk}+x_{ik}\le 1 & \text{for } 1\le i<j<k\le n \\ & x_{ij}\ge 0 & \text{for } 1\le i<j\le n \end{array}$$

Variables $x_{ij} = \begin{cases} 1 & \text{if } o_i \text{ and } o_j \text{ are clustered together in the consensus clustering} \\ 0 & \text{otherwise} \end{cases}$

Objective Function Coefficients

$$w_{ij} = \sum_{t=1}^{T} \epsilon(\pi^t(o_i), \pi^t(o_j)), \tag{4.2}$$

where $\Pi = \{\pi^1, ..., \pi^T\}$ is a set of consensus clusterings, $\epsilon(\pi^t(o_i), \pi^t(o_j)) = -1$ if o_i and o_j are assigned to the same cluster in π^t and $\epsilon(\pi^t(o_i), \pi^t(o_j)) = 1$ if o_i and o_j are assigned to different clusters in π^t.

For the remainder of this thesis we denote the GWLP as P. We denote the feasible region of P as (P).

4.2.2 How (P) Relates to (P_λ)

The two formulations are similar in that $(P_\lambda) = (P) \cap \{x | \sum_{i<j} x_{ij} = M(\lambda)\}$.

4.2.3 Properties and Interpreting Solutions of P

The following are properties and lemmas about (P) proven by Grotschel and Wakabayashi [GW89].

Lemma 4.2.1 *There is a one to one correspondence between the set of all clique partitions of $K_n = (V, E)$ and the set of all binary solutions of (P).*

We can use this lemma to interpret binary solutions of (P).

Interpreting Binary Solutions of (P)

- Specifically, a binary $\boldsymbol{x} \in (P)$ defines a clique partition $E(V_1, ..., V_r)$ of $K_n = (V, E)$ (or equivalently a cut $C(V_1, ..., V_r)$ of $K_n = (V, E)$ as follows:

$$C(V_1, ..., V_r) = \{(i, j) \in E | x_{ij} = 0\} \tag{4.3}$$

$$E(V_1, ..., V_r) = \{(i, j) \in E | x_{ij} = 1\} \tag{4.4}$$

- Specifically, a binary $\boldsymbol{x} \in (P)$ defines a partition (consensus clustering) $\{V_1, ..., V_r\}$ of $K_n = (V, E)$ as follows:
  - By removing the edges in $C(V_1, ..., V_r)$ from E, we are left with a set of disjoint cliques $H_1(V_1, E_1), ..., H_r(V_r, E_r)$ of K_n.
  - The partition (consensus clustering) is the set $\{V_1, ..., V_r\}$.
- Therefore, we say that a $\boldsymbol{x} \in (P)$ **defines a (partition) consensus clustering** p if and only if $\boldsymbol{x}$ is binary.

4.3 Properties and Interpreting Solutions of P_λ

Using the properties of P we can make a similar statement to that of lemma 4.2.1

Lemma 4.3.1 *There is a one to one correspondence between the set of all clique partitions of $K_n = (V, E)$ and the set of all binary solutions of (P_λ) for $\lambda \in [0, \binom{n}{2}]$.*

Proof Suppose there exists a clique partition $E(V_1, ..., V_r)$ (or cut $C(V_1, ..., V_r)$) of $K_n = (V, E)$. By lemma 4.2.1, there exists a binary solution $\boldsymbol{x} \in (P)$ that corresponds

to $C(V_1, ..., V_r)$. Specifically $x_{ij} = 0$ if $(i,j) \in C(V_1, ..., V_r)$ and $x_{ij} = 1$ otherwise. Because K_n is a complete graph, then $|E| = \binom{n}{2}$ and
$|\{(i,j) \in E | (i,j) \notin C(V_1, ..., V_r)\}| = \binom{n}{2} - |C(V_1, ..., V_r)|$

If we let $\lambda = |C(V_1, ..., V_r)|$, then it follows that

$$\begin{aligned}
\textstyle\sum_{1 \le i < j \le n} x_{ij} &= \textstyle\sum_{(i,j) \in C(V_1,...,V_r)} x_{ij} + \sum_{(i,j) \notin C(V_1,...,V_r)} x_{ij} \\
&= \textstyle\sum_{(i,j) \notin C(V_1,...,V_r)} x_{ij} \\
&= \binom{n}{2} - |C(V_1, ..., V_r)| \\
&= \binom{n}{2} - \lambda
\end{aligned}$$

Thus there exists a $\lambda \in [0, \binom{n}{2}]$ such that $\boldsymbol{x} \in (P_\lambda)$. Thus $C(V_1, ..., V_r)$ corresponds to a binary solution in (P_λ).

Now suppose there exists a binary $\boldsymbol{x} \in (P_\lambda)$. Because $(P_\lambda) = (P) \cap \{x | \sum_{i<j} x_{ij} = M(\lambda)\}$, then it must be the case that $\boldsymbol{x} \in (P)$. Thus by lemma 4.2.1, there exists a clique partition that corresponds to $\boldsymbol{x}$.

Because of lemma 4.3.1 we can interpret binary solutions of P_λ the same way as we did for binary solutions of P.

Interpreting Binary Solutions of (P_λ)

- Specifically, a binary $\boldsymbol{x}(\lambda) \in (P_\lambda)$ defines a clique partition $E(V_1, ..., V_r)$ of $K_n = (V, E)$ (or equivalently a cut $C(V_1, ..., V_r)$ of $K_n = (V, E)$) as follows:

$$C(V_1, ..., V_r) = \{(i,j) \in E | x(\lambda)_{ij} = 0\} \tag{4.5}$$

$$E(V_1, ..., V_r) = \{(i,j) \in E | x(\lambda)_{ij} = 1\} \tag{4.6}$$

- Specifically, a binary $\boldsymbol{x}(\lambda) \in (P_\lambda)$ defines a partition (consensus clustering) $\{V_1, ..., V_r\}$ of $K_n = (V, E)$ as follows:
  - By removing the edges in $C(V_1, ..., V_r)$ from E, we are left with a set of disjoint cliques $H_1(V_1, E_1), ..., H_r(V_r, E_r)$ of K_n.
  - The partition (consensus clustering) is the set $\{V_1, ..., V_r\}$.
- Therefore, we say that a $\boldsymbol{x}(\lambda) \in (P_\lambda)$ **defines a (partition) consensus clustering** p if and only if $\boldsymbol{x}(\lambda)$ is binary.

- Furthermore if $\boldsymbol{x}(\lambda_1) \in (P_{\lambda_1}), ..., \boldsymbol{x}(\lambda_R) \in (P_{\lambda_R})$, then we can also say that a set $\{\boldsymbol{x}(\lambda_1), ..., \boldsymbol{x}(\lambda_R)\}$ **defines a set of (partitions) consensus clusterings** $\{p^1, ..., p^R\}$ if and only if each vector in the set is binary.

- Next, (using the definition in 3.1) if $\boldsymbol{x}(\lambda_1) \in (P_{\lambda_1}), ..., \boldsymbol{x}(\lambda_R) \in (P_{\lambda_R})$, then say that a set $\{\boldsymbol{x}(\lambda_1), ..., \boldsymbol{x}(\lambda_R)\}$ **defines a CCA-β solution for** $\{\pi^1, ..., \pi^T\}$ if and only if

  - $\{\boldsymbol{x}(\lambda_1), ..., \boldsymbol{x}(\lambda_R)\}$ defines a set of consensus clusterings $\{p^1, ..., p^R\}$ and
  - $\{p^1, ..., p^R\}$ is a CCA-β solution for $\{\pi^1, ..., \pi^T\}$

- Finally, if CCA-OPPPA (CCA-ROPPPA) is applied to an input clustering set $\{\pi^1, ..., \pi^T\}$ and returns the set of vectors $\{\boldsymbol{x}(\lambda_1), ..., \boldsymbol{x}(\lambda_R)\}$, then we say that CCA-OPPPA (CCA-ROPPPA) **returned a CCA-β solution for** $\{\pi^1, ..., \pi^T\}$ if and only if $\{\boldsymbol{x}(\lambda_1), ..., \boldsymbol{x}(\lambda_R)\}$ defined a CCA-β solution for $\{\pi^1, ..., \pi^T\}$.

Note however, that not all optimal solutions of P_λ are binary. Because of this CCA-OPPPA and CCA-ROPPPA is not guaranteed to return a set of $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ that define a CCA-β solution. Thus this is why we call CCA-OPPPA and CCA-ROPPPA a heuristic of CCA-β.

Chapter 5

Parametric Programming Algorithms

In this chapter, we present and discuss the parametric linear programming algorithms that are used in our CCA-OPPPA and CCA-ROPPPA heuristics. In general, the goal of a parametric linear programming algorithm is to determine how the optimal solution and the optimal objective function value of a given parametric LP $LP(\theta)$ change as θ is perturbed over some interval where $LP(\theta)$ is feasible. A typical result that can be returned from a parametric programming algorithm is a *continuous* set of solutions $\{\boldsymbol{x}^*(\theta)\}$ that are optimal in $LP(\theta)$.

For the purpose of this thesis, the goal of applying a parametric linear programming algorithm specifically to P_λ over $\lambda \in [0, \binom{n}{2}]$ is to return a *discrete* set of vector solutions $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ (that are optimal in $P_{\lambda_1}, ..., P_{\lambda_R}$ respectively) that defines a CCA-β solution. Thus the two parametric programming algorithms we present in this chapter will only return a finite set of solutions. Sections 5.2 and 5.3 will discuss in further detail which solutions are chosen in the algorithms.

In section 5.2 we discuss the parametric programming algorithm that we use in our CCA-OPPPA heuristic. This parametric programming algorithm is found in the literature [Ber96] and we call it OPPPA in this dissertation. In section 5.3 we propose the parametric programming algorithm that we use in our CCA-ROPPPA heuristic. This parametric programming algorithm is a modification of OPPPA and we call it ROPPPA.

In order to introduce these parametric programming algorithms, we must first introduce some notation and lemmas regarding P_λ in section 5.1. Finally, although CCA-OPPPA and CCA-ROPPPA are not guaranteed to return CCA-β solutions, in section 5.4 we are able to prove some properties about the set of vector returned CCA-OPPPA and CCA-ROPPPA that are *necessary conditions* for a set of vectors to define a CCA-β solution.

5.1 Parametric Programming Algorithm Notation and Properties

5.1.1 Parametric Programming Algorithm Notation

First, in section 5.1.1 we introduce some notation that we will use in explaining the parametric programming algorithms that we introduce in sections 5.2 and 5.3. In order to use these parametric programming algorithms we must have P_λ satisfy some properties. In section 5.1.2 we show and prove these properties.

5.1.1.1 Primal Formulations

- **Parametric Primal Problem:** We define $P_\lambda = \{max\, \boldsymbol{c}'\boldsymbol{x} | \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{e}'\boldsymbol{x} = \binom{n}{2} - \lambda, \boldsymbol{x} \geq \boldsymbol{0}\}$ as the primal of P_λ (matrix notation) with parameter λ for all $\lambda \in [0, \binom{n}{2})]$. In this formulation we have $\boldsymbol{c}' = [c_{12}, c_{13}, ..., c_{1n}, c_{23}, c_{24}, ..., c_{n-1,n}]$, $\boldsymbol{x}' = [x_{12}, x_{13}, ..., x_{1n}, x_{23}, x_{24}, ..., x_{n-1,n}]$, $\boldsymbol{e}' = [1, 1, ..., 1]$, and $\boldsymbol{b} = [1, 1, ..., 1]$. Written another way, we write P_λ as:

$$\begin{array}{lll} \max & \sum_{i<j} c_{ij} x_{ij} & \\ \text{st} & \boldsymbol{a}_{ijk}^{(h)\prime} \boldsymbol{x} \leq b_{ijk}^{(h)} & \forall 1 \leq i < j < k \leq n, h = 1,2,3 \\ & \sum_{i<j} x_{ij} = \binom{n}{2} - \lambda & \\ & x_{ij} \geq 0 & \forall 1 \leq i < j \leq n \end{array}$$

  where $b_{ijk}^{(h)} = 1$ for all $1 \leq i < j < k \leq n$ and $h = 1, 2, 3$; and $\boldsymbol{a}_{ijk}^{(h)\prime}$ is a row vector of length $\binom{n}{2}$ where

  - $\boldsymbol{a}_{ijk}^{(1)\prime} \boldsymbol{x} \leq b_{ijk}^{(1)}$ represents the constraint $x_{ij} + x_{ik} - x_{jk} \leq 1$
  - $\boldsymbol{a}_{ijk}^{(2)\prime} \boldsymbol{x} \leq b_{ijk}^{(2)}$ represents the constraint $x_{ij} - x_{ik} + x_{jk} \leq 1$
  - $\boldsymbol{a}_{ijk}^{(3)\prime} \boldsymbol{x} \leq b_{ijk}^{(3)}$ represents the constraint $-x_{ij} + x_{ik} + x_{jk} \leq 1$.

- **Parametric Primal Problem (Standard Form):** We define $\bar{P}_\lambda = \{max\, \boldsymbol{c}'\boldsymbol{x} | \boldsymbol{A}\boldsymbol{x} + \boldsymbol{z} = \boldsymbol{b}, \boldsymbol{e}'\boldsymbol{x} = \binom{n}{2} - \lambda, \boldsymbol{x}, \boldsymbol{z} \geq \boldsymbol{0}\} = \{max\, \bar{\boldsymbol{c}}'\bar{\boldsymbol{x}} | \bar{\boldsymbol{A}}\bar{\boldsymbol{x}} = \boldsymbol{b}(\lambda), \bar{\boldsymbol{x}} \geq \boldsymbol{0}\}$ as the primal of P_λ in standard form (matrix notation) with parameter λ for all $\lambda \in [0, \binom{n}{2})]$. In this formulation we have

$\bar{\boldsymbol{c}}' = [\boldsymbol{c}'|0, ..., 0]$, $\bar{\boldsymbol{x}}' = [\boldsymbol{x}'|\boldsymbol{z}']$, $\boldsymbol{z}' = [z_{123}^{(1)}, z_{123}^{(2)}, z_{123}^{(3)}, ..., z_{n-2,n-1,n}^{(1)}, z_{n-2,n-1,n}^{(2)}, z_{n-2,n-1,n}^{(3)}]$, and

$$\bar{\boldsymbol{A}} = \left[\begin{array}{c|c} \boldsymbol{A} & \boldsymbol{I} \\ \hline \boldsymbol{e}' & \boldsymbol{0}' \end{array}\right]$$

Also in this formulation we define the right-hand-side (RHS) vector of $\bar{P}_\lambda$ as the function $\boldsymbol{b}(\lambda) = \boldsymbol{b} - \lambda\boldsymbol{\Delta b}$ where $\boldsymbol{b} = [\boldsymbol{b}'|\binom{n}{2}]'$ and $\boldsymbol{\Delta b} = [0, 0, ..., 0, 1]'$.

Written another way, we write $\bar{P}_\lambda$ as:

$$\begin{array}{lll} \max & \sum_{i<j} c_{ij}x_{ij} & \\ \text{st} & \boldsymbol{a}_{ijk}^{(h)\prime}\boldsymbol{x} + z_{ijk}^{(h)} = b_{ijk}^{(h)} & \forall 1 \leq i < j < k \leq n, h = 1, 2, 3 \\ & \sum_{i<j} x_{ij} = \binom{n}{2} - \lambda & \\ & x_{ij} \geq 0 & \forall 1 \leq i < j \leq n \\ & z_{ijk}^{(h)} \geq 0 & \forall 1 \leq i < j < k \leq n, h = 1, 2, 3 \end{array}$$

where $\boldsymbol{a}_{ijk}^{(h)\prime}$ is a row vector of length $\binom{n}{2}$ where

- $\boldsymbol{a}_{ijk}^{(1)\prime}\boldsymbol{x} + z_{ijk}^{(1)} = b_{ijk}^{(1)}$ represents the standard form constraint $x_{ij} + x_{ik} - x_{jk} + z_{ijk}^{(1)} = 1$,
- $\boldsymbol{a}_{ijk}^{(2)\prime}\boldsymbol{x} + z_{ijk}^{(2)} = b_{ijk}^{(2)}$ represents the standard form constraint $x_{ij} - x_{ik} + x_{jk} + z_{ijk}^{(2)} = 1$, and
- $\boldsymbol{a}_{ijk}^{(3)\prime}\boldsymbol{x} + z_{ijk}^{(3)} = b_{ijk}^{(3)}$ represents the constraint $-x_{ij} + x_{ik} + x_{jk} + z_{ijk}^{(3)} = 1$.

- **Parametric Primal Feasible Region:** We define (P_λ) and $(\bar{P}_\lambda)$ as the feasible regions of P_λ and $\bar{P}_\lambda$ respectively for all $\lambda \in [0, \binom{n}{2}]$.

- **Parametric Primal Solutions:** We use the notation $\boldsymbol{x}(\lambda)$ or $\bar{\boldsymbol{x}}(\lambda) := [\boldsymbol{x}(\lambda)'|\boldsymbol{z}(\lambda)']'$ to represent a solution of P_λ and $\bar{P}_\lambda$ respectively.

- **Parametric Primal Optimal Solutions:** We define P_λ^* and $\bar{P}_\lambda^*$ as the set of optimal solutions of P_λ and $\bar{P}_\lambda$ for $\lambda \in [0, \binom{n}{2}]$.

5.1.1.2 Dual Formulations

- **Parametric Dual Problem:** We define $\bar{D}_\lambda = \{min\, \boldsymbol{y}'\boldsymbol{b}(\lambda)|\boldsymbol{y}'\bar{\boldsymbol{A}} \geq \bar{\boldsymbol{c}}\}$ as the dual of $\bar{P}_\lambda$ for all $\lambda \in [0, \binom{n}{2}]$. Written another way, we write $\bar{D}_\lambda$ as:

$$
\begin{array}{lll}
\min & (\binom{n}{2} - \lambda)y_m + \sum_{h=1}^{3} \sum_{i<j<k} b_{ijk}^{(h)} y_{ijk}^{(h)} & \\
\text{st} & \boldsymbol{y'}\bar{\boldsymbol{A}}_{ij} \geq c_{ij} & \forall 1 \leq i < j \leq n \\
 & y_{ijk}^{(h)} \geq 0 & \forall 1 \leq i < j < k \leq n, h = 1,2,3
\end{array}
$$

Here we have $\boldsymbol{A}_{ij}$ as the column vector that corresponds to variable x_{ij} in $\boldsymbol{A}$. Also we define $\boldsymbol{y'} = [y_{123}^{(1)}, y_{123}^{(1)}, ...y_{n-2,n-1,n}^{(3)}, y_m]$ as a row vector of length $3\binom{n}{3} + 1$ where:

- $y_{ijk}^{(1)}$ is the dual variable that corresponds to the standard form primal constraint $\boldsymbol{a}_{ijk}^{(1)\prime}\boldsymbol{x} + z_{ijk}^{(1)} = b_{ijk}^{(1)}$ (or $x_{ij} + x_{ik} - x_{jk} + z_{ijk}^{(1)} = 1$)
- $y_{ijk}^{(2)}$ is the dual variable that corresponds to the standard form primal constraint $\boldsymbol{a}_{ijk}^{(2)\prime}\boldsymbol{x} + z_{ijk}^{(2)} = b_{ijk}^{(2)}$ (or $x_{ij} - x_{ik} + x_{jk} + z_{ijk}^{(2)} = 2$)
- $y_{ijk}^{(3)}$ is the dual variable that corresponds to the standard form primal constraint $\boldsymbol{a}_{ijk}^{(3)\prime}\boldsymbol{x} + z_{ijk}^{(3)} = b_{ijk}^{(3)}$ (or $-x_{ij} + x_{ik} + x_{jk} + z_{ijk}^{(3)} = 1$)
- y_m is the dual variable that corresponds to the standard form primal constraint $\sum_{i<j} x_{ij} = \binom{n}{2} - \lambda$

- **Parametric Dual Problem (Standard Form):** We define $\bar{\bar{D}}_\lambda = \{min\boldsymbol{y'}\boldsymbol{b}(\lambda) | \bar{\boldsymbol{A}}'\boldsymbol{y} - \boldsymbol{s} = \bar{\boldsymbol{c}},\ \boldsymbol{y}, \boldsymbol{s} \geq \boldsymbol{0}\}$ as $\bar{D}$ in standard form. Written another way, we write $\bar{\bar{D}}_\lambda$ as:

$$
\begin{array}{lll}
\min & (\binom{n}{2} - \lambda)y_m + \sum_{h=1}^{3} \sum_{i<j<k} b_{ijk}^{(h)} y_{ijk}^{(h)} & \\
\text{st} & \boldsymbol{y'}\bar{\boldsymbol{A}}_{ij} - s_{ij} = c_{ij} & \forall 1 \leq i < j \leq n \\
 & y_{ijk}^{(h)} - s_{ijk}^{(h)} = 0 & \forall 1 \leq i < j < k \leq n, h = 1,2,3 \\
 & s_{ijk}^{(h)} \geq 0 & \forall 1 \leq i < j < k \leq n, h = 1,2,3 \\
 & s_{ij} \geq 0 & \forall 1 \leq i < j \leq n \\
 & y_m URS &
\end{array}
$$

Thus we have introduced new slack variables where s_{ij} is the slack variable for the constraint $\boldsymbol{y'}\boldsymbol{A}_{ij} \geq c_{ij}$ and $s_{ijk}^{(h)}$ is the slack variable for the constraint $y_{ijk}^{(h)} \geq 0$.

- **Parametric Dual Feasible Region:** We define $(\bar{\bar{D}})$ and $(\bar{D})$ as the feasible regions of $\bar{\bar{D}}_\lambda$ and $\bar{D}_\lambda$ respectively for all $\lambda \in [0, \binom{n}{2}]$. Note that since the λ is in the objective function of the dual problem, the dual feasible region is the same for all $\lambda \in [0, \binom{n}{2}]$. Thus we do not use λ when denoting the *feasible regions* of the dual formulations.

- **Parametric Dual Solutions:** We use the notation $\boldsymbol{y}$ or $\boldsymbol{y}(\lambda)$ to represent a solution of $\bar{D}_\lambda$. We use the notation $(\boldsymbol{y}, \boldsymbol{s})$ or $(\boldsymbol{y}(\lambda), \boldsymbol{s}(\lambda))$ to represent a solution of $\bar{\bar{D}}_\lambda$

- **Parametric Dual Optimal Solutions:** We define $\bar{D}^*_\lambda$ and $\bar{\bar{D}}^*_\lambda$ as the set of optimal solutions of $\bar{D}_\lambda$ and $\bar{\bar{D}}_\lambda$ respectively for all $\lambda \in [0, \binom{n}{2}]$.

5.1.1.3 Other Notation

- **Parametric Optimal Objective Function Value:**

  Definition We define the function $f(\lambda)$ to be the optimal objective function value of P_λ (and also $\bar{P}_\lambda$, $\bar{D}_\lambda$, and $\bar{\bar{D}}_\lambda$) over the domain $[0, \binom{n}{2}]$. Thus we can also write

$$\begin{aligned} f(\lambda) &= \boldsymbol{c}'\boldsymbol{x}^*(\lambda) && (\boldsymbol{x} \in P^*_\lambda) \\ &= \bar{\boldsymbol{c}}'\bar{\boldsymbol{x}}^*(\lambda) && (\bar{\boldsymbol{x}} \in \bar{P}^*_\lambda) \\ &= [\boldsymbol{y}^*]'\boldsymbol{b}(\lambda) && (\boldsymbol{y}^* \in \bar{D}^*_\lambda) \\ &= [\boldsymbol{y}^*]'\boldsymbol{b}(\lambda) && ((\boldsymbol{y}^*, \boldsymbol{s}^*) \in \bar{\bar{D}}^*_\lambda) \end{aligned}$$

- **GWLP Feasible Region** We define (P) as the feasible region for the GWLP.

5.1.2 Parametric Programming Algorithm Properties

In order to use the parametric programming algorithm presented in section 5.2, we must have P_λ satisfy the following properties shown and proved in the lemmas below.

Lemma 5.1.1 *$(\bar{P}_\lambda)$ is bounded and nonempty for all $\lambda \in [0, \binom{n}{2}]$.*

Proof Let $[\boldsymbol{x}'|\boldsymbol{z}'] \in (\bar{P}_\lambda)$.

For all x_{ij} where $[\boldsymbol{x}'|\boldsymbol{z}'] \in (\bar{P}_\lambda)$, a linear combination of the constraints $x_{ij} + x_{ik} - x_{jk} + z^{(1)}_{ijk} = 1$ and $x_{ij} - x_{ik} + x_{jk} + z^{(2)}_{ijk} = 1$ gives us $2x_{ij} + z^{(1)}_{ijk} + z^{(2)}_{ijk} = 2$ or $x_{ij} = 1 - \frac{z^{(1)}_{ijk} + z^{(2)}_{ijk}}{2}$. Because $z^{(h)}_{ijk} \geq 0$ for all $1 \leq i < j < k \leq n$ and $h = 1, 2, 3$, $x_{ij} \leq 1$. Thus $\mathbf{0} \leq \boldsymbol{x} \leq \mathbf{1}$.

Furthermore because $\mathbf{0} \leq \boldsymbol{x} \leq \mathbf{1}$ it follows that

$$\begin{array}{lll} -1 & \leq x_{ij} + x_{ik} - x_{jk} & \leq 1 \\ 2 & \geq 1 - (x_{ij} + x_{ik} - x_{jk}) & \geq 0 \\ 2 & \geq z^{(1)}_{ijk} & \geq 0 \end{array}$$

Similar arguments show that $2 \geq z^{(2)}_{ijk} \geq 0$ and $2 \geq z^{(3)}_{ijk} \geq 0$. Thus $\mathbf{0} \leq \boldsymbol{z} \leq \mathbf{2}$. Since $\mathbf{0} \leq \boldsymbol{x} \leq \mathbf{1}$ and $\mathbf{0} \leq \boldsymbol{z} \leq \mathbf{2}$ for $[\boldsymbol{x}'|\boldsymbol{z}'] \in (\bar{P}_\lambda)$, $(\bar{P}_\lambda)$ is bounded for all $\lambda \in [0, \binom{n}{2}]$.

Furthermore the solution where $x_{ij} = \frac{\binom{n}{2} - \lambda}{\binom{n}{2}}$ for all $1 \leq i < j \leq n$ and $z^{(h)}_{ijk} = 1 - \frac{\binom{n}{2} - \lambda}{\binom{n}{2}}$ for all $i < j < k$ and $h = 1, 2, 3$ is feasible in $(\bar{P}_\lambda)$ for all $\lambda \in [0, \binom{n}{2}]$. Thus $(\bar{P}_\lambda)$ is nonempty for all $\lambda \in [0, \binom{n}{2}]$.

Lemma 5.1.2 *$(\bar{D})$ and $(\bar{\bar{D}})$ are nonempty.*

Proof Since $(\bar{P}_\lambda)$ is bounded and nonempty for all $\lambda \in [0, \binom{n}{2}]$ (by lemma 5.1.1), then $\bar{P}_\lambda$ will have an optimal solution for all $\lambda \in [0, \binom{n}{2}]$. Therefore the dual of $\bar{P}_\lambda$, $\bar{D}_\lambda$ will have an optimal solution. Thus $(\bar{D})$ is nonempty. Thus $(\bar{\bar{D}})$ is also nonempty.

We also discuss some properties of the function $f(\lambda)$ for $\lambda \in [0, \binom{n}{2}]$. The following lemmas about the nature of $f(\lambda)$ follows from Tsiktsiklis [BT97] and Berkelaar et al. [Ber96].

Lemma 5.1.3 *The function $f : [0, \binom{n}{2}] \to R$ defined by P_λ (or equivalently $\bar{P}_\lambda$) is concave, continuous, piecewise linear.*

Proof First we define $f(\lambda)$ via the dual $\bar{D}_\lambda$. Since $f(\lambda)$ is the optimal objective function value of $\bar{P}_\lambda$ and thus $\bar{D}_\lambda$ we can say $f(\lambda) = [\boldsymbol{y}^*]'\boldsymbol{b}(\lambda)$ where y^* is an optimal dual solution to $\bar{D}_\lambda$. Since $\bar{D}_\lambda$ is an LP, the optimal objective function value of $\bar{D}_\lambda$ is reached by at least one extreme point of $(\bar{D})$ for all $\lambda \in [0, \binom{n}{2}]$. If we define $\boldsymbol{y}^1, ..., \boldsymbol{y}^L$ as the extreme points of $(\bar{D})$, then we can say that

$$\begin{aligned} f(\lambda) &= min_{l=1,...,L}(\boldsymbol{y}^l)'\boldsymbol{b}(\lambda) \\ &= min_{l=1,...,L}((\boldsymbol{y}^l)'\boldsymbol{b} + \lambda(\boldsymbol{y}^l)'\Delta\boldsymbol{b}) \end{aligned}.$$

This shows that f is equal to the minimum of a finite collection of linear functions which implies that f is a piecewise linear concave function. Because f is concave, this it must be continuous on $(0, \binom{n}{2})$.

We only need to prove that f is right-continuous and left-continuous on the domain end points $\lambda = 0$ and $\lambda = \binom{n}{2}$ respectively. We will show right-continuity of 0, or $lim_{\lambda\to 0^+} f(\lambda) = f(0)$ (an analogous argument can be shown for $\binom{n}{2}$.)

$$\begin{aligned} lim_{\lambda\to 0^+} f(\lambda) &= lim_{\lambda\to 0^+}\boldsymbol{c}'\bar{\boldsymbol{x}}^*(\lambda) \qquad \text{where } \bar{\boldsymbol{x}}^*(\lambda) \in \bar{P}^*_\lambda \text{ for } \lambda \to 0^+ \\ &= \boldsymbol{c}'(lim_{\lambda\to 0^+}\bar{\boldsymbol{x}}^*(\lambda)) \\ &= f(0) \end{aligned}$$

The last two lines in the equation follow from the following reasoning. Since $(\bar{D})$ and $(\bar{P})$ are closed, $lim_{\lambda\to 0^+}\bar{\boldsymbol{x}}^*(\lambda)$ and $lim_{\lambda\to 0^+}\boldsymbol{y}^*(\lambda)$ are feasible solutions in $(\bar{P}_0)$ and $(\bar{D})$ respectively. Since $\boldsymbol{x}^*(\lambda)$ and $\boldsymbol{y}^*(\lambda)$ are optimal in $\bar{P}_\lambda$ and $\bar{D}_\lambda$ as $\lambda \to 0^+$, they satisfy the complementary slackness equations. It follows that $lim_{\lambda\to 0^+}\bar{\boldsymbol{x}}^*(\lambda)$ and $lim_{\lambda\to 0^+}\boldsymbol{y}^*(\lambda)$ also satisfy the complementary slackness equations. Thus by complementary slackness, $lim_{\lambda\to 0^+}\bar{\boldsymbol{x}}^*(\lambda)$ and

$lim_{\lambda \to 0^+} \boldsymbol{y}^*(\lambda)$ are optimal and feasible in $\bar{P}_0$ and $\bar{D}_0$ respectively. Thus $\boldsymbol{c}'(lim_{\lambda \to 0^+} \bar{\boldsymbol{x}}^*(\lambda)) = f(0)$. Hence f is continuous.

Definition We call intervals of λ for which $f(\lambda)$ is linear the **linearity intervals** of $f(\lambda)$. We call a point λ between two linearity intervals a **transition point** of $f(\lambda)$.

5.2 Optimal Partition Parametric Programming Algorithm

The first parametric programming algorithm we propose to use on $\bar{P}_\lambda$ on the interval $[0, \binom{n}{2}]$ is one first proposed by Berkelaar et al. [Ber96]. It iteratively determines the linearity intervals $\{[\lambda_{r-1}, \lambda_r]\}_{r=2}^{R}$ (and subsequently the transition points $\lambda_1, ..., \lambda_R$) in the domain of f. It also determines optimal solutions at these transition points (ie. $\{\bar{\boldsymbol{x}}^*(\lambda_1), ..., \bar{\boldsymbol{x}}^*(\lambda_R)\}$ where $\bar{\boldsymbol{x}}^*(\lambda_1) \in \bar{P}_{\lambda_1}, ..., \bar{\boldsymbol{x}}^*(\lambda_R) \in \bar{P}_{\lambda_R}$. We note that this parametric programming algorithm is not the most common one used in the literature [GG97]. We chose to use OPPPA and create a parametric programming algorithm that is a modification of OPPPA (which we call ROPPPA in section 5.3) for the following reasons.

1. First we use FICO Xpress-IVE, an optimization solver for linear programming and mixed integer programming, to code all LP formulations and parametric programming algorithms. Due to the way that the FICO Xpress-IVE software is designed, as far as we know, there does not exist a way to implement the classical approach to parametric programming in the software. However, because OPPPA involves solving a series of LPs we were able to implement OPPPA in FICO Xpress-IVE (and the parametric programming algorithm we propose in 5.3 which is based on OPPPA).

2. Additionally, the first part in executing the classical parametric programming algorithm in literature involves finding an optimal solution of $\bar{P}_{\lambda_1}$ for a given λ_1 using the Simplex algorithm. However, numerical tests show that solving $\bar{P}_{\lambda_1}$ becomes computationally burdensome as n becomes large. This is because $\bar{P}_\lambda$ has $3\binom{n}{3} + 1$ constraints. We choose to use OPPPA because we can make modifications to the algorithm (as shown in ROPPPA presented in chapter 5.3) such that not all of these $3\binom{n}{3} + 1$ constraints are needed. The classical parametric programming algorithm in the literature [GG97], however needs to use all of the constraints in order for the algorithm to work appropriately.

We describe the algorithm below. In each step we provide a summary of the step and then provide the necessary mathematical notation.

Optimal Partition Parametric Programming Algorithm (OPPPA)

Input: Parametric LP $\bar{P}_\lambda = \{max\ \bar{\boldsymbol{c}}'\bar{\boldsymbol{x}}|\bar{\boldsymbol{A}}\bar{\boldsymbol{x}} = \bar{\boldsymbol{b}}(\lambda), \bar{\boldsymbol{x}} \geq \boldsymbol{0}\}$ and range $[0, \binom{n}{2}]$ (or equivalently putting $P_\lambda = \{max\ \boldsymbol{c}'\boldsymbol{x}|\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{e}'\boldsymbol{x} = \binom{n}{2} - \lambda, \boldsymbol{x} \geq \boldsymbol{0}\}$ in standard form).

Output: Set of transition points $\{\lambda_1, .., \lambda_R\}$ of the function $f(\lambda) := \max_{\bar{\boldsymbol{x}}(\lambda)\in\bar{P}_\lambda} \boldsymbol{c}'\bar{\boldsymbol{x}}^*(\lambda)$ and the set of optimal solutions $\{\bar{\boldsymbol{x}}^*(\lambda_1), ..., \bar{\boldsymbol{x}}^*(\lambda_R)\}$ at these transition points (and subsequently $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$).

- **Step 1:**
  - *Summary:* Solve $\bar{P}_{\lambda_1} = \bar{P}_0$. Note that this will always result in an optimal solution $\bar{x}^*(\lambda_1)$ for which $x^*(\lambda_1)_{ij} = 1$ for all (i, j) pairs and $\boldsymbol{z}^* = 0$. Thus this first step always results in a binary solution which defines a consensus clustering with all objects in the same one cluster.

    1a Set $r = 1$.

    1b Set $\lambda_r = 0$.

    1c Solve $\bar{P}_{\lambda_r}$, define $\bar{\boldsymbol{x}}^*(\lambda_r) = (\bar{\boldsymbol{x}}^*(\lambda_r), \boldsymbol{z}^*(\lambda_r))$ as the optimal solution.

- **Step 2:**
  - *Summary:* Find a dual optimal solution $(\boldsymbol{y}^*, \boldsymbol{s}^*)$ of $\bar{\bar{D}}_{\lambda_r}$ that yields the most negative slope of $f(\lambda)$ at the transition point λ_r.

    2a Solve $\{max y_m|(\boldsymbol{y}, \boldsymbol{s}) \in (\bar{\bar{D}}), \boldsymbol{s}'\bar{\boldsymbol{x}}^*(\lambda_r) = 0\}$, where y_m is the dual variable in $(\boldsymbol{y}, \boldsymbol{s})$ that corresponds to the primal constraint $\sum_{i<j} \bar{x}_{ij} = M(\lambda)$. Define the optimal solution as $(\boldsymbol{y}^*, \boldsymbol{s}^*)$.

- **Step 3:**
  - *Summary:* Find the maximum value of λ such that $(\boldsymbol{y}^*, \boldsymbol{s}^*)$ is still optimal in $\bar{\bar{D}}_\lambda$.

    3a Solve $\{max\ \lambda|\bar{\boldsymbol{x}}(\lambda) \in (\bar{P}_\lambda), \bar{\boldsymbol{x}}(\lambda)'\boldsymbol{s}^* = 0\}$. Define the optimal value of λ as λ_{r+1} and the optimal value of $\bar{\boldsymbol{x}}(\lambda)$ as $\bar{\boldsymbol{x}}^*(\lambda_{r+1}) = (\boldsymbol{x}^*(\lambda_{r+1}), \boldsymbol{z}^*(\lambda_{r+1}))$.

    3b If $\bar{\boldsymbol{x}}(\lambda_{r+1}) = \bar{\boldsymbol{x}}(\lambda_r)$, then the algorithm terminates. Otherwise, let $r = r + 1$ and go to step 2.

Note that this algorithm in the literature can be used to find the transition points and corresponding optimal solutions at these transition points for any RHS parametric LP. However, the way we describe it above is specifically written for our parametric model $\bar{P}_\lambda$.

We now discuss some properties of the Optimal Partition Parametric Programming algorithm (OPPPA) to assist us in explaining why this algorithm is better equipped to meet the goals of gene expression consensus clustering than the classical approach to parametric programming (section 5.4).

Theorem 5.2.1 *If $\{\lambda_1, \lambda_2, ..., \lambda_R\}$ is the set of transition points in the domain of $f(\lambda)$ (in the case of the $\bar{P}_\lambda$ this is $[0, \binom{n}{2}]$) where $\lambda_1 < \lambda_2 < ... < \lambda_R$, then the OPPPA will return the distinct pairs $(\lambda_r, \bar{\boldsymbol{x}}^*(\lambda_r))$ in iteration r where $\bar{\boldsymbol{x}}^*(\lambda_r) \in P^*_{\lambda_r}$ for $r = 1, ..., R$ respectively.*

Proof Berkelaar et al. [Ber96] prove the theorem above. The proof for this is quite extensive but we will briefly summarize each step of the algorithm and why it works.

Step (1) Since $\lambda_1 = 0$ is an end point of the domain of $f(\lambda)$, it is a transition point.

Steps (2) and (3)

Slope of the Linearity Intervals (Interior) The proof of lemma 5.1.3 defined $f(\lambda) := \underbrace{min}_{l=1,...,L} \left([(\boldsymbol{y}^l)'\boldsymbol{b}] - \lambda[(\boldsymbol{y}^l)'\boldsymbol{\Delta b}]\right)$, a continous concave piecewise linear function where $\{\boldsymbol{y}^1, ..., \boldsymbol{y}^L\}$ is the set of extreme point of $\bar{D}$. The linear intervals of this function are the linearity intervals and the points λ in between two linearity intervals are the transition points. It is easy to see that for any λ in the interior of a linearity interval, that $f'(\lambda) = -(\boldsymbol{y}^l)'\boldsymbol{\Delta b} = -(\boldsymbol{y}^l)'[0, 0, ..., 0, 1] = -\boldsymbol{y}^l_m$ where $\boldsymbol{y}^l \in \bar{\bar{D}}^*_\lambda$ and y_m is the dual variable that corresponds to the parametric constraint in $\bar{P}_\lambda$.

Local Linearity and Differentiability of the Interior of Linearity Intervals Formulating the 'slope' at a transition point of f first requires the following lemma.

Lemma 5.2.1 *If a nondegenerate primal optimal basic feasible solution exists for $\bar{P}_\lambda$, then λ is in the interior of one of the linearity intervals of $f(\lambda)$ and hence $f(\lambda)$ is differentiable at λ.*

Proof Let $\boldsymbol{B}$ be the basis matrix for an optimal nondegenerate basic feasible solution $\bar{\boldsymbol{x}}$ of $\bar{P}_{\bar{\lambda}}$. Since $\bar{\boldsymbol{x}}$ is nondegenerate, this means the basic variables $\bar{\boldsymbol{x}}_{BV}$ of $\bar{\boldsymbol{x}}$ have the property $\bar{\boldsymbol{x}}_{BV} = \boldsymbol{B}^{-1}\boldsymbol{b}(\bar{\lambda}) > 0$ and thus $\bar{\boldsymbol{x}}_{BV} = \boldsymbol{B}^{-1}\boldsymbol{b}(\bar{\lambda} \pm \delta) > \boldsymbol{0}$ for some $\delta > 0$. And since the primal reduced costs of $\bar{\boldsymbol{x}}$ are not changed by perturbing λ, $\boldsymbol{B}$ is an optimal feasible basis for a local interval $(\bar{\lambda} - \delta, \bar{\lambda} + \delta)$. Thus we can say $f(\lambda) = \boldsymbol{c}'_B\boldsymbol{B}^{-1}\boldsymbol{b}(\lambda)$ for $\lambda \in (\bar{\lambda} - \delta, \bar{\lambda} + \delta)$ where $\boldsymbol{y} = \boldsymbol{c}'_B\boldsymbol{B}^{-1}$ is the corresponding dual optimal solution of $\bar{D}_\lambda$ for $\lambda \in (\bar{\lambda} - \delta, \bar{\lambda} + \delta)$. Hence $f(\lambda)$ is linear for $\lambda \in (\bar{\lambda} - \delta, \bar{\lambda} + \delta)$, thus it is in the interior of one of the linearity intervals. And since $\bar{\lambda}$ is locally linear, this implies f is differentiable at $\bar{\lambda}$.

Multiple Dual Optimal Solutions at Transition Points The contrapositive of lemma 5.2.1 states that if f is not in the interior of one of the linearity intervals of $f(\lambda)$ at λ, then any optimal basic feasible solution $\bar{\boldsymbol{x}}(\lambda)$ of $\bar{P}_\lambda$ is degenerate. A transition point λ^* of $f(\lambda)$ is not in the interior of a linearity interval and non-differentiable and hence any optimal basic feasible solution $\bar{\boldsymbol{x}}^*(\lambda^*)$ is primal degenerate. Primal degeneracy of an optimal basic feasible solution of $\bar{P}_{\lambda^*}$ implies there exists multiple optimal dual solutions $\boldsymbol{y}$ of $\bar{D}_{\lambda^*}$ (and hence multiple optimal dual solutions $(\boldsymbol{y}, \boldsymbol{s}) of \bar{\bar{D}}_{\lambda_r})$.

The Set of all Dual Optimal solutions of $\bar{\bar{D}}_{\lambda_r}$ Step (2) of the OPPPA solves the linear programming formulation $\{max\, y_m | (\boldsymbol{y}, \boldsymbol{s}) \in (\bar{\bar{D}}), \boldsymbol{s}'\bar{\boldsymbol{x}}^*(\lambda_r) = 0\}$. We show that the feasible region (DCS) of this LP is the set of all dual optimal solutions of $\bar{\bar{D}}_{\lambda_r}$. Let $(\boldsymbol{y}, \boldsymbol{s}) \in (DCS)$. Then $(\boldsymbol{y}, \boldsymbol{s})$ is feasible in the $\bar{\bar{D}}_{\lambda_r}$. Since $\boldsymbol{s}'\bar{\boldsymbol{x}}^*(\lambda_r) = 0$, $(\boldsymbol{y}, \boldsymbol{s})$ and $\bar{\boldsymbol{x}}^*(\lambda_r)$ satisfy the complementary slackness equations. Since $\bar{\boldsymbol{x}}^*(\lambda_r)$ is feasible and optimal (from step (1) or (3)) in $\bar{P}_{\lambda_r}$ this implies that $(\boldsymbol{y}, \boldsymbol{s})$ is optimal in $\bar{\bar{D}}_{\lambda_r}$.

The Set of all $(\lambda, \bar{\boldsymbol{x}}^*(\lambda))$ **for which**$(\boldsymbol{y}^*, \boldsymbol{s}^*)$ **is still Optimal in** $\bar{\bar{D}}_\lambda$ Step (3) of the OPPPA solves the linear program $\{max\, \lambda | \bar{\boldsymbol{x}}(\lambda) \in (\bar{P}_\lambda), \bar{\boldsymbol{x}}(\lambda)'\boldsymbol{s}^* = 0\}$, where $(\boldsymbol{y}^*, \boldsymbol{s}^*)$ is the dual optimal solution of the $\bar{\bar{D}}_{\lambda_r}$ selected in step (2). We show that the feasible region of this LP from step (3), which we call $(PCS(\boldsymbol{s}^*))$, is the set of all optimal solution pairs $(\lambda, \bar{\boldsymbol{x}}^*(\lambda))$ of $\bar{P}_\lambda$ for which $(\boldsymbol{y}^*, \boldsymbol{s}^*)$ is still optimal in $\bar{\bar{D}}_\lambda$. Let $(\lambda, \bar{\boldsymbol{x}}(\lambda)) \in (PCS(\boldsymbol{s}^*))$. Then $\bar{\boldsymbol{x}}(\lambda)$ is feasible in $\bar{P}_\lambda$. Since $(\boldsymbol{y}^*, \boldsymbol{s}^*)$ was a dual optimal solution of $\bar{\bar{D}}_{\lambda_r}$ (found in step (2)), it was feasible in $(\bar{\bar{D}}_{\lambda_r})$. Since $(\bar{\bar{D}}_{\lambda_r}) = (\bar{\bar{D}}) = (\bar{\bar{D}}_\lambda)$, $(\boldsymbol{y}^*, \boldsymbol{s}^*)$ is feasible in $\bar{\bar{D}}_\lambda$. Since $\bar{\boldsymbol{x}}(\lambda)'\boldsymbol{s}^* = 0$, $\bar{\boldsymbol{x}}(\lambda)$ and $(\boldsymbol{y}^*, \boldsymbol{s}^*)$ satisfy the complementary slackness equations. Thus by complementary slackness, $\bar{\boldsymbol{x}}(\lambda)$ and $(\boldsymbol{y}^*, \boldsymbol{s}^*)$ are optimal in $\bar{P}_\lambda$ and $\bar{\bar{D}}_\lambda$ respectively.

Selecting a $(\boldsymbol{y}^*, \boldsymbol{s}^*)$ **in (2) that will Yield a Unique Transition Point** λ_{r+1} Since the goal of the OPPPA is to find the unique transition points of $f(\lambda)$, $\{\lambda_1, \lambda_2, ..., \lambda_R\}$, the goal of the r^{th} iteration of the OPPPA is to find λ_{r+1} where $\lambda_r < \lambda_{r+1}$. We discussed how at a transition point λ_r, there are multiple dual optimal solutions in the set $\bar{\bar{D}}^*_{\lambda_r}$. The optimal value of λ found in step (3) with $\{max\ \lambda | \bar{\boldsymbol{x}}(\lambda) \in (\bar{P}_\lambda), \bar{\boldsymbol{x}}(\lambda)'\boldsymbol{s}^* = 0\}$ depends upon what dual optimal solution $(\boldsymbol{y}^*, \boldsymbol{s}^*) \in \bar{\bar{D}}^*_{\lambda_r}$ was selected in step (2). Let us define $\bar{\bar{D}}^*_{\lambda_r^+}$ to be the set of dual optimal solutions of the interior of the linearity interval to the right of λ_r. Berkelaar et al. [Ber96] prove that $\bar{\bar{D}}^*_{\lambda_r^+} \subset \bar{\bar{D}}^*_{\lambda_r}$. Suppose that in the LP $\{max\ \lambda | \bar{\boldsymbol{x}}(\lambda) \in (\bar{P}_\lambda), \bar{\boldsymbol{x}}(\lambda)'\boldsymbol{s}^* = 0\}$, we use $(\boldsymbol{y}^*, \boldsymbol{s}^*) \in \bar{\bar{D}}^*_{\lambda_r} / \bar{\bar{D}}^*_{\lambda_r^+}$. Thus $(\boldsymbol{y}^*, \boldsymbol{s}^*)$ is not optimal in $\bar{\bar{D}}_\lambda$ for $\lambda > \lambda_r$, and by complementary slackness $\bar{\boldsymbol{x}}(\lambda)'\boldsymbol{s}^* \neq 0$ for $\lambda > \lambda_r$. Thus the maximum value of λ is λ_r when $(\boldsymbol{y}^*, \boldsymbol{s}^*) \in \bar{\bar{D}}^*_{\lambda_r} / \bar{\bar{D}}^*_{\lambda_r^+}$. In other words, we have not increased the value of λ, so it is still the same transition point as before. Thus in order to find a $\lambda_{r+1} > \lambda_r$ in step (3), we must select a $(\boldsymbol{y}^*, \boldsymbol{s}^*) \in \bar{\bar{D}}_{\lambda_r^+}$ step (2).

It follows from Berkelaar et al. [Ber96] that if f is concave continuous piecewise linear and if $(\boldsymbol{y}^*, \boldsymbol{s}^*) \in \bar{\bar{D}}^*_{\lambda_r}$, then $-\boldsymbol{\Delta b}'\boldsymbol{y}^*$ lies between the slopes of the linearity intervals to the left and the right of λ_r (or $-\boldsymbol{\Delta b}'\boldsymbol{y}^+ \leq -\boldsymbol{\Delta b}'\boldsymbol{y}^* \leq -\boldsymbol{\Delta b}'\boldsymbol{y}^-$, where $\boldsymbol{y}^- \in \bar{\bar{D}}^*_{\lambda_r^-}$ and $\boldsymbol{y}^+ \in \bar{\bar{D}}^*_{\lambda_r^+}$). They also show that $\boldsymbol{\Delta b}'\boldsymbol{y}^* = \boldsymbol{\Delta b}'\boldsymbol{y}^+$ if and only if $\boldsymbol{y}^* \in \bar{\bar{D}}^*_{\lambda_r^+}$. Since $\bar{\bar{D}}^*_{\lambda_r^+} \subseteq \bar{\bar{D}}^*_{\lambda_r}$ and $\boldsymbol{\Delta b}'\boldsymbol{y}^* = \boldsymbol{\Delta b}'\boldsymbol{y}^+$ if and only if $\boldsymbol{y}^* \in \bar{\bar{D}}^*_{\lambda+}$, solving $\{min\ -\boldsymbol{\Delta b}'\boldsymbol{y} | (\boldsymbol{y}, \boldsymbol{s}) \in (\bar{\bar{D}}), \boldsymbol{s}'\bar{\boldsymbol{x}}^*(\lambda_r) = 0\}$ will yield a dual solution $(\boldsymbol{y}^*, \boldsymbol{s}^*) \in \bar{\bar{D}}_{\lambda_r^+}$. This LP is equivalent to solving $\{max\ \boldsymbol{\Delta b}'\boldsymbol{y} | (\boldsymbol{y}, \boldsymbol{s}) \in (\bar{\bar{D}}), \boldsymbol{s}'\bar{\boldsymbol{x}}^*(\lambda_r) = 0\}$ which is what we have in step (2).

Step (3) Finds the Next Transition Point λ_{r+1} Since the transition point λ_{r+1} is the endpoint of the linearity interval to the right of λ_r, and a linearity interval is the range of λ for which $(\boldsymbol{y}^*, \boldsymbol{s}^*)$ is optimal, then solving $\{max\ \lambda | \bar{\boldsymbol{x}}(\lambda) \in (\bar{P}_\lambda), \bar{\boldsymbol{x}}(\lambda)'\boldsymbol{s}^* = 0\}$ will yield λ_{r+1}.

5.3 Relaxed Optimal Partition Parametric Programming Algorithm

As discussed in the introduction, numerical tests in the literature have shown that the problem of finding an optimal solution to the GWLP (or P) becomes intractable as n becomes large. This is due largely to the fact that there are $3\binom{n}{3}$ constraints in the GWLP. Our numerical tests in chapter 7 likewise show that the task of running the OPPPA to completion becomes intractable as n becomes large. Because the feasible regions of P_λ and the GWLP are so similar, when employing a parametric programming algorithm on P_λ we would like to maintain a computation time that is comparable to other fast solving methods applied to the GWLP like those employed by Downing et al. [Dow10] (discussed in section 1.4). These methods involve iteratively adding constraints from the feasible region of (P) to produce a relaxed feasible region (R) for which the optimal solution $\boldsymbol{x}^*$ of R is also in P, (and thus optimal for P.)

We present a modifed version of the OPPPA which we call the Relaxed Optimal Partition Parametric Programming Algorithm (ROPPPA) which makes use of these methods. First, in section 5.3.1 we define these methods, which consist of a constraint addition routine (ie. row generation technique) to be performed on the certain LP formulations that are solved when executing OPPPA. In section 5.3.2 we show ROPPPA. Finally, in section 5.3.3 we discuss how we modified OPPPA to get ROPPPA.

5.3.1 Constraint Addition Routine

Constraint Addition Routine (CAR) This routine starts with an LP formulation that is a relaxation of the LP formulation $PCS(\boldsymbol{s}^*) := \{max\lambda | \boldsymbol{x}(\lambda) \in (P_\lambda), [\boldsymbol{x}(\lambda)' | (b_{ijk}^{(h)} - \boldsymbol{a}_{ijk}^{(h)} \boldsymbol{x}(\lambda))']' \boldsymbol{s}^* = 0\}$. We call this relaxation $RCS(\boldsymbol{s}^*)$, (note $(PCS(\boldsymbol{s}^*)) \subseteq (RCS(\boldsymbol{s}^*))$). The routine iteratively adds to $RCS(\boldsymbol{s}^*)$ all constraints in $PCS(\boldsymbol{s}^*)$ which are violated by the current optimal solution of $RCS(\boldsymbol{s}^*)$. The routine terminates when no more constraints in $PCS(\boldsymbol{s}^*)$ are violated by the current optimal solution of $RCS(\boldsymbol{s}^*)$.

Input: An LP formulation $PCS(\boldsymbol{s}^*) := \{max\ \lambda | \boldsymbol{x}(\lambda) \in (P_\lambda), [\boldsymbol{x}(\lambda)' | (b_{ijk}^{(h)} - \boldsymbol{a}_{ijk}^{(h)} \boldsymbol{x}(\lambda))']' \boldsymbol{s}^* = 0\}$

Output: An optimal solution $(\lambda^*, \boldsymbol{x}^*(\lambda^*))$ of $PCS(\boldsymbol{s}^*)$.

Step 1
- *Summary:* Solve an initial relaxed LP of $PCS(\boldsymbol{s}^*)$.
  - **1a** Define $RCS(\boldsymbol{s}^*) := \{max\lambda | \sum_{1 \le i < j \le n} \boldsymbol{x}(\lambda)_{ij} = M(\lambda), [\boldsymbol{x}(\lambda)' | (b_{ijk}^{(h)} - \boldsymbol{a}_{ijk}^{(h)} \boldsymbol{x}(\lambda))']' \boldsymbol{s}^* = 0, \boldsymbol{x}(\lambda) \ge \boldsymbol{0}$ and find an optimal solution.
  - **1b** Define this optimal solution as $(\lambda, \boldsymbol{x}^*(\lambda))$.

Step 2
- *Summary:* Add all constraints in $(PCS(\boldsymbol{s}^*))$ that are violated by $(\boldsymbol{x}^*(\lambda))$.
  - **2a** For all constraints ${\boldsymbol{a}'_{ijk}}^{(h)} \boldsymbol{x}^*(\lambda) \le b_{ijk}^{(h)}$ in $(PCS(\boldsymbol{s}^*)) \backslash (RCS(\boldsymbol{s}^*))$ for which ${\boldsymbol{a}'_{ijk}}^{(h)} \boldsymbol{x}^*(\lambda) > b_{ijk}^{(h)}$, add ${\boldsymbol{a}'_{ijk}}^{(h)} \boldsymbol{x}^*(\lambda) \le b_{ijk}^{(h)}$ to $(RCS(\boldsymbol{s}^*))$.
  - **2b** If ${\boldsymbol{a}'_{ijk}}^{(h)} \boldsymbol{x}^*(\lambda) \le b_{ijk}^{(h)}$ for all constraints in $(PCS(\boldsymbol{s}^*)) \backslash (RCS(\boldsymbol{s}^*))$, then the algorithm terminates.

Step 3
- Solve $RCS(\boldsymbol{s}^*)$ and define the optimal solution as $(\lambda, \boldsymbol{x}^*(\lambda))$. Return to step (2).

5.3.2 Algorithm

Now we introduce the ROPPPA which makes use of the constraint addition routine (row generation technique) shown in section 5.3.1. One thing to note is that OPPPA uses $\bar{P}_\lambda$ as input, while ROPPPA uses P_λ as input. We will explain why we did this in section 5.3.3.

Relaxed Optimal Partition Parametric Programing Algorithm (ROPPPA)

Input: Parametric LP $P_\lambda = \{max\ \boldsymbol{c}'\boldsymbol{x} | \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{e}'\boldsymbol{x} = \binom{n}{2} - \lambda, \boldsymbol{x} \geq \boldsymbol{0}\}$ and range $[0, \binom{n}{2}]$.

Output: Set of transition points $\{\lambda_1, .., \lambda_R\}$ of the function $f(\lambda) := \max_{\boldsymbol{x}(\lambda) \in P_\lambda} \boldsymbol{c}'\boldsymbol{x}^*(\lambda)$ and the set of optimal solutions $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ at these transition points.

Step 1

- *Summary:* Start with the optimal solution to $P_{\lambda_1} = P_0$, which we already know.

1a Set $r = 1$.

1b Set $\lambda_r = 0$.

1c Set $\bar{\boldsymbol{x}}(\lambda_r) = (\boldsymbol{x}(\lambda_r), \boldsymbol{z}(\lambda_r))$, where $\boldsymbol{x}^*(\lambda_r)_{ij} = 1$ for all $i < j$ and $\boldsymbol{z}^*(\lambda_r)^{(h)}_{ijk} = 0$ for all $i < j < k$ and $h = 1, 2, 3$.

Step 2

- *Summary:* Using a feasible region that is a simplified equivalent version of that in step (2) of OPPPA, find a dual optimal solution $(\boldsymbol{y}^*, \boldsymbol{s}^*)$ of $\bar{\bar{D}}_{\lambda_r}$ that yields the most negative slope of $f(\lambda)$ at the transition point λ_r.

2a Solve

$$\begin{array}{lll} \max & y_m & \\ \text{st} & \boldsymbol{y}'\bar{\boldsymbol{A}}_{ij} \geq c_{ij} & \forall i < j \\ & \boldsymbol{y} \geq \boldsymbol{0} & \\ & \sum_{i<j}(\boldsymbol{y}'\bar{\boldsymbol{A}}_{ij} - c_{ij})x^*(\lambda_r)_{ij} + \sum_{h=1}^{3}\sum_{i<j<k}(y^{(h)}_{ijk})z^*(\lambda_r)^{(h)}_{ijk} = 0 & \end{array}$$

, where y_m is the dual variable in $(\boldsymbol{y}, \boldsymbol{s})$ that corresponds to the primal constraint $\sum_{i<j} x_{ij} = M(\lambda)$. Define $\boldsymbol{y}^*$ as the optimal solution.

2b If $\boldsymbol{y}^*$ is the optimal solution, define ${s^*_{ijk}}^{(h)} := {y^*_{ijk}}^{(h)}$ for all $1 \leq i < j < k \leq n$ and $h = 1, 2, 3$; and $s^*_{ij} := (\boldsymbol{y}^{*\prime}\bar{\boldsymbol{A}}_{ij} - c_{ij})$ for all $1 \leq i < j \leq n$.

Step 3

- *Summary:* Find the maximum value of λ such that $(\boldsymbol{y}^*, \boldsymbol{s}^*)$ is still optimal in $\bar{D}_{\lambda_r}$ using CAR.

3a Find an optimal solution of $PCS(\boldsymbol{s^*}) := \{max\ \lambda | \boldsymbol{x}(\lambda) \in (P_\lambda), [\boldsymbol{x}(\lambda)' | (b^{(h)}_{ijk} - \boldsymbol{a}^{(h)}_{ijk}\boldsymbol{x}(\lambda))']'\boldsymbol{s}^* = 0\}$ using CAR.

3b Define the optimal solution as $(\lambda_{r+1}, \boldsymbol{x}^*(\lambda_{r+1}))$.

3c Define $z^*(\lambda_{r+1})^{(h)}_{ijk} := (b^{(h)}_{ijk} - \boldsymbol{a}^{(h)}_{ijk}\boldsymbol{x}^*(\lambda))$ for all $1 \leq i < j < k \leq n$ and $h = 1, 2, 3$.

3d If $\boldsymbol{x}^*(\lambda_{r+1}) = \boldsymbol{x}^*(\lambda_r)$, then the algorithm terminates. Otherwise, set $r := r + 1$ and go to step 2.

5.3.3 Discussion

In practice, the algorithm $ROPPPA$ executes the OPPPA on P_λ for $\lambda \in [0, \binom{n}{2}]$ using a significantly reduced set of constraints in the LP solved in step (2). Note that steps (1)-(3) of OPPPA each have an LP that needs to be solved. We discuss the inefficiencies in solving these LPs stated in steps (1), (2), and (3) of the OPPPA. We then explain the improved techniques used in the $ROPPPA$ which (for steps (1) and (3)) greatly reduced the computation time when compared to the $OPPPA$. Although we do not apply any improved techniques to the LP solved in step (2) in ROPPPA, we discuss some suggested techniques that *could* be applied in future work.

Step (1) of the OPPPA This step (executed only once) solves the primal problem $\bar{P}_{\lambda_1} = \bar{P}_0$. However, we know that the only optimal solution of $\bar{P}_0$ is $x^*(0)_{ij} = 1$ for all $i < j$ and $z(0)^{*(h)}_{ijk} = 0$ for all $i < j < k$ and $h = 1, 2, 3$, because $\bar{P}_0$ contains only this one solution. Since all we need to proceed to step (2) is the optimal solution $\bar{\boldsymbol{x}}^*(0) = (\boldsymbol{x}^*(0), \boldsymbol{z}^*(0))$, we do not need to actually solve $\bar{P}_0$.

Step (2) of the OPPPA This step (executed in each iteration) solves $\{max\ y_m | (\boldsymbol{y}, \boldsymbol{s}) \in (\bar{\bar{D}}), \boldsymbol{s}'\bar{\boldsymbol{x}}^*(\lambda_r) = 0\}$. Written out another way this is:

$$
\begin{array}{lll}
\max & y_m & \\
\text{st} & \boldsymbol{y}'\bar{\boldsymbol{A}}_{ij} - s_{ij} = c_{ij} & \forall 1 \le i < j \le n \\
 & y^{(h)}_{ijk} - s^{(h)}_{ijk} = 0 & \forall^{1 \le i<j<k \le n,}_{h=1,2,3} \\
 & s^{(h)}_{ijk} \ge 0 & \forall^{1 \le i<j<k \le n,}_{h=1,2,3} \\
 & s_{ij} \ge 0 & \forall 1 \le i < j \le n \\
 & y_m URS & \\
 & \sum_{i<j} s_{ij} x^*(\lambda_r)_{ij} + \sum_{h=1}^{3} \sum_{i<j<k} s^{(h)}_{ijk} {}_{ijk} z^*(\lambda_r)^{(h)}_{ijk} = 0 &
\end{array}
$$

This formulation has $2 \cdot 3\binom{n}{3} + \binom{n}{2} + 1$ variables and $\binom{n}{2} + 3\binom{n}{3} + 1$ main constraints. It was written in this format because $\boldsymbol{s}^* = (s^*_{12}, ..., s^*_{n-1,n}, s^{*(1)}_{123}, ..., s^{*(3)}_{n-2,n-1,n})$ is needed in order to solve the LP in step (3). However, we can greatly reduce the number of constraints and variables needed in order to solve the LP in step (2) by rewriting it as:

$$
\begin{array}{lll}
\max & y_m & \\
\text{st} & \boldsymbol{y}'\bar{\boldsymbol{A}}_{ij} \geq c_{ij} & \forall 1 \leq i < j \leq n \\
& y^{(h)}_{ijk} \geq 0 & \forall^{1 \leq i<j<k \leq n,}_{h=1,2,3} \\
& y_m URS & \\
& \sum_{i<j}(\boldsymbol{y}'\boldsymbol{A}_{ij} - c_{ij})x^*(\lambda_r)_{ij} + \sum_{h=1}^{3}\sum_{i<j<k} y^{(h)}_{ijk} z^*(\lambda_r)^{(h)}_{ijk} = 0 &
\end{array}
$$

Here we deleted the s_{ij} and $s^{(h)}_{ijk}$ variables and the simplified $y^{(h)}_{ijk} - s^{(h)}_{ijk} = 0$ constraints. After solving the given equivalent simplified LP formulation, we can calculate the extra optimal variables needed in step (3) by setting $s^{*(h)}_{ijk} := y^{*(h)}_{ijk}$ and $s^*_{ij} := \boldsymbol{y}*'\,\boldsymbol{A}_{ij} - c_{ij}$.

The simplified LP formulation now has $\binom{n}{2} + 1$ main constraints and $3\binom{n}{3} + 1$ variables. This LP formulation has a great many more variables than constraints. For future work, a possible way to make ROPPPA execute faster could involve applying column generation techniques to this simplified LP formulation in step (2).

Step (3) of the OPPPA This step (executed in each iteration) solves the primal problem $\{max\ \lambda | \bar{\boldsymbol{x}}(\lambda) \in (\bar{P}_\lambda), \bar{\boldsymbol{x}}(\lambda)'\boldsymbol{s}^* = 0\}$ which has $3\binom{n}{3} + 2$ constraints and $\binom{n}{2} + 3\binom{n}{3} + 1$ variables. We can first simplify this LP formulation to the equivalent LP formulation $\{max\,\lambda | \boldsymbol{x}(\lambda) \in (P_\lambda), [\boldsymbol{x}(\lambda)' | (b^{(h)}_{ijk} - \boldsymbol{a}^{(h)}_{ijk}\boldsymbol{x}(\lambda))']'\boldsymbol{s}^* = 0\}$ which now has $\binom{n}{2} + 1$ variables and $3\binom{n}{3} + 2$ constraints. This LP formulation has many more constraints than variables. We can employ the CAR methods (from section 5.3.1) to the problem to reduce the number of constraints needed to find an optimal solution to this LP formulation.

In these methods we start with an initial problem of
$RCS(\boldsymbol{s}^*) = \{max\ \lambda | \sum_{i<j} x(\lambda)_{ij} = M(\lambda), [\boldsymbol{x}(\lambda)' | (b^{(h)}_{ijk} - \boldsymbol{a}^{(h)}_{ijk}\boldsymbol{x}(\lambda))']'\boldsymbol{s}^* = 0, \boldsymbol{x} \geq \boldsymbol{0}\}$

Here we initialized with the nonnegativity constraints, the summation constraint, and the complementary slackness constraint, but we substituted $(b^{(h)}_{ijk} - \boldsymbol{a}^{(h)}_{ijk}{}'\boldsymbol{x}(\lambda))$ in for $z^{(h)}_{ijk}$ in the complementary slackness constraint.

Numerical tests showed that by using CAR in this fashion greatly reduced the computational time needed to execute step (3) in the ROPPPA as opposed to step (3) in OPPPPA.

5.4 Discussion

OPPPA and ROPPPA described above return a finite set of optimal solutions $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ of $P_{\lambda_1}, ..., P_{\lambda_{\bar{R}}}$ respectively when applied to $\bar{P}_\lambda$ (P_λ) for $\lambda \in [0, \binom{n}{2})]$. Unfortunately, we cannot guarantee the set of solutions $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ returned by OPPPA or ROPPPA define a CCA-β solution. However, we can show that the set of solutions $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ returned by OPPPA or ROPPPA will always satisfy the following necessary (but not sufficient) conditions for a set of solutions $\{\boldsymbol{x}(\lambda_1), ..., \boldsymbol{x}(\lambda_R)\}$ defining a CCA-β solution.

Necessary Conditions of a Vector Set Defining a CCA-β Solution

Lemma 5.4.1 *Let $\{\boldsymbol{x}(\lambda_1), ..., \boldsymbol{x}(\lambda_R)\}$ be a set of vector solutions ($\boldsymbol{x}(\lambda_1) \in P_{\lambda_1}, ..., \boldsymbol{x}(\lambda_R) \in P_{\lambda_R}$) that define a CCA-$\beta$ solution $\{p^1, ..., p^R\}$. The following necessary conditions must hold.*

1. *$\boldsymbol{x}(\lambda_1)$ defines a consensus clustering p^1 with all objects in the same cluster ($p^1 = O$).*
2. *$\boldsymbol{x}(\lambda_R)$ defines a consensus clustering p^R with all objects in singleton clusters ($p^r = \{\{o_1\}, ..., \{o_n\}\}$).*
3. *$\boldsymbol{x}(\lambda_r) \neq \boldsymbol{x}(\lambda_{r'})$ for all $1 \leq r < r' \leq R$.*

Proof

1. Because $\{\boldsymbol{x}(\lambda_1), ..., \boldsymbol{x}(\lambda_R)\}$ defines a CCA-β solution $\{p^1, ..., p^R\}$ and the first consensus clustering in a CCA-β solution is the one with all objects in the same cluster, $\boldsymbol{x}(\lambda_1)$ defines a consensus clustering with all objects in the same cluster.
2. Because $\{\boldsymbol{x}(\lambda_1), ..., \boldsymbol{x}(\lambda_R)\}$ defines a CCA-β solution $\{p^1, ..., p^R\}$ and the last consensus clustering in a CCA-β solution is the one with all objects in the singleton clusters, $\boldsymbol{x}(\lambda_1)$ defines a consensus clustering with all objects in singleton clusters.
3. Because $\{\boldsymbol{x}(\lambda_1), ..., \boldsymbol{x}(\lambda_R)\}$ defines a CCA-β solution $\{p^1, ..., p^R\}$ and a CCA-β solution does not return any duplicate consensus clusterings (ie $p^r \neq p^{r'}$), then $\boldsymbol{x}(\lambda_r) \neq \boldsymbol{x}(\lambda_{r'})$ for all $1 \leq r < r' \leq R$.

In the lemmas below, we show that the results of CCA-OPPPA and CCA-ROPPPA always satisfy these necessary conditions.

CCA-OPPPA and CCA-ROPPPA Results Satisfy These Necessary Conditions

Lemma 5.4.2 *Let $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ be a set of vector solutions returned by CCA-OPPPA or CCA-ROPPPA. The following conditions must hold.*

1. *$\boldsymbol{x}^*(\lambda_1)$ defines a consensus clustering p^1 with all objects in the same cluster ($p^1 = O$).*

2. *$\boldsymbol{x}^*(\lambda_1)$ defines a consensus clustering p^R with all objects in singleton clusters ($p^r = \{\{o_1\}, ..., \{o_n\}\}$).*

3. *$\boldsymbol{x}^*(\lambda_r) \neq \boldsymbol{x}^*(\lambda_{r'})$ for all $1 \leq r < r' \leq R$.*

Proof

1. Step (1) of OPPPA and ROPPPA always return $\boldsymbol{x}^*(\lambda_1) \in P_{\lambda_1}$ where $\lambda_1 = 0$. Because the only solution in P_{λ_1} is the one in which $\boldsymbol{x}^*(0) = [1, 1, ..., 1]$ and this defines a consensus clustering with all objects in the same cluster, the first vector solution returned by CCA-OPPPA and CCA-ROPPPA will always define the consensus clustering with all objects in the same cluster.

2. Berkelaar et al. show that when the range λ (over which the parametric program algorithm is applied to $\bar{P}_\lambda$) is as large as possible (with $(\bar{P}_\lambda)$ still being nonempty) the final transition point λ_R returned by OPPPA will be the right end point of this range.
   For $\lambda \in [0, \binom{n}{2}]$ if we define $x_{ij} = \frac{\binom{n}{2}-\lambda}{\binom{n}{2}}$ $(1 \leq i < j \leq n)$ and $z_{ijk}^{(h)} = 1 - \frac{\binom{n}{2}-\lambda}{\binom{n}{2}}$ $(1 \leq i < j < k \leq n,\ h = 1, 2, 3)$, then $\bar{x} \in \bar{P}_\lambda$.
   If $\lambda > \binom{n}{2}$, then $\sum_{i<j} x_{ij} < 0$. But because $x_{ij} \geq 0$ for all $i > j$, then this implies that $(\bar{P}_\lambda)$ is empty for $\lambda > \binom{n}{2}$. Thus $\lambda = \binom{n}{2}$ is the right endpoint of the largest range over which the parametric program algorithm can be applied to $\bar{P}_\lambda$ (with $(\bar{P}_\lambda)$ still being nonempty). Thus the final transition point returned by OPPPA will be $\lambda_R = \binom{n}{2}$.
   Because the only solution in $P_{\binom{n}{2}}$ is the one in which $\boldsymbol{x}^*(\binom{n}{2}) = [0, 0, ..., 0]$ and this defines a consensus clustering with all objects in singleton clusters, the last vector solution returned by CCA-OPPPA and CCA-ROPPPA will always define the consensus clustering with all objects in singleton clusters.

3. Berkelaar et al. [Ber96] prove how the OPPPA yields $\{\lambda_1, \lambda_2, ..., \lambda_R\}$ which are all of the distinct transition points of $f(\lambda)$ for $\lambda \in [0, \binom{n}{2}]$ in order from smallest to largest. Because $\lambda_r \neq \lambda_s$ for all $r \neq s$, it follows that $\boldsymbol{x}^*(\lambda_r) \neq \boldsymbol{x}^*(\lambda_{r'})$ for all $1 \leq r < r' \leq R$.

The third result of the lemma above is important as the classical approach to parametric programming often returns duplicate solutions.

Although the first and last results returned by CCA-OPPPA and CCA-ROPPPA ($\boldsymbol{x}^*(\lambda_1)$ and $\boldsymbol{x}^*(\lambda_R)$) define the first and last results of a CCA-β solution (p^1 and p^R), we have no guarantee that the remaining results returned by CCA-OPPPA or CCA-ROPPPA will make it so that $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ defines a CCA-β solution. In chapter 6 we give two additional conditions, that if met, will guarantee that $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ defines a CCA-β solution.

Chapter 6

Structural Proofs

Because our CCA-OPPPA and CCA-ROPPPA heuristics are not guaranteed to return a CCA-β solutions for all input clustering sets $\{\pi^1, ..., \pi^T\}$, we apply CCA-ROPPPA on a series of input clusterings sets $\{\pi^1, ..., \pi^T\}$ to assess the frequency with which CCA-ROPPPA returned a CCA-β solution.

The task of assessing whether a set of nested consensus clusterings $\{p^1, ..., p^R\}$ defines a CCA-β solution for a given input clustering set $\{\pi^1, ..., \pi^T\}$ or not can be in and of itself a computationally burdensome. However, we show in this chapter that if the vector results returned by CCA-OPPPA or CCA-ROPPPA defines a set of nested consensus clusterings $\{p^1, ..., p^R\}$, then this nested set of consensus clusterings defines a CCA-β solution. Thus in our numerical tests in chapter 7, we need only to test if the set of vectors returned by CCA-ROPPPA define a set of nested consensus clusterings.

6.1 Lemmas

In order to prove this in theorem 7.5.1, we first must prove the following two lemmas.

Lemma 6.1.1 *Let $G(V, E)$ be a complete graph with edge weights $c : E \to R$. Let $\hat{\boldsymbol{x}}(\lambda_{r-1}) \in (P_{\lambda_{r-1}})$ and $\bar{\boldsymbol{x}}(\lambda_r) \in (P_{\lambda_r})$ define the partitions $p^{r-1} = \{U_1, ..., U_s\}$ and $p^r = \{V_1, ..., V_t\}$ of V respectively where p^r is nested in p^{r-1}.*

Then the following equation holds:

$$\beta_C(p^r|p^{r-1}) = \frac{\boldsymbol{c}'\hat{\boldsymbol{x}}(\lambda_{r-1}) - \boldsymbol{c}'\bar{\boldsymbol{x}}(\lambda_r)}{\boldsymbol{e}'\hat{\boldsymbol{x}}(\lambda_{r-1}) - \boldsymbol{e}'\bar{\boldsymbol{x}}(\lambda_r)} \tag{6.1}$$

Proof

$$
\begin{aligned}
\beta_C(p^r|p^{r-1}) = \quad & \beta_C(V_1, ..., V_t|U_1, ..., U_s) \\
&= \frac{|C(V_1,...,V_t)| - |C(U_1,...,U_s)|}{\sum\limits_{1\le q<q'\le t} |V_q||V_{q'}| - \sum\limits_{1\le w<w'\le s} |U_w||U_{w'}|} \\
&= \frac{\left(\sum\limits_{(i,j)\in C(V_1,...,V_t)} c_{ij}\right) - \left(\sum\limits_{(i,j)\in C(U_1,...,U_s)} c_{ij}\right)}{\left(\sum\limits_{(ij)\in C(V_1,...,V_t)} 1\right) - \left(\sum\limits_{(i,j)\in C(U_1,...,U_s)} 1\right)} && (1) \\
&= \frac{\left(\sum\limits_{(i,j)\in E:\bar{x}_{ij}(\lambda_r)=0} c_{ij}\right) - \left(\sum\limits_{(i,j)\in E:\hat{x}_{ij}(\lambda_{r-1})=0} c_{ij}\right)}{\left(\sum\limits_{(i,j)\in E:\bar{x}_{ij}(\lambda_r)=0} 1\right) - \left(\sum\limits_{(i,j)\in E:\hat{x}_{ij}(\lambda_{r-1})=0} 1\right)} && (2) \\
&= \frac{\left(\sum\limits_{(i,j)\in E} c_{i,j}(1-\bar{x}_{ij}(\lambda_r))\right) - \left(\sum\limits_{(i,j)\in E} c_{i,j}(1-\hat{x}_{ij}(\lambda_{r-1}))\right)}{\left(\sum\limits_{(i,j)\in E} (1-\bar{x}_{ij}(\lambda_r))\right) - \left(\sum\limits_{(i,j)\in E} (1-\hat{x}_{ij}(\lambda_{r-1}))\right)} && (3) \\
&= \frac{\mathbf{c}'\hat{\boldsymbol{x}}(\lambda_{r-1}) - \mathbf{c}'\bar{\boldsymbol{x}}(\lambda_r)}{\mathbf{e}'\hat{\boldsymbol{x}}(\lambda_{r-1}) - \mathbf{e}'\bar{\boldsymbol{x}}(\lambda_r)} && (4)
\end{aligned}
$$

In line 1, the numerator follows from the definition of a cut. The denominator of line 1 follows from the following argument.

Because $G(V, E)$ is a complete graph, the graph induced by $V_q \cup V_{q'}$ is also a complete graph. Because $V_q \cap V_{q'} = \phi$ and $V_q \cup V_{q'}$ is a complete graph, the graph induced by the the edges in E between V_q and $V_{q'}$ (or $\{(i, j) \in E | i \in V_q, j \in V_{q'}\}$) is a complete bipartite graph. And the number of edges in a complete bipartite graph (defined by the V_q and $V_{q'}$) is $|V_q||V_{q'}|$. Thus the following gives us the denominator of line 1:

$$
\begin{aligned}
\sum_{1\le q<q'\le t} |V_q||V_{q'}| \quad &= \sum_{1\le q<q'\le t} |\{(i,j) \in E | i \in V_q, j \in V_{q'}\}| \\
&= |\{(i,j) \in E | \exists q \neq q', i \in V_q, j \in V_{q'}\}| \\
&= |\{(i,j) \in C(V_1, ..., V_t)| \\
&= \sum_{(i,j)\in C(V_1,...,V_t)} 1
\end{aligned}
$$

Lemma 6.1.2 *Let f be a continuous concave function. If $\lambda_1, \lambda_2, \lambda_3$ is in the domain of f and $\lambda_1 < \lambda_2 < \lambda_3$, then*

$$
\frac{f(\lambda_2) - f(\lambda_1)}{\lambda_2 - \lambda_1} \ge \frac{f(\lambda_3) - f(\lambda_1)}{\lambda_3 - \lambda_1} \tag{6.2}
$$

Proof Since $\lambda_1 < \lambda_2 < \lambda_3$, there exists an $\alpha \in (0, 1)$ such that $\lambda_2 = \alpha\lambda_1 + (1 - \alpha)\lambda_3$. Since f is concave, it follows that

$$
\begin{aligned}
\frac{f(\lambda_2)-f(\lambda_1)}{\lambda_2-\lambda_1} &= \frac{f(\alpha\lambda_1+(1-\alpha)\lambda_3)-f(\lambda_1)}{\alpha\lambda_1+(1-\alpha)\lambda_3-\lambda_1} \\
&\geq \frac{\alpha f(\lambda_1)+(1-\alpha)f(\lambda_3)-f(\lambda_1)}{\alpha\lambda_1+(1-\alpha)\lambda_3-\lambda_1} \\
&= \frac{(1-\alpha)f(\lambda_3)-(1-\alpha)f(\lambda_1)}{(1-\alpha)\lambda_3-(1-\alpha)f\lambda_1} \\
&= \frac{f(\lambda_3)-f(\lambda_1)}{\lambda_3-\lambda_1}
\end{aligned}
$$

6.2 Theorem

Now we will prove that if the vector results returned by CCA-OPPPA or CCA-ROPPPA defines a set of nested consensus clusterings $\{p^1, ..., p^R\}$, then this nested set of consensus clusterings defines a CCA-β solution.

Theorem 6.2.1 *Let $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ be returned by CCA-OPPPA or CCA-ROPPPA, given the input clusterings $\{\pi^1, ..., \pi^T\}$. If $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ defines a set of nested consensus clusterings $\{p^1, ..., p^R\}$, then $\{p^1, ..., p^R\}$ is a CCA-β solution for $\{\pi^1, ..., \pi^T\}$.*

Proof

($\Rightarrow$)

Let $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ be returned by CCA-OPPPA or CCA-ROPPPA, given the input clusterings $\{\pi^1, ..., \pi^T\}$. Suppose that $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ defines a set of nested consensus clusterings $\{p^1, ..., p^R\}$ For $r = 2, ..., R$, we can define $p^{r-1} = \{U_1, ..., U_s\}$ and $p^r = \{V_1, ..., V_t\}$ respectively, where p^r is nested in p^{r-1}. Thus the β-ratio of $C(p^r)$ given $C(p^{r-1})$ is:

$$\beta_C(p^r|p^{r-1}) = \beta_C(V_1, ..., V_t|U_1, ..., U_s) = \frac{|C(V_1, ..., V_t)| - |C(U_1, ..., U_s)|}{\sum_{1\leq q<q'\leq t}|V_q||V_{q'}| - \sum_{1\leq w<w'\leq s}|U_w||U_{w'}|}. \tag{6.3}$$

.

We can then say:

$$
\begin{aligned}
\beta_C(p^r|p^{r-1}) = \quad & \beta_C(V_1,...,V_t|U_1,...,U_s) & \\
& = \frac{|C(V_1,...,V_t)|-|C(U_1,...,U_s)|}{\sum_{1\le q<q'\le t}|V_q||V_{q'}|-\sum_{1\le w<w'\le s}|U_w||U_{w'}|} & \\
& = \frac{\boldsymbol{c}'\boldsymbol{x}^*(\lambda_{r-1})-\boldsymbol{c}'\boldsymbol{x}^*(\lambda_r)}{\mathbf{e}'\boldsymbol{x}^*(\lambda_{r-1})-\mathbf{e}'\boldsymbol{x}^*(\lambda_r)} & \text{(by lemma 6.1.1)} \\
& = \frac{\boldsymbol{c}'\boldsymbol{x}^*(\lambda_{r-1})-\boldsymbol{c}'\boldsymbol{x}^*(\lambda_r)}{(M-\lambda_{r-1})-(M-\lambda_r)} & \\
& = -\frac{\boldsymbol{c}'\boldsymbol{x}^*(\lambda_{r-1})-\boldsymbol{c}'\boldsymbol{x}^*(\lambda_r)}{\lambda_{r-1}-\lambda_r} & \\
& = -\frac{f(\lambda_{r-1})-f(\lambda_r)}{\lambda_{r-1}-\lambda_r} & \text{(definition of } f) \\
& = -\frac{f(\lambda_{r-1})-f(\lambda)}{\lambda_{r-1}-\lambda} & \forall\lambda\in(\lambda_{r-1},\lambda_r] \\
& \le -\frac{f(\lambda_{r-1})-f(\lambda)}{\lambda_{r-1}-\lambda} & \forall\lambda>\lambda_{r-1} \text{ (by lemma 6.1.2)} \\
& = -\frac{\boldsymbol{c}'\boldsymbol{x}^*(\lambda_{r-1})-\max\limits_{\boldsymbol{x}(\lambda)\in(P_\lambda)}\boldsymbol{c}'\boldsymbol{x}(\lambda)}{\lambda_{r-1}-\lambda} & \text{(definition of } f) \\
& = -\frac{\boldsymbol{c}'\boldsymbol{x}^*(\lambda_{r-1})+\min\limits_{\boldsymbol{x}(\lambda)\in(P_\lambda)}-\boldsymbol{c}'\boldsymbol{x}(\lambda)}{\lambda_{r-1}-\lambda} & \\
& = -\frac{\min\limits_{\boldsymbol{x}(\lambda)\in(P_\lambda)}\boldsymbol{c}'\boldsymbol{x}^*(\lambda_{r-1})+-\boldsymbol{c}'\boldsymbol{x}(\lambda)}{\lambda_{r-1}-\lambda} & \\
& = -\max_{\boldsymbol{x}(\lambda)\in(P_\lambda)}\frac{\boldsymbol{c}'\boldsymbol{x}^*(\lambda_{r-1})+-\boldsymbol{c}'\boldsymbol{x}(\lambda)}{\lambda_{r-1}-\lambda} & \text{(because } \lambda_{r-1}-\lambda<0) \\
& = \min_{\boldsymbol{x}(\lambda)\in(P_\lambda)}\frac{\boldsymbol{c}'\boldsymbol{x}^*(\lambda_{r-1})-\boldsymbol{c}'\boldsymbol{x}(\lambda)}{-(\lambda_{r-1}-\lambda)} & \\
& = \min_{\boldsymbol{x}(\lambda)\in(P_\lambda)}\frac{\boldsymbol{c}'\boldsymbol{x}^*(\lambda_{r-1})-\boldsymbol{c}'\boldsymbol{x}(\lambda)}{(M-\lambda_{r-1})-(M-\lambda)} & \\
& = \min_{\boldsymbol{x}(\lambda)\in(P_\lambda)}\frac{\boldsymbol{c}'\boldsymbol{x}^*(\lambda_{r-1})-\boldsymbol{c}'\boldsymbol{x}(\lambda)}{\boldsymbol{e}'\boldsymbol{x}^*(\lambda_{r-1})-\boldsymbol{e}'\boldsymbol{x}(\lambda)} & \\
& \le \frac{\boldsymbol{c}'\boldsymbol{x}^*(\lambda_{r-1})-\boldsymbol{c}'\boldsymbol{x}(\lambda)}{\boldsymbol{e}'\boldsymbol{x}^*(\lambda_{r-1})-\boldsymbol{e}'\boldsymbol{x}(\lambda)} & \forall\boldsymbol{x}(\lambda)\in(P_\lambda),\lambda>\lambda_{r-1} \\
& \le \frac{\boldsymbol{c}'\boldsymbol{x}^*(\lambda_{r-1})-\boldsymbol{c}'\boldsymbol{x}(\lambda)}{\boldsymbol{e}'\boldsymbol{x}^*(\lambda_{r-1})-\boldsymbol{e}'\boldsymbol{x}(\lambda)} & \substack{\forall x(\lambda)\in(P_\lambda),\lambda>\lambda_{r-1},\, where \\ x(\lambda)\, defines\, a\, partition\, \hat{p}^r\, nested\, in\, p^{r-1}} \\
& = \beta_C(\hat{p}^r|p^{r-1}) & \text{for all } \hat{p}^r \text{ nested in } p^{r-1}
\end{aligned}
$$

Thus $\beta_C(p^r|p^{r-1})$ is the minimum β-ratio given $C(p^{r-1})$ for $r=2,...,R$. Thus if $\{\boldsymbol{x}^*(\lambda_1),...,\boldsymbol{x}^*(\lambda_R)\}$ is returned by CCA-OPPPA or CCA-ROPPPA for some input clustering set $\{\pi^1,...,\pi^T\}$ and defines a set of nested consensus clusterings $\{p^1,...,p^R\}$, then $\{p^1,...,p^R\}$ is a CCA-β solution for $\{\pi^1,...,\pi^T\}$.

Chapter 7

Numerical Testing

In this chapter we use numerical tests to a.) assess CCA-ROPPPAs ability to find a CCA-β solution and b.) compare CCA-ROPPPA to two other consensus clustering algorithms, which we call **CCA-GWLP** and **CCA-UPGMA** and describe in section 7.1. Specifically, we compare 1.) the average execution time of each of the three consensus clustering algorithms, 2.) the quality of the consensus clusterings returned by each of the consensus clustering algorithms, and 3.) the nature of the consensus clusterings returned by each of the consensus clustering algorithms.

Our numerical tests that we discuss in this chapter suggest that CCA-ROPPPA has the following benefits.

1. CCA-ROPPPA returned a CCA-β solution for 78.40% of the test input clustering sets.

2. In our numerical tests, on average 92.04% of the consecutive vectors pairs returned by CCA-ROPPPA ($\boldsymbol{x}^*(\lambda_{r-1})$, $\boldsymbol{x}^*(\lambda_r)$) defined two consensus clusterings (p^{r-1}, p^r) such that $\beta_C(p^r|p^{r-1})$ was the minimum β-ratio given $C(p^{r-1})$.

3. CCA-ROPPPA yields more options for consensus clustering solutions than CCA-GWLP and CCA-UPGMA.

4. CCA-ROPPPA outperforms CCA-GWLP in returning a consensus clustering with the desired or expected number of clusters.

5. CCA-ROPPPA outperforms CCA-UPGMA (especially when using real data sets) and CCA-GWLP when the researcher

   (a) first desires a consensus clustering where the number of clusters is as close as possible to the 'inherent' number of clusters and

(b) secondly desires the RAND score of the consensus clustering and the 'correct clustering' as large as possible.

Our numerical tests in this chapter suggest that CCA-ROPPPA has the following drawbacks.

1. CCA-UPGMA and CCA-GWLP on average have faster execution times than CCA-ROPPPA.

2. Unlike CCA-UPGMA, CCA-ROPPPA does not always return solutions that define consensus clusterings.

3. Unlike CCA-UPGMA, CCA-ROPPPA does not always return consensus clusterings that are nested in the preceding consensus clustering.

4. CCA-ROPPPA does not always return a CCA-β solution.

The input of each of these three consensus clustering algorithms is a set of input clusterings. There are several different input clustering set characteristics (or factors) that could affect a.) the likelihood that CCA-ROPPPA returns a CCA-β solution, b.) the average execution time of each of the three consensus clustering algorithms, c.) the quality of the consensus clusterings returned by each of the consensus clustering algorithms, and d.) the nature of the consensus clusterings returned by each of the consensus clustering algorithms. We list six of these factors in the box below.

In section 7.2 we show how we construct 648 sets of input clusterings that take into account these factors. We use each of these 648 sets of input clusterings as input for CCA-ROPPPA, CCA-GWLP, and CCA-UPGMA. The results returned by CCA-ROPPPA, CCA-GWLP, and CCA-UPGMA for each of these 648 sets of input clusterings are graphically displayed in the appendix.

In section 7.3 we discuss computer and software specifications used to a.) create each of these 648 sets of input clusterings and b.) execute CCA-ROPPPA, CCA-GWLP, and CCA-UPGMA.

In section 7.4 we discuss how to interpret the graphical displays of the results returned by CCA-ROPPPA, CCA-GWLP, and CCA-UPGMA that are located in the appendix. In section 7.5 we test the effectiveness of our CCA-ROPPPA heuristic, that is, we discuss CCA-ROPPPAs ability to return CCA-β solutions. Finally, in section 7.6 we compare the execution time, consensus clustering quality, and consensus clustering nature of CCA-ROPPPA, CCA-GWLP, and CCA-UPGMA.

The list of factors we wish to incorporate into our numerical tests are below.

Factor 1 The number of objects in $O = \{o_1, ..., o_n\}$.

Factor 2 The number of input clusterings in $\Pi = \{\pi^1, ..., \pi^T\}$.

Factor 3 The **average pairwise RAND score (ARS)** of the input clusterings in $\Pi = \{\pi^1, ..., \pi^T\}$, or [Ran71].

$$ARS = \frac{1}{\binom{T}{2}} \sum_{1 \leq t < t' \leq T} RAND(\pi^t, \pi^{t'}) \tag{7.1}$$

(Note RAND is defined in section 2.2).

Factor 4 The number of 'inherent' clusters in the set $O = \{o_1, ..., o_n\}$. (We will discuss what is meant by 'inherent' clusters in section 7.2.1).

Factor 5 How many input clusterings in $\Pi = \{\pi^1, ..., \pi^T\}$ have the 'inherent' number of clusters observed in the set $O = \{o_1, ..., o_n\}$.

Factor 6 The order of the objects in the coassociation matrices (further explained in section 7.2.3).

7.1 Algorithms Compared to CCA-ROPPPA

In this section we discuss the two consensus clustering algorithms which we compare to CCA-ROPPPA. We also discuss the motivation as to why we chose these consensus clustering algorithms.

Co-occurrence vs. Median Partition Problem Consensus Clustering In section 1.3 we describe that there are two main types of consensus clustering algorithms: 1). co-occurrence consensus clustering algorithms and 2.) consensus clustering algorithms which use the Median Partition Problem (MPP). The consensus clustering algorithm that we propose, CCA-β, is a co-occurrence consensus clustering algorithm. Therefore the first algorithm that we will compare CCA-β (and CCA-ROPPPA) to is a MPP consensus clustering algorithm. Specifically we will use the Mirkin distance to measure similarity between two partitions in the MPP, because that is the most commonly used metric in the literature. Additionally we will use the Grotschel-Wakabayashi BILP formulation of the MPP with Mirkin distance, which we call GWIP in this dissertation (section 1.4), to implement this algorithm because it will provide the optimal answer. We will also relax the binary constraint in GWIP. We call this LP formulation, **GWLP** (section 4.2).

Median Partition Problem Consensus Clustering Method (using Mirkin Distance) (CCA-GWLP)
Input: Let $\{\pi^1, ..., \pi^T\}$ be a set of input clusterings(partitions) created from the same set of objects $O = \{o_1, ..., o_n\}$ objects.
Output: A vector solution $\boldsymbol{x}^*$ that is the optimal solution of GWLP.

1. Create a $n \times n$ co-association matrix $\boldsymbol{W} = [w_{ij}]$, where

$$w_{ij} = \sum_{t=1}^{T} \epsilon(\pi^t(o_i), \pi^t(o_j)) \tag{7.2}$$

   $\epsilon(\pi^t(o_i), \pi^t(o_j)) = -1$ if o_i and o_j are assigned to the same cluster in π^t and $\epsilon(\pi^t(o_i), \pi^t(o_j)) = 1$ if o_i and o_j are assigned to different clusters in π^t.

2. Using $\boldsymbol{W} = [w_{ij}]$, as the objective function in GWLP (section 1.4), solve for the optimal solution x^* of GWLP using the Newton Barrier Method and the constraint addition routine described in 5.3.1.

Note that because CCA-GWLP uses GWLP, instead of GWIP, the optimal vector solution $\boldsymbol{x}^*$ that is returned is not necessarily binary, and thus does not necessarily define a consensus clustering. If $\boldsymbol{x}^*$ is binary, it does define a consensus clustering.

Agglomerative vs. Divisive Hierarchical Co-occurrence Consensus Clustering In section 2.3 we define divisive hierarchical clustering algorithms as clustering algorithms that begin with all objects in a clustering, then in each iteration at least one cluster is split into at least two clusters until all objects are in singleton clusters. An **agglomerative hierarchical clustering algorithm** is a clustering algorithm that begins with all objects in singleton clusters, then in each iteration exactly two clusters are joined into one cluster until all objects are in one cluster. CCA-β uses a *divisive* hierarchical clustering algorithm, where all objects begin in one cluster in the first partition p^1, then in each iteration a new partition p^r is formed by having at least one cluster from p^{r-1} split into at least two clusters. Specifically, the clusters in p^{r-1} are split in such a way that $C(p^r)$ is a minimum β-ratio cut given $C(p^{r-1})$.

Monti et al. (2003) [Mon03] use an agglomerative hierarchical clustering algorithm as the second step of a co-occurrence consensus clustering algorithm. Specifically they use average-linkage criteria (or the **UPGMA algorithm**) [SM58] to decide which two clusters to join in each iteration. Therefore we compare the use of their agglomerative average-linkage hierarchical clustering algorithm in the second step of a co-occurrence consensus clustering algorithm to CCA-β (CCA-ROPPPA).

Agglomerative Average-Linkage Co-Occurence Consensus Clustering Method (CCA-UPGMA) [Mon03]
Input: Let $\{\pi^1, ..., \pi^T\}$ be a set of input clusterings(partitions) created from the same set of objects $O = \{o_1, ..., o_n\}$ objects.
Output: A set of nested consensus clusterings(partitions) $\{p^1, ..., p^R\}$ of $O = \{o_1, ..., o_n\}$.

1. Create a $n \times n$ co-association matrix $\boldsymbol{C} = [c_{ij}]$, where

$$c_{ij} = \sum_{t=1}^{T} \delta(\pi^t(o_i), \pi^t(o_j)) \tag{7.3}$$

   $\delta(\pi^t(o_i), \pi^t(o_j)) = 1$ if o_i and o_j are assigned to the same cluster in π^t and $\delta(\pi^t(o_i), \pi^t(o_j)) = 0$ if o_i and o_j are assigned to different clusters in π^t.

2. Use the agglomerative hierarchical clustering, using average-linkage criteria (or UPGMA), on the similarity matrix $\boldsymbol{S} = T - [c_{ij}]$ to recluster the objects.

Note that CCA-UPGMA will automatically return a set of nested consensus clusterings $\{p^1, ..., p^R\}$. CCA-ROPPPA (which is a heuristic for CCA-β) and CCA-GWLP (which is a heuristic for CCA-GWIP) do not have this guarantee because they are not guaranteed to return *binary* vector solutions which *define consensus clustering solutions.*

7.2 Creating Input for CCA-UPGMA, CCA-GWLP, and CCA-ROPPPA

In this section we describe the design of our experiment and the way we generate data for the experiment. In order to assess the effects that the six factors listed in the beginning of this chapter have on a.) CCA-ROPPPAs ability to find a CCA-β solution and b.) the execution time, consensus clustering nature, and consensus clustering quality of CCA-ROPPPA, CCA-GWLP, and CCA-UPGMA we must incorporate these six factors into our experimental design. In this chapter we describe each step of our experimental design process and indicate which of the six factors are taken into account in that step.

7.2.1 Data Sets

CCA-UPGMA, CCA-GWLP, and CCA-ROPPPA each use a set of input clusterings as input. To create a set of input clusterings we take a set n objects $O = \{o_1, ..., o_n\}$ and apply a clustering algorithm to this set of objects T times. Each execution of the clustering algorithm produces an input clustering π^t. The set of input clusterings is thus $\Pi = \{\pi^1, ..., \pi^T\}$. Because the main application of this thesis is focused on gene expression clustering, four of our sets of objects O (to be clustered) are gene expression data sets from the Broad Institute [Pro15] described in table 7.1. For each of these data sets, an object defines a tissue sample. Each tissue sample has F expression levels corresponding to F genes, thus the gene expression level of the f^{th} gene is the f^{th} attribute value of the gene/object. The gene expression data sets are also useful because they have a known 'inherent' number of clusters. For instance, the Acute Leukemia Tissue Sample data set is known to contain tissue samples of patients with three different types of Leukemia: Acute Myloid Leukemia (AML), Acute Lymphoblastic Leukemia B-cell (ALL-B), and Acute Lymphoblastic Leukemia T-call (ALL-T). Therefore, for the sake of this experiment we know *before clustering* that clustering the $n = 38$ objects into three clusters will produce the most useful clustering result. Each of the datasets have a known 'inherent' number of clusters which we denote with ξ.

The tissuemulti and dlbcl data sets each had more than 100 objects, and thus would be beyond the capabilities of CCA-ROPPPA to solve in a reasonable amount of time without employing techniques from the literature referenced in section 1.4. Because the purpose of this dissertation is to explore a.) the quality and nature of the results of CCA-ROPPPA and b.) the theoretical connection of between CCA-ROPPPA and the cluster-ratio cut we created smaller datasets by randomly sampling 50 objects from each data set.

Artifical data was also used. This artificial data attempted to control for a.) the number of objects in the data set b.) the number of 'inherent' clusters in the data set and c.) the 'clusterability' of the dataset.

We chose from a collection of artificial 2D clustering datasets on a github repository, Clustering-Benchmark [Bar15]. Most of the datasets come from clustering papers which we cite and show in figure 7.1 and are designed to test various components of clustering algorithms. The datasets we use from this project are all 2D (ie. each object only has two attributes) so we can visualize how close the inherent clusters are and how clusterable the data is. All of the artificial clustering data sets have more than 200 objects, a, so we create smaller artificial datasets by randomly sampling 50 objects from each data set we used in the Clustering-Benchmark repository. The five original artificial data sets and citations of their original sources are in the left column of figure 7.1. To the right of each of these plotted artificial data sets in table 7.1 is the 50 sampled objects we used in our analysis.

Note that we are not choosing our artificial clustering data sets to assess the ability of a *clustering algorithm*. Instead we are choosing our artificial clustering data sets in their ability to create different types of input clustering sets. These varying types of input clustering sets will then be used to assess the ability our *consensus clustering algorithm*. For instance, the clusters in 2d-4c-no4-50sample0 are more separated than the clusters in spiral-50sample0. Because of this, we predict that the k-means algorithm will produce more input clusterings that are the same when the original data set is 2d-4c-no4-50sample0, and thus the average pairwise RAND score (ARS) of a set of input clusterings created from 2d-4c-no4-50sample0 will be higher than that of the average pairwise RAND score (ARS) of a set of input clusterings created from spiral-50sample0 (factor 3). Another way of saying this is that 2d-4c-no4-50sample0 is more 'clusterable' than spiral-50sample0.

This part of our data generation process varies the following factors discussed in the beginning of this chapter.

Factor 1 First the data sets have a different number of objects n.

Factor 4 Second the data sets have a different number of 'inherent' clusters. That is, variations in ξ are accounted for.

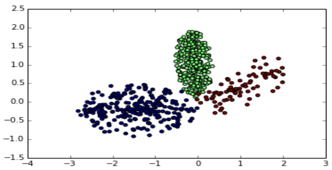
7.2.2 Create Partition Sets (Set of Input Clusterings)

Definition The inputs of CCA-UPGMA, CCA-GWLP, and CCA-ROPPPA are a set of T input clusterings created by clustering the set of objects $O = \{o_1, ..., o_n\}$ T times. We call this set of input clusterings $\{\pi^1, ..., \pi^T\}$ a **partition set** of size T of the data set O in our numerical tests.

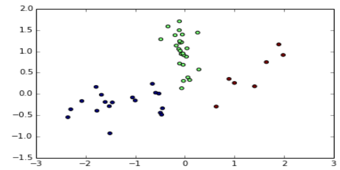
In this section we discuss how we create 36 partition sets for each of the 9 data sets O described in table 7.1.

For each of the 9 data sets, we used the k-means algorithm to cluster the data 156 times. Thus, we create 156 input clusterings for each data set O. The k-means algorithm allows for the user to preselect K, the desired number of clusters in the resulting clustering. For each of the 9 data sets O, we preselected the number of clusters to ask for, K, in the 156 input clusterings using the following procedure.

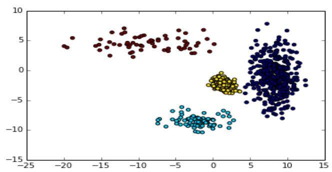
Input Clustering Trial 1 ($K = \xi - 1$) We create 39 input clusterings by executing the k-means algorithm 39 times on O, each time requesting $K = \xi - 1$ clusters (where ξ is



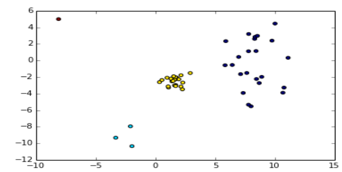
2d-3c-no123 [Han]



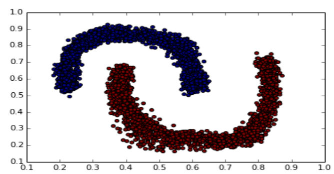
2d-3c-no123-50sample0



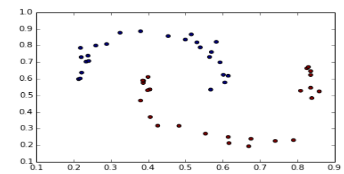
2d-4c-no4 [Han]



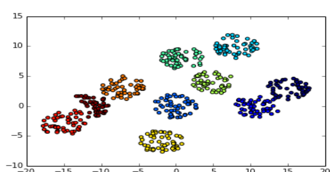
2d-4c-no4-50sample0



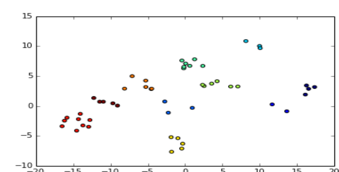
banana [Bar15]



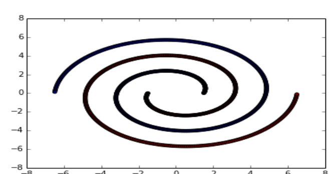
banana-50sample0



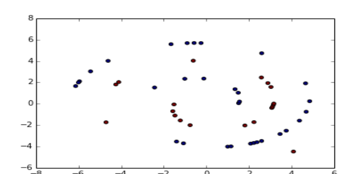
elliptical-10-2 [Ban] [BP10] [BM04] [BM01]



elliptical-10-2-50sample0



spiral [HK04]



spiral-50sample0

Figure 7.1 Artificial data sets and their random samples of 50 objects

Table 7.1 Real and artificial data sets

Data (O)	Number of Tissue Samples (Objects) in Data Set (n)	Number of Inherent Clusters in the Data Set (ξ)	Cluster Categories
Acute Leukemia Tissue Samples (leukemia)	38	3	Acute Myloid Leukemia (AML), Acute Lymphoblastic Leukemia B-Cell (ALL-B), Acute Lymphoblastic Leukemia T-Cell (ALL-T)
Brain Tumor Tissue Samples (brain)	42	5	Medulloblastoma tumor, Malignant Glioma tumor, Normal Cerebella, supratentorial primitive neuroectodermal tumor (PNET), atypical teratoid/rhabdoid tumor
Normal Tissue Samples (tissuemulti-50sample0)	50	4	Breast, Colon, Prostate, Lung
Diffuse Large B Cell Lymphoma Tissue Samples (dlbcl-50sample0)	50	3	(Subtypes: oxidative phosphorylation (OxPhos), B-cell response (BCR), and host response (HR))
2d-3c-no123-50sample0	50	3	(artificial data)
2d-4c-no4-50sample0	50	4	(artificial data)
banana-50sample0	50	2	(artificial data)
elliptical-10-2-50sample0	50	10	(artificial data)
spiral-50sample0	50	2	(artificial data)

the number of inherent clusters in O). These 39 input clusterings are placed in Input Clustering Trial Set 1.

Input Clustering Trial 2 ($K = \xi$) We then create 39 more input clusterings by executing the k-means algorithm 39 times on O, each time requesting $K = \xi$ clusters. These 39 input clusterings are placed in Input Clustering Trial Set 2.

Input Clustering Trial 3 ($K = \xi + 1$) We then create 39 more input clusterings by executing the k-means algorithm on O, each time requesting $K = \xi + 1$ clusters. These 39 input clusterings are placed in Input Clustering Trial Set 3.

Input Clustering Trial 4 ($K \in \{\xi - 1, \xi, \xi + 1\}$) Finally we create 39 more input clusterings by executing the k-means algorithm on O, each time randomly selecting K from the set $\{\xi - 1, \xi, \xi + 1\}$, where $P(K = \xi - 1) = P(K = \xi) = P(K = \xi + 1) = \frac{1}{3}$. Keeping the resulting input clusterings created in the order in which they were created, these 39 input clusterings are placed in Input Clustering Trial Set 4.

Then, from each input clustering trial set for each data set O, we create 9 partition sets as follows:

Partition Set Size (T=1) Create three partition sets with $T = 1$ input clustering each. We call them partition sets 1-A, 1-B, and 1-C.

Partition Set Size (T=4) Create three partition sets with $T = 4$ input clusterings each. We call them partition sets 4-A, 4-B, and 4-C.

Partition Set Size (T=8) Create three partition sets with 8 input clusterings each. We call them partition sets 8-A, 8-B, and 8-C.

Note that 39 input clusterings = (input clusterings of size 1) $\cdot$ 3 + (input clusterings of size 4) $\cdot$ 3 + (input clusterings of size 8) $\cdot$ 3.

Finally, to assess how similar the input clusterings in a given partition set are to one another, we use the average pairwise RAND index of the input clusterings in the partition set [Ran71]. That is, for each partition set $\Pi = \{\pi^1, ..., \pi^T\}$ we calculate:

$$ARS = \frac{1}{\binom{T}{2}} \sum_{1 \leq t < t' \leq T} RAND(\pi^t, \pi^{t'}) \tag{7.4}$$

This part of our data generation process varies the following factors discussed in the beginning of this chapter.

Factor 2 First we use different numbers of input clusterings in the partition sets.

Factor 5 Second the partitions sets created from each of the four Input Clustering Trial Sets vary in the number of clusters in the input clusterings. Therefore the factor of how many input clusterings have the 'inherent' number of clusters is accounted for.

Factor 3 Finally, the average pairwise RAND index of the input clusterings in the partition set is calculated and accounted for.

7.2.3 Create Permutation Sets and Coassociation Matrices

In step (1) of CCA-UPGMA and CCA-ROPPPA we create a coassociation matrix $C = [c_{ij}]$ where

$$c_{ij} = \sum_{t=1}^{T} \delta(\pi^t(o_i), \pi^t(o_j)) \tag{7.5}$$

$\delta(\pi^t(o_i), \pi^t(o_j)) = 1$ if o_i and o_j are assigned to the same cluster in π^t and $\delta(\pi^t(o_i), \pi^t(o_j)) = 0$ if o_i and o_j are assigned to different clusters in π^t. In step (1) of CCA-GWLP we create a coassociation matrix $W = [w_{ij}]$ where

$$w_{ij} = \sum_{t=1}^{T} \epsilon(\pi^t(o_i), \pi^t(o_j)) \tag{7.6}$$

$\delta(\pi^t(o_i), \pi^t(o_j)) = -1$ if o_i and o_j are assigned to the same cluster in π^t and $\epsilon(\pi^t(o_i), \pi^t(o_j)) = 1$ if o_i and o_j are assigned to different clusters in π^t.

Thus, the next step in our experimental design is for each partition set $\Pi = \{\pi^1, ..., \pi^T\}$ created, we create two coassociation matricies, which we call the **ordered coassocation matrix** and the **unordered coassociation matrix**.

Ordered Coassociation Matrix of Π

1. Let $O^{(1)}$ be the set of objects in O ordered according to their inherent clusters. That is, the first inherent cluster of objects in O are listed as objects $o_1^{(1)}, o_2^{(1)}, ..., o_{r'}^{(1)}$, the second inherent cluster of objects in O are listed as objects $o_{r'+1}^{(1)}, o_{r'+2}^{(1)}, ..., o_{r''}^{(1)}$, and so on.
2. Let $C^{(1)} = [c_{ij}]$, where i and j are the ith and jth objects listed in $O^{(1)}$.
3. Let $W^{(1)} = [w_{ij}]$, where i and j are the ith and jth objects listed in $O^{(1)}$.

Thus we call $C^{(1)}$ our 'ordered coassociation matrix' for CCA-UPGMA and CCA-ROPPPA and $W^{(1)}$ is our 'ordered coassociation matrix' for CCA-GWLP.

Unordered Coassociation Matrix of Π

1. Let $O^{(2)}$ be the set of objects in O in a randomly permuted order.
2. Let $C^{(2)} = [c_{ij}]$, where i and j are the ith and jth objects listed in $O^{(2)}$.
3. Let $W^{(2)} = [w_{ij}]$, where i and j are the ith and jth objects listed in $O^{(2)}$.

Thus we call $C^{(2)}$ our 'unordered coassociation matrix' for CCA-UPGMA and CCA-ROPPPA and $W^{(2)}$ is our 'unordered coassociation matrix' for CCA-GWLP.

Definition To help us clarify which ordering of objects in O was used to create the coassociation matrices in the three consensus clustering algorithms, we define a **permutation set** $\Pi^{(i)}$ of the partition set $\Pi = \{\pi^1, ..., \pi^T\}$ as the pairing of Π and $O^{(i)}$.

This part of our data generation process varies the following factors discussed in the beginning of this chapter.

Factor 6 The order of the objects in which the coassociation matrices were created from.

7.2.4 Use Each Coassociation Matrix as Input for the Three Consensus Clustering Algorithms

The procedure above described a methodology for creating input and executing step (1) of CCA-UPGMA, CCA-GWLP, and CCA-ROPPPA. That is, we created 648 different inputs (ie. 648 permutation sets) for each of these consensus clustering algorithms and specifically detailed two ways of constructing the co-association matrices listed in step (1) of each of these consensus clustering algorithms. To summarize, we have

$$\begin{aligned} 648 \text{ permutation sets} = \quad & (9 \text{ data sets}) \\ & \times(4 \text{ input clustering trials}) \\ & \times(9 \text{ partition sets}) \\ & \times(2 \text{ permutation set types}) \end{aligned}$$

Finally, we use these 648 permutation sets to execute step (2) of CCA-UPGMA, CCA-GWLP, and CCA-ROPPPA 648 times.

CCA-UPGMA will produce a series of nested consensus clusterings displayed in a dendrogram.

CCA-GWLP takes the coassocation matrix $\boldsymbol{W^{(i)}}$ from step (1) and translates this to a cost vector for the GWLP as $\boldsymbol{w^{(i)}} = [w_{12}^{(i)}, w_{13}^{(i)}, ..., w_{n-1,n}^{(i)}]$. CCA-GWLP will produce an optimal solution $\boldsymbol{x}^*$ of the GWLP that uses this cost vector. Again, if $\boldsymbol{x}^*$ is binary, it defines a consensus clustering p^* that is the optimal solution to the Median Partition Problem given the partition set Π.

CCA-ROPPPA takes the coassocation matrix $\boldsymbol{C^{(i)}}$ from step (1) and translates this to a cost vector for the P_λ as $\boldsymbol{c^{(i)}} = [c_{12}^{(i)}, c_{13}^{(i)}, ..., c_{n-1,n}^{(i)}]$. CCA-ROPPPA will produce a series of vector solutions $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ that are optimal solutions of P_{λ_1},...,P_{λ_R} respectively. Again, if a solution $\boldsymbol{x}^*(\lambda_r)$ is binary, it defines a consensus clustering. From theorem , if CCA-ROPPPA returns a set of nested consensus clusterings $\{p^1, ..., p^R\}$, then $\{p^1, ..., p^R\}$ is a CCA-β solution of $\{\pi^1, ..., \pi^R\}$.

We graphically display the results of CCA-UPGMA, CCA-GWLP, and CCA-ROPPPA for each of the 648 permutation sets in the appendix. Section 7.4 discusses how to interpret these graphical displays.

7.3 Computer, Software, and LP Solving

CCA-GWLP and CCA-ROPPPA were solved and coded in FICO Xpress-IVE. CCA-UPGMA was solved and coded using the scipy.cluster.hierarchy module in Python 2.7. All permutation sets and coassociation matrices were also generated using Python 2.7. All FICO Xpress-IVE and Python 2.7 code was run on an Intel Core 2 Duo CPU with 2.20 GHz with 4.0 Gbytes of RAM.

CCA-ROPPPA and CCA-GWLP involve solving one or more LPs. The Newton Barrier method in FICO Xpress-IVE was used to solve all LPs in CCA-ROPPPA and CCA-GWLP. Furthermore, the presolve option in FICO Xpress-IVE was used to solve all LPs in CCA-ROPPPA and CCA-GWLP. In section 7.5.2 we solve CCA-ROPPPA without using the FICO Xpress-IVE presolve option to see if this made a difference in our results.

7.4 Interpreting Consensus Clustering Results

In the appendix, we graphically summarized the results of CCA-UPGMA, CCA-GWLP, and CCA-ROPPPA for each of the 648 permutation sets that we created and used as input. Each page of the appendix shows the results of CCA-UPGMA, CCA-GWLP, and CCA-ROPPPA using one of the permutation sets as input. Obviously this makes for too many pages of results to include in the body of this dissertation, so we refer the reader to the appendix located online for the visual summary of each test.

In this section, we describe how to interpret the graphical results of each of the three consensus clustering algorithms located in the appendix.

7.4.1 Permutation Set Names

For example, figure 7.2 visually summarizes the results of CCA-UPGMA, CCA-GWLP, and CCA-ROPPPA for the permutation set labeled as dlbcl-50sample0-trial-2-partition8A-perm1. The first part of the permutation set name (dlbcl-50sample0) indicates the original set of objects that was clustered one or more times with the k-means algorithm. The second part of the permutation set name (trial-2) indicates that the way the input clusterings (partitions) in the permutation set were generated. For instance, the input clusterings (partitions) in dlbcl-50sample0-trial-2-partition8A-perm1 were made using input clustering trial set 2 described in section 7.2.2. That is, because the dlbcl-50sample0 data set had $\xi = 3$ inherent clusters, each of the input clusterings in permutation set dlbcl-50sample0-trial-2-partition8A-perm1 was created by clustering the dlbcl-50sample0 data set with k-means and specifically requesting $\xi = 3$ clusters. The third part of the permutation set name (8A) means that there are 8 partitions in the permutation set. Finally the fourth part of the permutation set name (perm1) means that the coassociation matrices created from this permutation set were $C^{(1)}$ and $W^{(1)}$ (or were the 'ordered coassociation matrices) described in section 7.2.3.

7.4.2 Interpreting CCA-UPGMA Results

The first graph in figure 7.2 shows the results of CCA-UPGMA using permutation set dlbcl-50sample0-trial-2-partition8A-perm1. The consensus clustering results are displayed in a dendrogram, which is the most common way of displaying the results of the UPGMA algorithm. The labels of the n objects in O and the 'correct cluster label' of each object are shown on the horizontal axis of the dendrogram. The number on the horizontal axis refers to the object number, and the letter refers to the label of the correct cluster the object belongs to.

We note that in each iteration of UPGMA exactly two clusters are combined into one cluster until all objects are in the same cluster. Thus technically CCA-UPGMA will have n iterations and thus result in n partitions. However, consider the dendrogram in figure 7.2. In this case $n = 50$, however the dendrogram shown visibly shows only 11 distinct levels (interpreted as 11 distinct consensus clusterings). In this algorithm, all $n = 50$ objects began in singleton clusters (at the first level). Then, in the first iteration object $b - 0$ and $b - 4$ were joined to form a cluster (second level). In the next iteration (also at the second level) $(b-0, b-4)$ was joined with $b-7$ to form a cluster. In total, 39 iterations of cluster joins happen at this second level creating a consensus clustering with 11 clusters shown at the second level.

For the purpose of our analyses, we will only interpret consensus clusterings from the dendrogram that result from joinings at *distinct* levels in the dendrogram. Thus we interpret CCA-UPGMA as returning 11 consensus clusterings for dlbcl-50sample0-trial-2-partition8A-perm1, instead of 50.

7.4.3 Interpreting CCA-GWLP Results

The second graph in figure 7.2 shows the results of CCA-GWLP using permutation set dlbcl-50sample0-trial-2-partition8A-perm1. CCA-GWLP returns one vector solution $\boldsymbol{x}^*$ which can either be fractional or binary. We represent this vector solution in the following way.

The labels of the n objects in O and the 'correct cluster label' of each object are shown on the horizontal axis of the box. The number on the horizontal axis refers to the object number, and the letter refers to the label of the correct cluster the object belongs to.

If the vector solution $\boldsymbol{x}^*$ is fractional, we draw a horizontal line of x's on the graph. If the vector solution $\boldsymbol{x}^*$ is binary, it defines a consensus clustering. Therefore we draw a solid horizontal line for each cluster in the consensus clustering defined by $\boldsymbol{x}^*$. All objects on the horizontal axis that share the same horizontal line are in the same cluster in the consensus clustering defined by $\boldsymbol{x}^*$. For instance, the second graph in figure 7.2 shows that CCA-GWLP returned a consensus clustering with 4 clusters. The first horizontal line represents the first cluster, where object 15 is the only object in the first cluster. The third horizontal line represents the third cluster which is comprised of objects 2, 5, 6, 13, 17, 19, 21, 29, 30, 31, 39, and 45.

7.4.4 Interpreting CCA-ROPPPA Results

The third graph in figure 7.2 shows the results of CCA-ROPPPA using permutation set dlbcl-50sample0-trial-2-partition8A-perm1. CCA-ROPPPA returns a set of vector solutions $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ which can either be fractional or binary. We represent these vector solutions in the following way.

The labels of the n objects in O and the 'correct cluster label' of each object are shown on the horizontal axis's of the boxes. The number on the horizontal axis refers to the object number, and the letter refers to the label of the correct cluster the object belongs to.

The vertical axis's of the boxes indicates which solution in the returned set of vector solutions

$\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ that we are referring to. That is, 'Iter r' on the vertical axis of the graph refers to the vector solution $\boldsymbol{x}^*(\lambda_r)$. If the vector solution $\boldsymbol{x}^*(\lambda_r)$ is fractional, we draw a horizontal line of x's at 'Iter r' on the graph. If the vector solution $\boldsymbol{x}^*(\lambda_r)$ is binary, it defines a consensus clustering. Therefore at 'Iter r we draw a solid horizontal line for each cluster in the consensus clustering defined by $\boldsymbol{x}^*(\lambda_r)$. All objects on the horizontal axis that share the same horizontal line in 'Iter r' are in the same cluster in the consensus clustering defined by $\boldsymbol{x}^*(\lambda_r)$.

For $r = 2, ..., R$, if a.) $\boldsymbol{x}^*(\lambda_r)$ defines a consensus clustering, b.) $\boldsymbol{x}^*(\lambda_{r-1})$ defines a consensus clustering, and c.) the consensus clustering defined by $\boldsymbol{x}^*(\lambda_r)$ is nested in the consensus clustering defined by $\boldsymbol{x}^*(\lambda_{r-1})$, then the vector solutions $\boldsymbol{x}^*(\lambda_r)$ and $\boldsymbol{x}^*(\lambda_{r-1})$ will be represented in the same box. Note that the third graph in figure 7.2 has only one box. Thus these conditions are met for all $r = 2, ..., R$. Given our results from chapter 6, this means that if the results displayed in the third graph are all in the same box, then CCA-ROPPPA returned a CCA-β solution.

However, for each vector solution $\boldsymbol{x}^*(\lambda_r)$ in which one of the following situations happens, a new box is formed beginning at 'Iter r' along with a new set of horizontal axis labels that correspond to this box: a.) $\boldsymbol{x}^*(\lambda_r)$ is fractional, b.) $\boldsymbol{x}^*(\lambda_{r-1})$ is fractional, or c.) the consensus clustering defined by $\boldsymbol{x}^*(\lambda_r)$ is not nested in the consensus clustering defined by $\boldsymbol{x}^*(\lambda_{r-1})$.

Finally, the network graph in figure 7.2 displays the nested nature of any consensus clusterings returned by CCA-ROPPPA using permutation set dlbcl-50sample0-trial-2-partition8A-perm1. All vector solutions $\boldsymbol{x}^*(\lambda_r)$ in $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ that define a consensus clustering are drawn as node 'Iter r' in the graph. The number of clusters in the consensus clustering defined by $\boldsymbol{x}^*(\lambda_r)$ is also shown in node 'Iter r'. The nodes are placed in numerical order in a circle, beginning with node 1 and going counterclockwise. The arrows are drawn as follows.

In Flow Arrows For each node s (except for the first) an arrow is drawn to node s from another node t_{max} where t_{max} is the largest value of t in which a.) $1 \leq t < s$ and b.) the CCA-ROPPPA returned a consensus clustering corresponding to Iter s is nested in the consensus clustering corresponding to Iter t. If $t_{max}+1 = s$, this means that the consensus clustering defined by $\boldsymbol{x}^*(\lambda_s)$ is nested in the consensus clustering defined by $\boldsymbol{x}^*(\lambda_{s-1})$. In the third graph in figure 7.2, this would be shown by the consensus clusterings represented by $\boldsymbol{x}^*(\lambda_{s-1})$ and $\boldsymbol{x}^*(\lambda_s)$ being drawn in the same box. In the network graph, we draw a bold arrow if this is the case.

Out Flow Arrows For each node s (except for the last) an arrow is drawn from node s to another node t_{min} where t_{min} is the smallest value of t in which a.) $s < t \leq R$ and b.) the consensus clustering corresponding to Iter t is nested in the consensus clustering corresponding to Iter s. If $s+1 = t_{min}$, this means that the consensus clustering defined by $\boldsymbol{x}^*(\lambda_s)$ is nested in the consensus clustering defined by $\boldsymbol{x}^*(\lambda_{s+1})$. In the third graph in figure 7.2, this would be shown by the consensus clusterings represented by $\boldsymbol{x}^*(\lambda_{s+1})$ and $\boldsymbol{x}^*(\lambda_s)$ being drawn in the same box. In the network graph, we draw a bold arrow if this is the case.

7.4.5 Examples

For instance, the third graph in figure 7.2 shows that CCA-ROPPPA (using permutation set dlbcl-50sample0-trial-2-partition8A-perm1) returned a set of 11 vector solutions $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_{11})\}$. Since each row of the graph is not a row of x's, this means that each vector in $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_{11})\}$ is binary, which means that each vector defines a consensus clustering which we demonstrate with 11 rows of solid horizontal lines in the graph. Furthermore, because each row of the graph is in the same box, this indicates that for $r = 2, ..., 11$ the consensus clustering defined by $\boldsymbol{x}^*(\lambda_r)$ is nested in the consensus clustering defined by $\boldsymbol{x}^*(\lambda_{r-1})$. Therefore, the third graph in figure 7.2 shows us that CCA-ROPPPA when applied to dlbcl-50sample0-trial-2-partition8A-perm1 returned a CCA-β solution. Also the fact that all the arrows in the network graph are bold in figure 7.2 shows us that CCA-ROPPPA when applied to dlbcl-50sample0-trial-2-partition8A-perm1 returned a CCA-β-solution.

For another example, the third graph in figure 7.3 shows that CCA-ROPPPA (using another permutation set dlbcl-50sample0-trial-2-partition8C-perm1) returned a set of 18 vector solutions $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_{18})\}$. The row of x's in iter 3 shows that $\boldsymbol{x}^*(\lambda_3)$ was a fractional solution and therefore did not define a consensus clustering. Therefore, the third graph in figure 7.3 shows us that CCA-ROPPPA when applied to dlbcl-50sample0-trial-2-partition8C-perm1 did not return a CCA-β solution. The fact that not all the arrows in the network graph in figure 7.3 are bold also shows us that CCA-ROPPPA when applied to dlbcl-50sample0-trial-2-partition8C-perm1 did not return a CCA-β solution.

For one final example, the third graph in figure 7.4 shows that CCA-ROPPPA (using another permutation set dlbcl-50sample0-trial-2-partition8B-perm2) returned a set of 11 vector solutions $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_{11})\}$. Since each row of the graph is not a row of x's, this means that each vector in $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_{11})\}$ is binary, which means that each vector defines a consensus clustering which we demonstrate with 11 rows of solid horizontal lines in the graph. However, because a new box is created at 'iter 8', this tells us that the consensus clustering defined by $\boldsymbol{x}^*(\lambda_8)$ is not nested in the consensus clustering defined by $\boldsymbol{x}^*(\lambda_7)$. Therefore, the third graph in figure 7.4 shows us that CCA-ROPPPA when applied to dlbcl-50sample0-trial-2-partition8Bperm2 did not return a CCA-β solution. The fact that not all the arrows in the network graph in figure 7.4 are bold also shows us that CCA-ROPPPA when applied to dlbcl-50sample0-trial-2-partition8B-perm2 did not return a CCA-β solution.

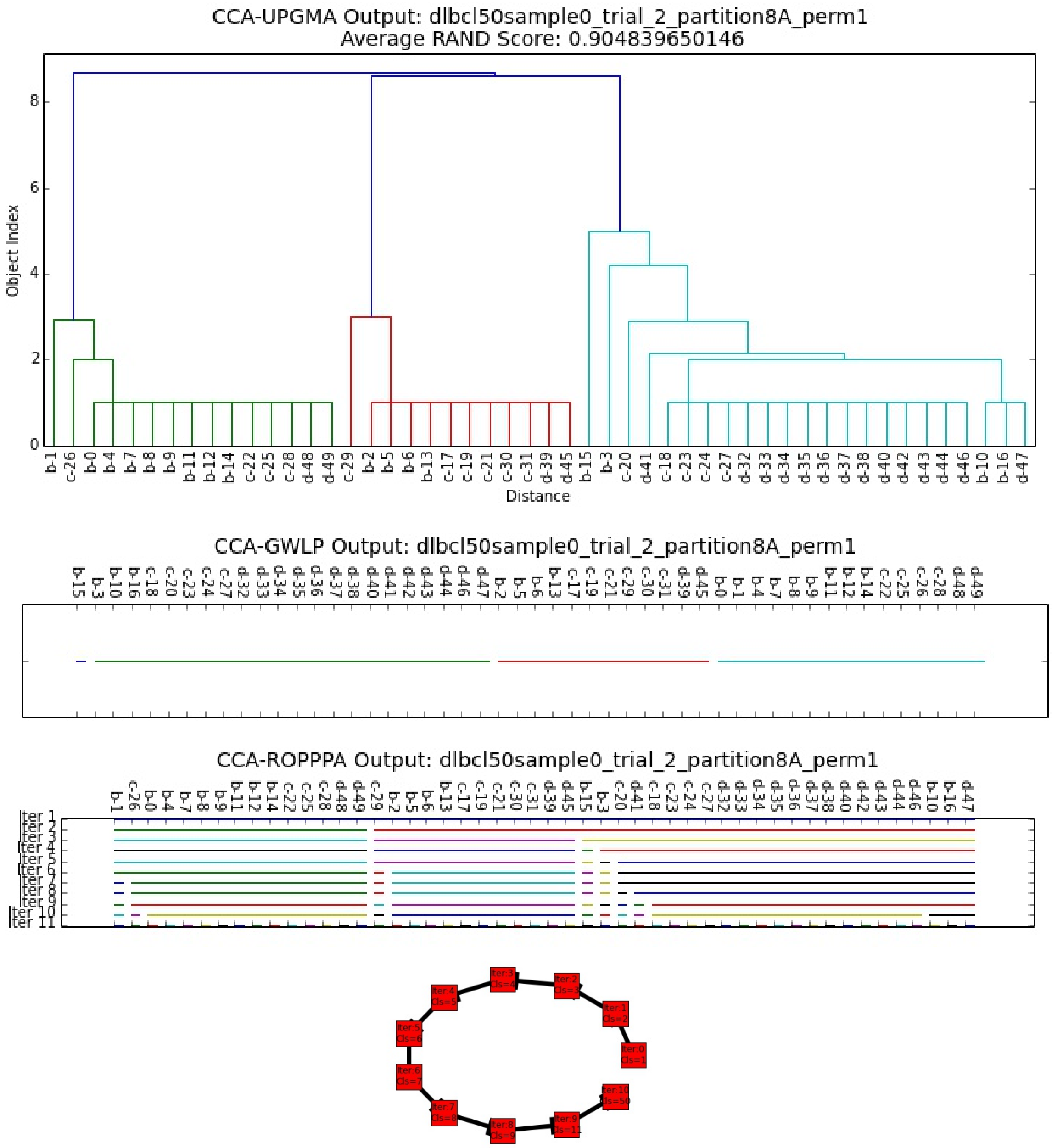


Figure 7.2 Results for dlbcl50sample0-trial-2-partition8A

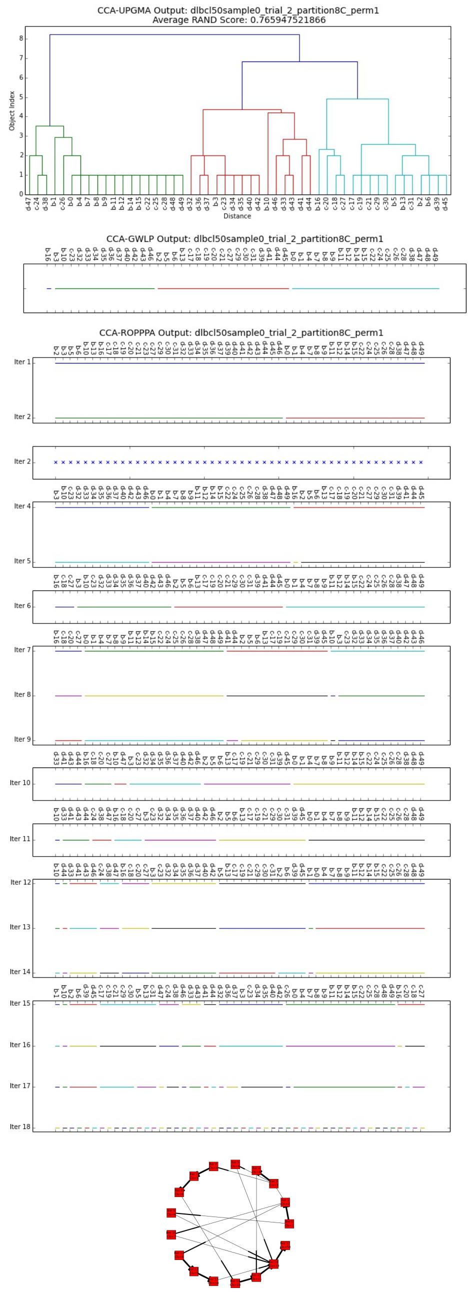


Figure 7.3 Results for dlbcl50sample0-trial-2-partition8A-perm1

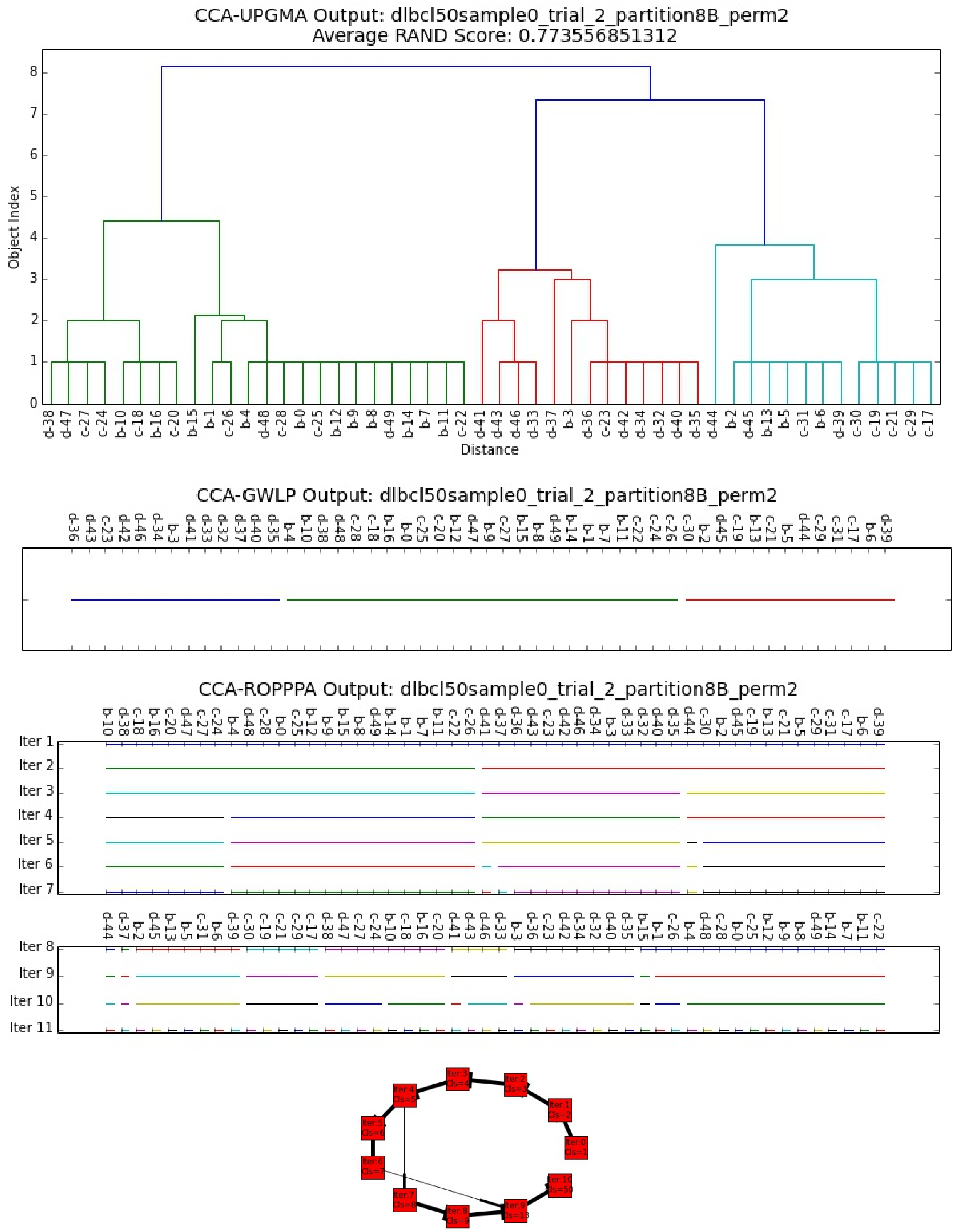


Figure 7.4 Results for dlbcl50sample0-trial-2-partition8-perm2

7.5 Testing when CCA-ROPPPA returns a CCA-β Solution

In section 7.5.1 we will summarize the frequency with which the 648 vector sets returned by CCA-ROPPPA defined a CCA-β solution when using the presolve option in the Xpress-IVE LP solver. We will also run CCA-ROPPPA in the Xpress-IVE software package without using presolve capabilties to test if this affects the results of CCA-ROPPPA, specifically whether CCA-ROPPPA returns a CCA-β solution or not. We summarize these results in section 7.5.2.

As shown in theorem , when CCA-ROPPPA returns a set of nested consensus clusterings, this set of nested consensus clusterings is a CCA-β solution for the given $\{\pi^1, ..., \pi^T\}$. To help us test and discuss how often this happens in our numerical tests, we defining the following conditions.

Definition Let $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ be the set of vector solutions returned by CCA-ROPPPA. We define the following conditions below:

Condition A $\boldsymbol{x}^*(\lambda_r)$ defines a consensus clustering p^r (is binary) for $r = 1, ..., R$.

Condition B $\boldsymbol{x}^*(\lambda_r)$ defines a consensus clustering p^r (is binary) for $r = 1, ..., R$ AND p^r is nested in p^{r-1} for $r = 2, ..., R$.

Note that if condition B is satisfied, then condition A is satisfied. Also note that if condition B is satisfied, then CCA-ROPPPA returned a CCA-β solution.

7.5.1 CCA-ROPPPA using Presolve in Xpress-IVE

In this section we examine the 648 vectors sets returned by CCA-ROPPPA when presolve was used in Xpress-IVE. Specifically, we examine the following metrics:

- the percent of the 648 vector sets $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ returned by CCA-ROPPPA that defined CCA-β solutions (or satisfied conditions A and B)

- the percent of the 648 vector sets $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ returned by CCA-ROPPPA that had EACH vector in the set define a consensus clustering (or satisfied condition A)

- average percent of vectors $\boldsymbol{x}^*(\lambda_r)$ in each of the the 648 vector sets $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ returned by CCA-ROPPPA that defined a consensus clustering

- the average percent of consecutive vector pairs in each of the the 648 vector sets $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ returned by CCA-ROPPPA in which both vectors in the pair define a consensus clustering where the second is nested in the first.

Finally, in this section we examine if any of the 6 factors listed in the beginning of chapter 7 affected the four metrics described above.

Table 7.2 shows that when presolve was used in Xpress-IVE, CCA-ROPPPA returned a set of vector solutions that satisfied conditions A and B for 78.40% of the 648 permutation sets that were used as input for CCA-ROPPPA. By theorem , this is equivalent to saying that CCA-ROPPPA returned a CCA-β solution for 78.40% of the 648 permutation sets that were used as input for CCA-ROPPPA. Table 7.2 also shows that when presolve was used in Xpress-IVE, CCA-ROPPPA returned a set of vector solutions that satisfied condition A for 89.20% of the 648 permutation sets that were used as input for CCA-ROPPPA. This is equivalent to saying that CCA-ROPPPA returned a set of vectors in which each vector defined a consensus clustering for 89.20% of the 648 permutation sets that were used as input for CCA-ROPPPA.

Despite the fact that CCA-ROPPPA returned a CCA-β solution for only 78.40% of the 648 permutation sets, table 7.2 also shows that on average a high percent (98.37%) of the vector solutions $\boldsymbol{x}^*(\lambda_r)$ in the returned set of vector solutions $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ defined consensus clusterings. Furthermore, table 7.2 also shows that on average a high percent (92.04%) of the consecutive pairs of vector solutions $(\boldsymbol{x}^*(\lambda_r), \boldsymbol{x}^*(\lambda_{r+1}))$ in the returned set of vector solutions $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ defined two consensus clusterings in which the second was nested in the first. Thus by theorem , on average 92.04% of the consecutive pairs of vector solutions $(\boldsymbol{x}^*(\lambda_r), \boldsymbol{x}^*(\lambda_{r+1}))$ in the returned set of vector solutions $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ defined two consensus clusterings p^r and p^{r+1} in which $C(p^r)$ was a β-cut given $C(p^{r+1})$.

Table 7.3 shows that CCA-ROPPPA always returned a CCA-β solution when the permutation sets were generated from three of the five artificial data sets: 2d-4c-no4-50sample0, elliptical-10-2-50sample0, and spiral-50sample0. The other two artificial data sets, 2d-3c-no123-50sample0 and banana-50sample0, had CCA-ROPPPA return a CCA-β solution when applied to 94.44% and 98.61% of the permutation sets generated from them respectively. When permutation sets were generated from the real data sets, CCA-ROPPPA returned CCA-β solutions with lower frequency.

This shows that CCA-ROPPPAs ability to return CCA-β solutions may be affected by the original data set that is used to create the permutation set that CCA-ROPPPA uses as input (factors 1 and 4).

Table 7.4 shows that CCA-ROPPPA always returned a CCA-β solution when the permu-

Table 7.2 CCA-ROPPPA using Presolve in Xpress-IVE: Summary Statistics Total

	Mean (Std) ARS of the Permutation Set	Percent of Vector Sets that Defined CCA-β Solutions (Satisfied Conditions A and B)	Percent of Vector Sets that Satisfied Condition A	Mean (Std) Percent of Vectors in Each Vector Set that Defined Consensus Clusterings	Mean (Std) Percent of Consecutive Pairs in Each Vector Set that Defined Consensus Clusterings with the Second Nested in the First
All Permutation Sets	0.9396 (0.0988)	78.40%	89.20%	98.37% (5.46%)	92.04% (17.17%)

tation set had only one partition (input clustering). In addition, CCA-ROPPPA returned a CCA-β solution in 70.37% and 64.81% of permutation sets that had 4 and 8 partitions respectively. This show that the number partitions in the permutation may affect CCA-ROPPPAs ability to return a CCA-β solution (factor 2).

Table 7.5 shows that CCA-ROPPPA returned CCA-β solution most often when the permutation set was generated in input clustering trial set 1, followed by input clustering trial set 2, and finally by those generated in input clustering trial sets 3 and 4 (factor 5).

Table 7.6 shows that CCA-ROPPPA returned a CCA-β solution in 78.70% when the permutation set was ordered vs. 78.09% when the permutation set was unordered. This indicates that the order of the permutation set did not seem to affect CCA-ROPPPAs ability to return CCA-β solutions (factor 6).

Finally, figure 7.5 shows that, in general, as the minimum ARS of a set of permutation sets increased, the percent of vector sets that CCA-ROPPPA returned (using these permutation sets as input) that defined CCA-β solutions increased (factor 3).

7.5.2 CCA-ROPPPA without using Presolve in Xpress-IVE

In this section we compare the 648 vector sets returned by CCA-ROPPPA when presolve was used in Xpress-IVE and the 648 vector sets returned by CCA-ROPPPA when presolve was NOT used in Xpress-IVE. Specifically, we use the following metrics.

Table 7.3 CCA-ROPPPA using Presolve in Xpress-IVE: Summary Statistics by Original Data

Data Set the Permutation Set Originated From	Mean (Std) ARS of the Permutation Set	Percent of Vector Sets that Defined CCA-β Solutions (Satisfied Conditions A and B)	Percent of Vector Sets that Satisfied Condition A	Mean (Std) Percent of Vectors in Each Vector Set that Defined Consensus Clusterings	Mean (Std) Percent of Consecutive Pairs in Each Vector Set that Defined Consensus Clusterings with the Second Nested in the First
2d-3c-no123-50sample0	0.9774 (0.0523)	94.44%	100.00%	100.00% (0.00%)	98.41% (6.59%)
2d-4c-no4-50sample0	0.9866 (0.0239)	100.00%	100.00%	100.00% (0.00%)	100.00% (0.00%)
banana-50sample0	0.9364 (0.1395)	98.61%	98.61%	99.65% (2.95%)	99.54% (3.93%)
brain	0.8843 (0.0874)	33.33%	56.94%	92.77% (10.67%)	68.76% (25.45%)
dlbcl-50sample0	0.8658 (0.1054)	48.61%	66.67%	95.80% (7.46%)	78.92% (23.98%)
elliptical-10-2-50sample0	0.9982 (0.0031)	100.00%	100.00%	100.00% (0.00%)	100.00% (0.00%)
leukemia	0.9118 (0.1049)	75.00%	90.28%	98.16% (5.67%)	92.33% (15.17%)
spiral-50sample0	0.934 (0.1342)	100.00%	100.00%	100.00% (0.00%)	100.00% (0.00%)
tissuemulti-50sample0	0.9623 (0.0373)	55.56%	90.28%	98.97% (3.21%)	90.38% (12.69%)

Table 7.4 CCA-ROPPPA using Presolve in Xpress-IVE: Summary Statistics by Permutation Set Size

Number of Partitions in the Permutation Set	Mean (Std) ARS of the Permutation Set	Percent of Vector Sets that Defined CCA-β Solutions (Satisfied Conditions A and B)	Percent of Vector Sets that Satisfied Condition A	Mean (Std) Percent of Vectors in Each Vector Set that Defined Consensus Clusterings	Mean (Std) Percent of Consecutive Pairs in Each Vector Set that Defined Consensus Clusterings with the Second Nested in the First
1	1 (0)	100.00%	100.00%	100.00% (0.00%)	100.00% (0.00%)
4	0.9057 (0.1141)	70.37%	88.89%	97.91% (6.79%)	89.57% (19.13%)
8	0.9132 (0.1044)	64.81%	78.70%	97.21% (6.28%)	86.54% (20.55%)

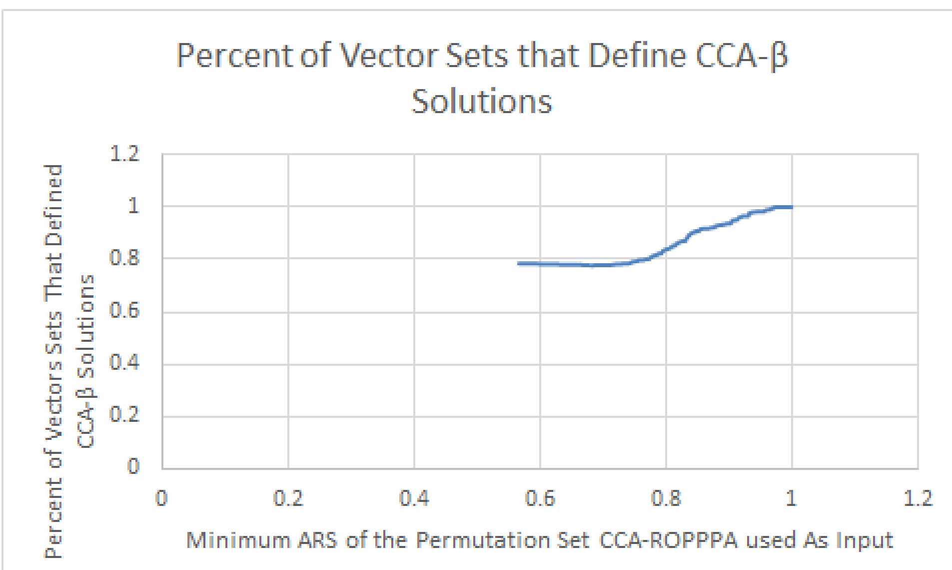


Figure 7.5 Percent of vector sets that defined CCA-β solutions by ARS.

Table 7.5 CCA-ROPPPA using Presolve in Xpress-IVE: Summary Statistics by Input Clustering Set Trial

Input Clustering Trial Set the Permutation Set was Made With	Mean (Std) ARS of the Permutation Set	Percent of Vector Sets that Defined CCA-β Solutions (Satisfied Conditions A and B)	Percent of Vector Sets that Satisfied Condition A	Mean (Std) Percent of Vectors in Each Vector Set that Defined Consensus Clusterings	Mean (Std) Percent of Consecutive Pairs in Each Vector Set that Defined Consensus Clusterings with the Second Nested in the First
1	0.9706 (0.0616)	87.04%	91.98%	98.88% (4.17%)	93.85% (16.79%)
2	0.9631 (0.0715)	82.10%	89.51%	98.03% (7.03%)	92.95% (17.23%)
3	0.9516 (0.0746)	72.22%	89.51%	98.81% (3.90%)	91.65% (15.41%)
4	0.8733 (0.1368)	72.22%	85.80%	97.77% (6.08%)	89.70% (18.99%)

Table 7.6 CCA-ROPPPA using Presolve in Xpress-IVE: Summary Statistics by Permutation Set Type

Permutation Set Type	Mean (Std) ARS of the Permutation Set	Percent of Vector Sets that Defined CCA-β Solutions (Satisfied Conditions A and B)	Percent of Vector Sets that Satisfied Condition A	Mean (Std) Percent of Vectors in Each Vector Set that Defined Consensus Clusterings	Mean (Std) Percent of Consecutive Pairs in Each Vector Set that Defined Consensus Clusterings with the Second Nested in the First
Ordered	0.9396 (0.0989)	78.70%	89.51%	98.46% (5.37%)	92.17% (17.13%)
Unordered	0.9396 (0.0989)	78.09%	88.89%	98.29% (5.56%)	91.90% (17.26%)

- We examine the percent of permutation sets that had different resulting vector sets returned by CCA-ROPPPA when presolve was and was not used.

- For each of these permutation sets that had different resulting vectors sets returned by CCA-ROPPPA when presolve was and was not used, we examine the average percent of vector pairs $(\boldsymbol{x}^*(\lambda_r), \bar{\boldsymbol{x}}^*(\lambda_r))$ that were different (where $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ and $\{\bar{\boldsymbol{x}}^*(\lambda_1), ..., \bar{\boldsymbol{x}}^*(\lambda_R)\}$ were returned by CCA-ROPPPA using the same permutation set and with and without using presolve respectively).

- We also examine heuristic performance of CCA-ROPPPA with and without presolve by examining the following metrics (also used in 7.5.1):

  - the percent of the 648 vector sets $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ returned by CCA-ROPPPA (with and without using presolve) that defined CCA-β solutions (or satisfied conditions A and B)
  - the percent of the 648 vector sets $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ returned by CCA-ROPPPA (with and without using presolve) that had EACH vector in the set define a consensus clustering (or satisfied condition A)

Table 7.7 CCA-ROPPPA Using and Not Using Presolve in Xpress-IVE: Summary Statistics

	Mean (Std) ARS of the Permutation Set	Percent of Permutation Sets that had Different Resulting Vector Sets Returned by CCA-ROPPPA when Presolve was and was not Used
All Permutation Sets	0.9396 (0.0988)	17.90%

- average percent of vectors $\boldsymbol{x}^*(\lambda_r)$ in each of the the 648 vector sets $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ returned by CCA-ROPPPA (with and without using presolve) that defined a consensus clustering
- the average percent of consecutive vector pairs in each of the the 648 vector sets $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ returned by CCA-ROPPPA (with and without using presolve) in which both vectors in the pair define a consensus clustering where the second is nested in the first.

Table 7.7 shows us that 17.90% of the 648 permutation sets had CCA-ROPPPA return a different set of results when presolve was and was not used. Of these 17.90% of permutation sets, it was the case that each of the two vector sets returned for by CCA-ROPPPA for the same permutation set had the same number of vectors (or $R = \bar{R}$). In addition, table 7.8 shows us that of these 17.90% of permutation sets, on average 14.41% of the vectors in the two vector sets returned were different (or 14.41% of the vectors $\boldsymbol{x}^*(\lambda_r)$ in $\{\boldsymbol{x}^*(\lambda_1), , \boldsymbol{x}^*(\lambda_R)\}$ had $\boldsymbol{x}^*(\lambda_r) \neq \bar{\boldsymbol{x}}^*(\lambda_r)$).

Table 7.9 summarizes CCA-ROPPPAs heuristic performance when presolve was and was not used in Xpress-IVE. All four metrics shows that CCA-ROPPPA performed slightly better when presolve was used in Xpress-IVE.

In addition, tables 7.10-7.13 suggest that the original data set the permutation set was generated from, the size of the permutation set, and the input clustering trial set the permutation set was generated from may affect the chance that CCA-ROPPPA returns different solutions with and without presolve.

Table 7.8 Different resulting vectors sets returned by CCA-ROPPPA when presolve was and was not used: Summary Statistics

	Mean (Std) ARS of the Permutation Set	Average Percent of the vectors $\boldsymbol{x}^*(\lambda_r)$ in $\{\boldsymbol{x}^*(\lambda_1), , \boldsymbol{x}^*(\lambda_R)\}$ had $\boldsymbol{x}^*(\lambda_r) \neq \bar{\boldsymbol{x}}^*(\lambda_r)$
Permutation Sets that had Different Resulting Vector Sets Returned by CCA-ROPPPA when Presolve was and was not Used	0.8299 (0.0700)	14.41%

Table 7.9 CCA-ROPPPA Using and Not Using Presolve in Xpress-IVE: Heuristic Performance Summary Statistics

	Mean (Std) ARS of the Permutation Set	Percent of Vector Sets that Defined CCA-β Solutions (Satisfied Conditions A and B)	Percent of Vector Sets that Satisfied Condition A	Mean (Std) Percent of Vectors in Each Vector Set that Defined Consensus Clusterings	Mean (Std) Percent of Consecutive Pairs in Each Vector Set that Defined Consensus Clusterings with the Second Nested in the First
All Permutation Sets (CCA-ROPPPA with Presolve)	0.9396 (0.0988)	78.40%	89.20%	98.37% (5.46%)	92.04% (17.17%)
All Permutation Sets (CCA-ROPPPA without Presolve)	0.9396 (0.0988)	77.16%	81.17%	96.57% (8.30%)	90.29% (19.59%)

Table 7.10 CCA-ROPPPA Using and Not Using Presolve in Xpress-IVE: Summary Statistics by Data

Data Set the Permutation Set Originated From	Mean (Std) ARS of the Permutation Set	Percent of Permutation Sets that had Different Resulting Vector Sets Returned by CCA-ROPPPA when Presolve was and was not Used
2d-3c-no123-50sample0	0.9774 (0.0523)	5.56%
2d-4c-no4-50sample0	0.9866 (0.0239)	0.00%
banana-50sample0	0.9364 (0.1395)	1.39%
brain	0.8843 (0.0874)	55.56%
dlbcl-50sample0	0.8658 (0.1054)	48.61%
elliptical-10-2-50sample0	0.9982 (0.0031)	0.00%
leukemia	0.9118 (0.1049)	15.28%
spiral-50sample0	0.934 (0.1342)	0.00%
tissuemulti-50sample0	0.9623 (0.0373)	34.72%

Table 7.11 CCA-ROPPPA Using and Not Using Presolve in Xpress-IVE: Summary Statistics by Partition Set Size

Number of Permutations in the Permutation Set	Mean (Std) ARS of the Permutation Set	Percent of Permutation Sets that had Different Resulting Vector Sets Returned by CCA-ROPPPA when Presolve was and was not Used
1	1 (0)	0.00%
4	0.9057 (0.1141)	26.85%
8	0.9132 (0.1044)	26.85%

Table 7.12 CCA-ROPPPA Using and Not Using Presolve in Xpress-IVE: Summary Statistics by Input Clustering Trial Set

Input Clustering Trial Set the Permutation Set was Made With	Mean (Std) ARS of the Permutation Set	Percent of Permutation Sets that had Different Resulting Vector Sets Returned by CCA-ROPPPA when Presolve was and was not Used
1	0.9706 (0.0616)	11.11%
2	0.9631 (0.0715)	12.96%
3	0.9516 (0.0746)	23.46%
4	0.8733 (0.1368)	24.07%

Table 7.13 CCA-ROPPPA Using and Not Using Presolve in Xpress-IVE: Summary Statistics by Permutation Set Type

Permutation Set Type	Mean (Std) ARS of the Permutation Set	Percent of Permutation Sets that had Different Resulting Vector Sets Returned by CCA-ROPPPA when Presolve was and was not Used
Ordered	0.9396 (0.0989)	17.90%
Unordered	0.9396 (0.0989)	17.90%

7.6 Comparing CCA-ROPPPA to CCA-GWLP and CCA-UPGMA

In this section we will compare CCA-ROPPPA to CCA-GWLP and CCA-UPGMA. In section 7.6.1 we will compare the execution time of the three consensus clustering algorithms. In section 7.6.2 we will compare the consensus clustering nature and quality of CCA-ROPPPA to CCA-GWLP. In section 7.6.3 we will compare the consensus clustering nature and quality of CCA-ROPPPA to CCA-UPGMA.

7.6.1 Execution Times

Table 7.14 summarizes the computation time of CCA-UPGMA, CCA-GWLP (with presolve), CCA-ROPPPA (with presolve) when using the 648 permutation sets as input. It shows that on average CCA-GWLP had the fastest computation time, followed by CCA-UPGMA, followed by CCA-ROPPPA.

7.6.2 Consensus Clustering Nature and Quality: CCA-ROPPPA and CCA-GWLP

In this section we compare the nature and quality of the consensus clusterings returned by CCA-ROPPPA to the nature and quality of the consensus clusterings returned by CCA-GWLP.

Specifically we assess the following:

Table 7.14 Summary statistics of CCA-UPGMA, CCA-GWLP, and CCA-ROPPPA computation times on the 648 permutation sets.

	Mean (Std) ARS of the Permutation Set	Mean (Std) CCA-UPGMA Time (s)	Mean (Std) CCA-GWLP Time (s)	Mean (Std) CCA-ROPPPA Time (s)
All Permutation Sets	0.9396 (0.0988)	0.2400 (0.3000)	0.1030 (0.0829)	11.7966 (14.3188)

Quality

- whether CCA-ROPPPA and CCA-GWLP returned a consensus clustering with the 'desired or expected' number of clusters
- the RAND score of the 'most desirable' consensus clustering returned by CCA-ROPPPA and the 'correct clustering'; and the RAND score of the 'most desirable' consensus clustering returned by CCA-GWLP and the 'correct clustering'

Nature

- the number of consensus clusterings returned by CCA-ROPPPA and CCA-GWLP
- the PERCENT of permutation sets in which CCA-GWLP returned a consensus clustering that was ONE of the consensus clusterings returned by CCA-ROPPPA
- the DIFFERENCES between the permutation sets in which CCA-GWLP did and did not return a consensus clustering that was ONE of the consensus clusterings returned by CCA-ROPPPA

First we discuss the number of consensus clusterings returned by CCA-ROPPPA and CCA-GWLP. Because CCA-GWLP is not guaranteed to return a binary vector solution, it is not guaranteed to return any consensus clusterings solutions. In addition, because CCA-GWLP involves solving only one LP formulation, at most one consensus clustering can be returned by CCA-GWLP. Our tests shows that all 648 vectors returned by CCA-GWLP were binary and thus defined a consensus clustering.

It is also the case that because CCA-ROPPPA is not guaranteed to return all binary vector solutions, it is not guaranteed to return results that all define consensus clusterings. However, unlike CCA-GWLP, CCA-ROPPPA can return multiple vectors, and thus its results can define

Table 7.15 Percent of Permutation Sets in which CCA-GWLP Returned a Consensus Clustering that was One of the Consensus Clusterings Returned by CCA-ROPPPA

	Permutation Sets in which CCA-GWLP Returned a Consensus Clustering that was ONE of the Consensus Clusterings returned by CCA-ROPPPA
All Permutation Sets	98.92%

more than one consensus clustering. Our results show that all 648 vectors sets returned by CCA-ROPPPA defined at least two consensus clustering solutions, specifically a.) the consensus clustering with all objects in the same cluster, and b.) the consensus clustering with all objects in singleton clusters. A table in the appendix gives the average number of consensus clusterings returned by CCA-ROPPPA, aggregated by the original data set, input clustering trial set, and partition set size.

Next, table 7.15 shows that 98.92% of the 648 permutation sets had CCA-GWLP return a consensus clustering that was ONE of the consensus clusterings returned by CCA-ROPPPA.

Next we compare CCA-GWLP to CCA-ROPPPA in its ability to produce useful consensus clustering results. Specifically, we discuss whether CCA-ROPPPA and CCA-GWLP returned a consensus clustering with the 'desired or expected' number of clusters. Because CCA-GWLP will always produce at most one consensus clustering, while CCA-ROPPPA can produce more than one consensus clustering, CCA-ROPPPA inherently has the ability to provide more options when it comes to choosing one or more consensus clusterings. One scenario in which this may be beneficial is when the consensus clustering returned by CCA-GWLP does not have the desired number of clusters, where ξ is the desired or 'inherent' number of clusters in the original data set. This may happen because CCA-GWLP does not allow the user to preselect the number of clusters that he or she wants in the resulting consensus clustering. One could also expect that the number of clusters in each input clustering in the partition set would affect the number clusters returned by CCA-GWLP. For instance, if each input clustering in the partition set had $\xi + 1$ clusters, then we would expect a consensus clustering of the partitions in the partition set to have $\xi + 1$ clusters.

For instance figure 7.6 shows the results of CCA-UPGMA, CCA-GWLP, and CCA-ROPPPA for the permutation set dlbcl-50sample0-trial-3-partition4C-perm1. The dataset dlbcl-50sample0 has $\xi = 3$ inherent (or desired) clusters, however because this permutation set is of trial 3, all of the input clusterings in the permutation set have $\xi + 1 = 4$ clusters. Therefore a consensus clustering partition with 3 or 4 clusters would be the desired or expected number of clusters respectively. However, figure 7.6 shows that CCA-GWLP returns a consensus clustering with 8 clusters for the permutation set dlbcl-50sample0-trial-3-partition4C-perm1. CCA-ROPPPA, on the other hand returns a consensus clustering with 3 clusters in iteration (3) and consensus clusterings with 4 clusters in iterations (4-8).

For CCA-GWLP, CCA-ROPPPA, and CCA-UPGMA we say that the consensus clustering algorithm returned a consensus clustering with 'desired' or 'expected' number of cluster if one of the following is true for at least one of the consensus clusterings returned by the consensus clustering algorithm.

- If the permutation set is of input clustering trial 1, and the consensus clustering algorithm returns a consensus clustering with $\xi - 1$ clusters or ξ clusters.
- If the permutation set is of input clustering trial 2, the consensus clustering algorithm returns a consensus clustering with ξ clusters.
- If the permutation set is of input clustering trial 3, the consensus clustering algorithm returns a consensus clustering with $\xi + 1$ clusters or ξ clusters.
- If the permutation set is of input clustering trial 4, the consensus clustering algorithm returns a consensus clustering with $\xi - 1$, ξ, or $\xi + 1$ clusters.

Table 7.16 shows that CCA-GWLP returned a consensus clustering with the desired or expected number of clusters for 566 (87.35%) of the 648 permutation sets, whereas CCA-ROPPPA returned a consensus clustering with the desired or expected number of clusterings for 647 (99.85%) of the 648 permutation sets. In addition, of the 82 permutation sets in which CCA-GWLP did not return a consensus clustering with the desired or expected number of clusters, 81 had CCA-ROPPPA return a consensus clsutering with the desired or expected number of clusters.

Therefore, this suggests that because CCA-ROPPPA may be more effective than CCA-GWLP at returning a consensus clustering with the desired or expected number of clusters. These results makes sense as CCA-ROPPPA has the opportunity to return more than one consensus clustering results, as opposed to CCA-GWLP which can only return at most one.

In addition, all of the permutation sets in which CCA-GWLP did not return the desired or expected number of clusters specifically returned MORE than the desired or expected number of

Table 7.16 Whether CCA-ROPPPA and CCA-GWLP Returned a Consensus Clustering with the Desired or Expected Number of Clusters

	CCA-GWLP Did Not Return a Partition with the Desired or Expected Number of Clusters	CCA-GWLP Returned a Partition with the Desired or Expected Number of Clusters	Total
CCA-ROPPPA Did Not Return a Partition with the Desired or Expected Number of Clusters	1	0	1
CCA-ROPPPA REtruned a Partition with the Desired or Expected Number of Clusters	81	566	647
Total	82	566	648

clusters. Table 7.17 shows what percentage of permutation sets returned more than the desired or expected number of clusters by the original data set the permutation set was generated from.

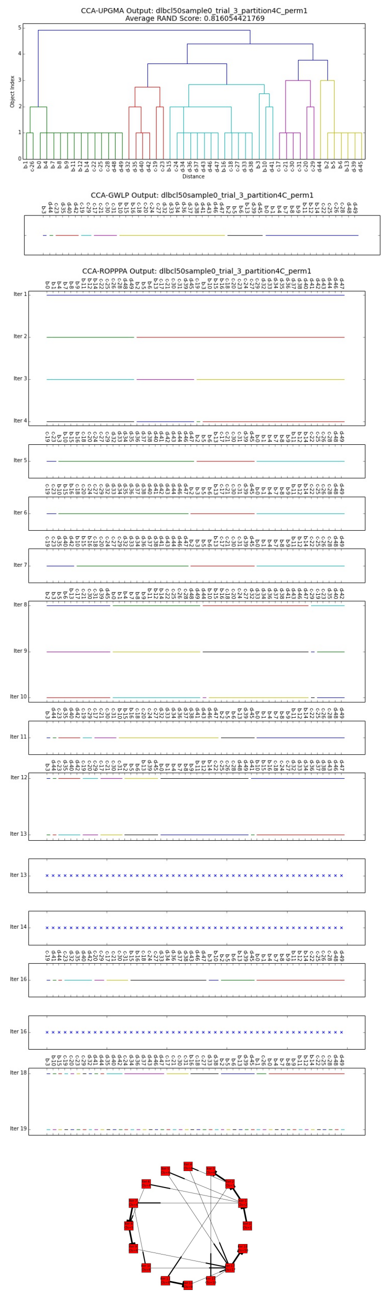


Figure 7.6 Results for dlbcl50sample0-trial-3-partition4C-perm1

Table 7.17 Percent of Permutation Sets from this Data Set in which CCA-GWLP Returned More than the Expected or Desired Number of Clusters

Data Set	Percent of Permutation Sets from this Data Set in which CCA-GWLP Returned More than the Expected or Desired Number of Clusters
2d-3c-no123-50sample0	2.78%
2d-4c-no4-50sample0	0%
banana-50sample0	0%
brain	55.56%
dlbcl-50sample0	25.00%
elliptical-10-2-50sample0	0%
leukemia	5.56%
spiral-50sample0	0%
tissuemulti-50sample0	25.00%

Next we assess the RAND score of the 'most desirable' consensus clustering returned by CCA-ROPPPA and the 'correct clustering' of O; and the RAND score of the 'most desirable' consensus clustering returned by CCA-GWLP and the 'correct clustering' of O. We first note that each of the original data sets O used in our analyses came with a correct clustering, which we will call $p_{correct}$. Because CCA-ROPPPA can return more than one consensus clustering, we assume that the researcher using CCA-ROPPPA would most desire a consensus clustering in which the number of clusters is closest to ξ (the number of clusters in $p_{correct}$). We denote ξ_{close} as a number of clusters in a consensus clustering returned by CCA-ROPPPA that is closest to ξ. Note, the ideal consensus clustering would have $\xi_{close} = \xi$. It follows that if there are multiple consensus clusterings with ξ_{close} clusters, the consensus clustering that has the highest RAND score with $p_{correct}$ would be most desirable to the researcher. We denote $RAND_{close}$ as the highest RAND score of a consensus clustering returned by CCA-ROPPPA with ξ_{close} clusters. Note, the ideal consensus clustering would have $\xi_{close} = \xi$ and $RAND_{close} = 1$.

With this in mind, we used the following method to select the most desirable returned consensus clustering from CCA-ROPPPA (CCA-UPGMA), which we call p_{close}.

Method for Selecting p_{close} in CCA-ROPPPA (CCA-UPGMA)

Input: Set of consensus clusterings of O returned by CCA-ROPPPA (CCA-UPGMA), $\{p_1, ..., p_R\}$, and the correct clustering of O, $p_{correct}$, which has ξ clusters.

Output: One of the consensus clusterings from $\{p_1, ..., p_R\}$, which we call p_{close}.

1. Set $p_{close} = p_1$, $\xi_{close} = 1$, and $RAND_{close} = RAND(p_1, p_{correct})$.
2. For $r = 2, ..., R$
   (a) Let ξ_r be the number of clusters in p_r.
   (b) If $|\xi - \xi_r| < |\xi - \xi_{close}|$ or ($|\xi - \xi_r| = |\xi - \xi_{close}|$ and $RAND(p_r, p_{correct}) > RAND(p_{close}, p_{correct})$), then:
      - $p_{close} = p_r$
      - $\xi_{close} = \xi_r$
      - $RAND_{close} = RAND(p_r, p_{correct})$

Becase CCA-GWLP returns at most one consensus clustering p, for CCA-GWLP we define the $p_{close} = p$. It follows that for CCA-GWLP we define $RAND_{close} = RAND(p_{close}, p_{correct})$ and ξ_{close} as the number of clusters in p.

Table 7.18 shows that 449 of the 648 permutation sets had $(\xi_{close}, RAND_{close})$ of CCA-GWLP and $(\xi_{close}, RAND_{close})$ of CCA-ROPPPA as the same. It shows that 168 of the 648 permutation sets had CCA-ROPPPA have a better ξ_{close} than that of CCA-GWLP, whereas only 1 of the 648 permutation sets had CCA-GWLP have a better ξ_{close} than that of CCA-ROPPPA.

Table 7.18 also shows that of the 479 permutation sets that had ξ_{close} of CCA-GWLP and ξ_{close} of CCA-ROPPPA as the same, 30 had $RAND_{close}$ of CCA-ROPPPA better than CCA-GWLP whereas 0 had $RAND_{close}$ of CCA-GWLP better than CCA-ROPPPA. Therefore, this shows that if the researcher first desires a ξ_{close} that is as close as possible to ξ and secondly desires $RAND_{close}$ as large as possible, then CCA-ROPPPA outperforms CCA-GWLP.

Finally, to further compare the nature of CCA-ROPPPA results to CCA-GWLP results, we look for differences in:

- the sets of permutation sets where CCA-GWLP returned a consensus clustering solution that was one of the consensus clusterings returned by CCA-ROPPPA
- the sets of permutation sets where CCA-GWLP did not return a consensus clustering solution that was one of the consensus clusterings returned by CCA-ROPPPA

Table 7.18 The RAND Score of the Most Desirable Consensus Clustering Returned by CCA-ROPPPA (CCA-GWLP) and the Correct Clustering

Permutation Sets	ξ_{close} of CCA-ROPPPA is Better Than That of CCA-GWLP	ξ_{close} of CCA-ROPPPA is Equal to That of CCA-GWLP	ξ_{close} of CCA-ROPPPA is Worse Than That of CCA-GWLP	Total
$RAND_{close}$ of CCA-ROPPPA is Better Than That of CCA-GWLP	59	30	0	89
$RAND_{close}$ of CCA-ROPPPA is Equal to That of CCA-GWLP	10	449	0	459
$RAND_{close}$ of CCA-ROPPPA is Worse Than That of CCA-GWLP	99	0	1	100
Total	168	479	1	648

Specifically we looked for differences in ARS, the original data set, permutation set sizes, permutation set type (1 or 2), input clustering set type.

Table 7.15 shows that only 1.08% (or 7) of the 648 permutation sets had CCA-GWLP not return a consensus clustering solution that was one of the consensus clusterings returned by CCA-ROPPPA. They were the following:

- dlbcl-50sample0-trial-4-partition4B-perm1
- dlbcl-50sample0-trial-4-partition4B-perm2
- brain-trial-1-partition8C-perm1
- brain-trial-1-partition8C-perm2
- tissuemulti-50sample0-trial-3-partition4C-perm1
- tissuemulti-50sample0-trial-3-partition4C-perm2
- banana-50sample0-trial-4-partition8B-perm2

Table 7.19 ARS Difference Between the Permutation Sets in which CCA-GWLP Did and Did Not Return a Consensus Clustering that was One of the Consensus Clusterings Returned by CCA-ROPPPA

Permutation Sets	Mean (Std) ARS of Permutation Sets	Number of Permutation Sets
That had CCA-GWLP not return a consensus clustering solution that was one of the consensus clusterings returned by CCA-ROPPPA	0.7752 (0.0903)	7
That had CCA-GWLP return a consensus clustering solution that was one of the consensus clusterings returned by CCA-ROPPPA	0.9414 (0.0975)	641

Because there are so few permutation sets that had CCA-GWLP not return a consensus clustering solution that was one of the consensus clusterings returned by CCA-ROPPPA, it is difficult to assess whether the original data set, the input clustering trial set, and the partition set size affected the likelihood of this result. We can note that all permutation sets containing exactly one input clustering had CCA-GWLP return a consensus clustering solution that was one of the consensus clusterings returned by CCA-ROPPPA. Also notice that 7 of these permutation sets corresponded to just 4 partition sets. Finally table 7.19 doesn't seem to indicate significant differences in ARS between the two sets of permutation sets.

7.6.3 Consensus Clustering Nature and Quality: CCA-ROPPPA and CCA-UPGMA

In this section we compare the nature and quality of the consensus clusterings returned by CCA-ROPPPA to the nature and quality of the consensus clusterings returned by CCA-UPGMA.

Specifically we assess the following:

Quality

- the number of consensus clusterings returned by CCA-ROPPPA and CCA-UPGMA that are nested in the preceding consensus clustering

- whether CCA-ROPPPA and CCA-UPGMA returned a consensus clustering with the 'desired or expected' number of clusters
- the RAND score of the 'most desirable' consensus clustering returned by CCA-ROPPPA and the 'correct clustering'; and the RAND score of the 'most desirable' consensus clustering returned by CCA-UPGMA and the 'correct clustering'

Nature

- the number of consensus clusterings returned by CCA-ROPPPA and CCA-UPGMA
- the PERCENT of permutation sets in which the results of CCA-UPGMA were different than those of CCA-ROPPPA
- the DIFFERENCES between the permutation sets in which the results of CCA-UPGMA were different than those of CCA-ROPPPA

First we discuss the number of consensus clusterings returned by CCA-ROPPPA and CCA-UPGMA.

Because CCA-UPGMA uses UPGMA (as opposed to solving LPs that need to return binary optimal solutions in order to represent a partition), CCA-UPGMA will return at least two consensus clusterings, specifically a.) one with all objects in the same cluster and b.) one with all objects in singleton clusters. Although CCA-ROPPPA is not guaranteed to return any consensus clusterings solutions, our results from section 7.6.2 show that all 648 vector sets returned by CCA-ROPPPA also defined at least two consensus clustering solutions, specifically a.) the consensus clustering with all objects in the same cluster, and b.) the consensus clustering with all objects in singleton clusters.

From our discussions in section 7.4 on how to interpret the results of CCA-UPGMA, CCA-UPGMA will return *at most* n consensus clusterings. Although, it is possible for CCA-ROPPPA to return more than n consensus clusterings, our results show that all 648 vector sets returned by CCA-ROPPPA also defined no more than n consensus clusterings.

Table 7.20 shows the average number of consensus clusterings returned by CCA-ROPPPA and CCA-UPGMA. It does not show a significant difference. However, the table in the appendix finds the average number of input clusterings returned by CCA-UPGMA and CCA-ROPPPA grouped by the original data set, the number of partitions in the partition set, and the input clustering trial set. It shows that CCA-ROPPPA returned the same number of consensus clusterings as CCA-UPGMA when the permutation set had exactly one partition. Because one third of our permutation sets have exactly one partition, we examined the data even more.

Table 7.20 The Number of Consensus Clusterings Returned by CCA-ROPPPA and CCA-UPGMA

	Mean (Std) Number of Consensus Clusterings Returned by CCA-ROPPPA	Mean (Std) Number of Consensus Clusterings Returned by CCA-UPGMA
All Permutation Sets	5.59 (4.66)	5.06 (3.48)

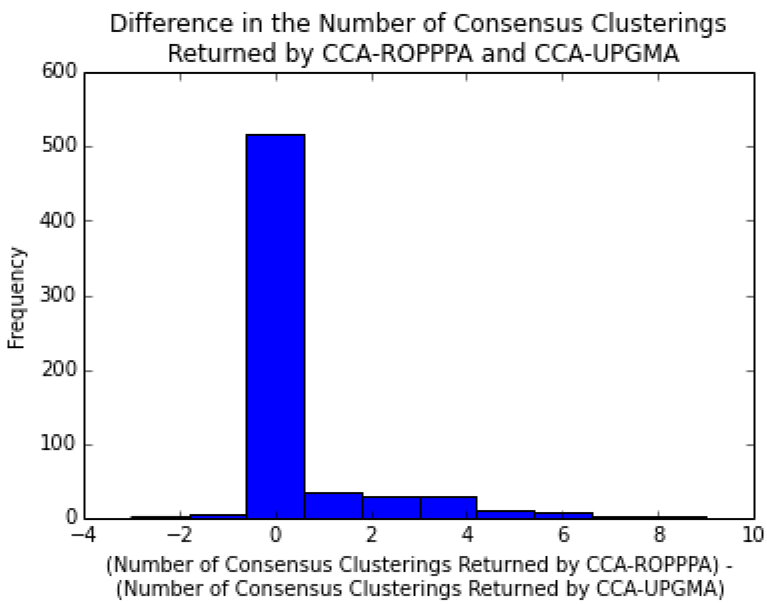


Figure 7.7 Difference in the Number of Consensus Clusterings Returned by CCA-ROPPPA and CCA-UPGMA

Figure 7.7 shows that most of the 648 permutation sets had CCA-ROPPPA and CCA-UPGMA returned the same number of consensus clusterings and that CCA-ROPPPA was more likely than CCA-UPGMA to return a higher number of consensus clusterings. Thus this indicates that CCA-ROPPPA may provide more consensus clustering options for the researcher than CCA-UPGMA.

Next we look at the PERCENT of permutation sets in which the results of CCA-UPGMA were different than those of CCA-ROPPPA. Table 7.21 shows that 77.01% of the 648 permutation sets had CCA-ROPPPA and CCA-UPGMA return the same set of consensus clusterings. It shows that 2.78% of permutation sets had the CCA-UPGMA consensus clusterings returned being a strict subset of CCA-ROPPPA consensus clusterings returned, whereas only 0.77% of

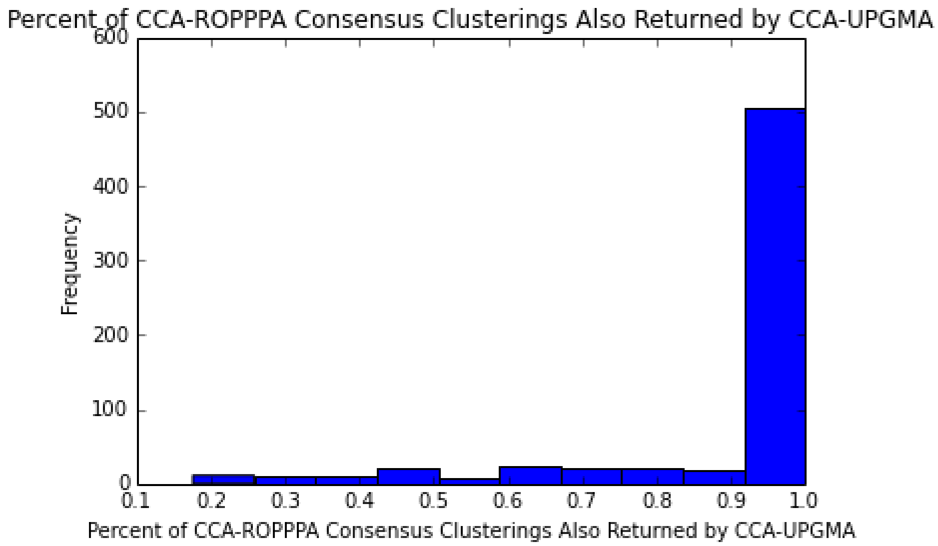


Figure 7.8 Percent of CCA-ROPPPA Consensus Clusterings Also Returned by CCA-UPGMA

permutation sets had the CCA-ROPPPA consensus clusterings returned being a strict subset of CCA-UPGMA consensus clusterings returned. Finally, it shows that 19.44% of permutation sets had at least one consensus clustering returned by CCA-UPGMA that was not returned by CCA-ROPPPA, and vice versa. Thus this shows that while CCA-ROPPPA is able to give more consensus clustering options than CCA-UPGMA, there are many cases where there is not an overlap (or subset) in results.

In addition, figures 7.8 and 7.9 show the histogram of the percent of returned consensus clusterings returned by CCA-ROPPPA that are also returned by CCA-UPGMA and the histogram of the percent of returned consensus clusterings returned by CCA-UPGMA that are also returned by CCA-ROPPPA. They show that the results of most permutation sets have high overlap in both consensus clustering algorithms, while some permutation sets had low overlap in both consensus clustering algorithms. Furthermore, figure 7.10 shows a positive correlation between the percent of CCA-ROPPPA consensus clusterings that were also CCA-UPGMA clusterings and the percent of CCA-UPGMA consensus clusterings that were also CCA-ROPPPA clusterings.

We will discuss differences in permutation sets that did and did not have high overlapping results at the end of this section.

Next we will discuss the number of consensus clusterings returned by CCA-ROPPPA and

Table 7.21 Percent of Permutation Sets in which the Results of CCA-UPGMA were Different than those of CCA-ROPPPA

Permutation Sets Where:	Number	Percent
Returned CCA-UPGMA Consensus Clusterings = Returned CCA-ROPPPA Consensus Clusterings	499	77.01%
Returned CCA-UPGMA Consensus Clusterings $\subset$ Returned CCA-ROPPPA Consensus Clustering	18	2.78%
CCA-ROPPPA Consensus Clusterings $\subset$ Returned CCA-UPGMA Consensus Clustering	5	0.77%
(CCA-ROPPPA Consensus Clusterings $\not\subset$ Returned CCA-UPGMA Consensus Clustering) and (CCA-UPGMA Consensus Clusterings $\not\subset$ Returned CCA-ROPPPA Consensus Clustering)	126	19.44%
Total	648	100.00%

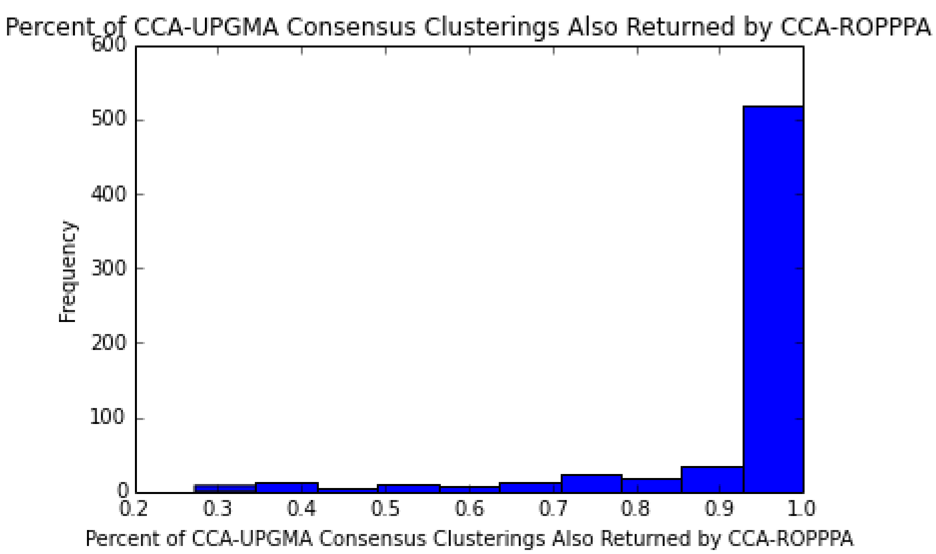


Figure 7.9 Percent of CCA-UPGMA Consensus Clusterings Also Returned by CCA-ROPPPA

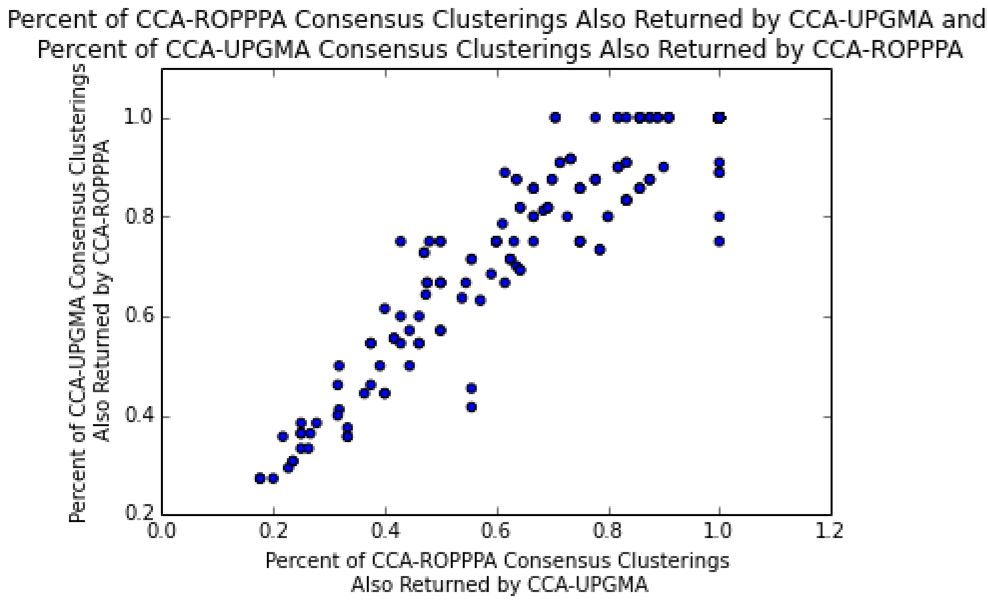


Figure 7.10 Percent of CCA-ROPPPA Consensus Clusterings Also Returned by CCA-UPGMA and Percent of CCA-UPGMA Consensus Clusterings Also Returned by CCA-ROPPPA

CCA-UPGMA that are nested in the preceding consensus clustering. Because CCA-UPGMA uses UPGMA, all consensus clusterings returned by CCA-UPGMA will be nested in the preceding consensus clustering. The consensus clustering algorithm CCA-β is also designed to return a set of consensus clusterings in which each consensus clustering is nested in the previous consensus clustering. However, because CCA-ROPPPA is a heuristic of CCA-β, this property is not guaranteed. As we discussed in section 7.5.1, table 7.2 shows that 78.40% of permutation sets had all vectors returned define a consensus clustering that was nested in the preceding consensus clustering and that the average permutation set had 92.04% of it's returned consecutive vector pairs define consensus clusterings in which the second was nested in the first. If it is the case that researcher requires all consensus clusterings to be nested in the previous one, then CCA-UPGMA is clearly more suitable.

Next we compare the percent of permutation sets in which CCA-ROPPPA and CCA-UPGMA returned a consensus clustering with the number of 'desired' or 'expected' clusters, using the definition of 'desired' and 'expected' number of clusters given in section 7.6.2. Table 7.22 shows that CCA-ROPPPA returned a consensus clustering with the desired or expected number of clusters in 647/648 of the permutation sets, while CCA-UPGMA returned a consensus clustering with the desired or expected number of clusters in 646/648 of the permutation

Table 7.22 Whether CCA-ROPPPA and CCA-UPGMA Returned a Consensus Clustering with the Desired or Expected Number of Clusters

	CCA-UPGMA Did Not Return a Partition with the Desired or Expected Number of Clusters	CCA-UPGMA Returned a Partition with the Desired or Expected Number of Clusters	Total
CCA-ROPPPA Did Not Return a Partition with the Desired or Expected Number of Clusters	1	0	1
CCA-ROPPPA Returned a Partition with the Desired or Expected Number of Clusters	1	646	647
Total	2	646	648

sets. Thus, this does not give any evidence that either consensus clustering algorithm is better at returning a consensus clustering with the the desired or expected number of clusters.

Now we compare $RAND_{close}$ of CCA-ROPPPA and CCA-UPGMA. We use the definitions of $RAND_{close}$ and p_{close} for CCA-ROPPPA and CCA-UPGMA given in section 7.6.2.

Table 7.23 shows that 564 of the 648 permutation sets had $(\xi_{close}, RAND_{close})$ of CCA-UPGMA and $(\xi_{close}, RAND_{close})$ of CCA-ROPPPA as the same. It shows that 4 of the 648 permutation sets had CCA-UPGMA have a better ξ_{close} than that of CCA-ROPPPA, whereas only 1 of the 648 permutation sets had CCA-ROPPPA have a better ξ_{close} than that of CCA-UPGMA. Because these numbers (4 and 1) are very small, we do not assume that CCA-ROPPPA or CCA-UPGMA is better at returning a better ξ_{close}.

Table 7.23 also shows that of the 643 permutation sets that had ξ_{close} of CCA-UPGMA and ξ_{close} of CCA-ROPPPA as the same, 55 had $RAND_{close}$ of CCA-ROPPPA better than CCA-UPGMA whereas 24 had $RAND_{close}$ of CCA-UPGMA better than CCA-ROPPPA. Therefore, this shows that if the researcher first desires a ξ_{close} that is as close as possible to ξ and secondly desires $RAND_{close}$ as large as possible, then CCA-ROPPPA outperforms CCA-UPGMA in the

Table 7.23 The RAND Score of the Most Desirable Consensus Clustering Returned by CCA-ROPPPA (CCA-UPGMA) and the Correct Clustering

Permutation Sets	ξ_{close} of CCA-ROPPPA is Better Than That of CCA-UPGMA	ξ_{close} of CCA-ROPPPA is Equal to That of CCA-UPGMA	ξ_{close} of CCA-ROPPPA is Worse Than That of CCA-UPGMA	Total
$RAND_{close}$ of CCA-ROPPPA is Better Than That of CCA-UPGMA	0	55	3	58
$RAND_{close}$ of CCA-ROPPPA is Equal to That of CCA-UPGMA	0	564	0	564
$RAND_{close}$ of CCA-ROPPPA is Worse Than That of CCA-UPGMA	1	24	1	26
Total	1	643	4	648

datasets that we tested.

Finally, we will discuss the differences between the permutation sets in which the results of CCA-UPGMA were different than those of CCA-ROPPPA. For those that were different, we will assess whether CCA-ROPPPA or CCA-UPGMA provided better results. We will, again, use the $RAND_{close}$ of CCA-UPGMA and CCA-ROPPPA to assess which consensus clustering algorithm performed better. Tables 7.24, 7.25, 7.26, and 7.27 below show the average percent of CCA-UPGMA (CCA-ROPPPA) consensus clusterings that are also CCA-ROPPPA (CCA-UPGMA) consensus clusterings aggregated by the original data set, the input clustering trial, the partition set size, and the permutation type respectively.

Table 7.24 shows that three of the artificial data sets (2d-4c-no45sample0, elliptical-10-2-50sample0, and spiral-50sample0) had all permutations sets have CCA-ROPPPA and CCA-UPGMA return the exact same set of consensus clusterings. In addition, we can see that all of artificial data sets had higher average match percentages than all of the real data sets. All the real data sets had a lot more differences in the consensus clustering results returned by CCA-ROPPPA and CCA-UPGMA. Furthermore, three of the four real data sets had more permutation sets in which CCA-ROPPPA had a higher $RAND_{close}$ than CCA-UPGMA.

Table 7.25 shows that input clustering trial 1 had the least disagreement between CCA-

ROPPPA and CCA-UPGMA results and input clustering trial 3 had the most disagreement between CCA-ROPPPA and CCA-UPGMA results. Input clustering trial sets 2-4 had more permutation sets in which CCA-ROPPPA had a higher $RAND_{close}$ than CCA-UPGMA.

Table 7.26 shows that all permutation sets with only partition had CCA-ROPPPA and CCA-UPGMA return the exact same set of consensus clusterings. There is not enough evidence to suggest that permutation sets with four and eight partitions had different average match percentages between CCA-ROPPPA and CCA-UPGMA. However, both permutation sets of size four and eight were more likely to have CCA-ROPPPA have a higher $RAND_{close}$ than CCA-UPGMA.

Finally table 7.27 did not show enough evidence to suggest that ordered and unordered permutation sets had different average match percentages between CCA-ROPPPA and CCA-UPGMA. However, both ordered and unordered permutation sets were more likely to have CCA-ROPPPA have a higher $RAND_{close}$ than CCA-UPGMA.

Table 7.24 The Differences Between the Permutation Sets in Which the Results of CCA-UPGMA were Different than those of CCA-ROPPPA: by Original Data

Original Data	Mean (Std) ARS of the Permutation Set	Mean (Std) Percent of CCA-UPGMA that are Also Returned by CCA-ROPPPA	Mean (Std) Percent of CCA-ROPPPA that are Also Returned by CCA-UPGMA	Number of Permutation Sets where $RAND_{close}$ of CCA-UPGMA was Higher	Number of Permutation Sets where $RAND_{close}$ of CCA-ROPPPA was Higher
2d-3c-no123-50sample0	0.9774 (0.0523)	98.81% (7.09%)	98.26% (8.47%)	0	4
2d-4c-no4-50sample0	0.9866 (0.0239)	100.00% (0.00%)	100.00% (0.00%)	0	0
banana-50sample0	0.9364 (0.1395)	99.38% (3.75%)	99.72% (2.36%)	1	0
brain	0.8843 (0.0874)	76.33% (23.44%)	67.61% (27.14%)	8	25
dlbcl-50sample0	0.8658 (0.1054)	79.12% (24.89%)	72.92% (29.13%)	14	14
elliptical-10-2-50sample0	0.9982 (0.0031)	100.00% (0.00%)	100.00% (0.00%)	0	0
leukemia	0.9118 (0.1049)	95.68% (8.32%)	93.70% (11.75%)	0	7
spiral-50sample0	0.934 (0.1342)	100.00% (0.00%)	100.00% (0.00%)	0	0
tissuemulti-50sample0	0.9623 (0.0373)	92.89% (11.44%)	88.76% (14.99%)	3	8

Table 7.25 The Differences Between the Permutation Sets in Which the Results of CCA-UPGMA were Different than those of CCA-ROPPPA: by Input Clustering Trial

Input clustering Trial Set the Permutation Set was Made With	Mean (Std) ARS of the Permutation Set	Mean (Std) Percent of CCA-UPGMA that are Also Returned by CCA-ROPPPA	Mean (Std) Percent of CCA-ROPPPA that are Also Returned by CCA-UPGMA	Number of Permutation Sets where $RAND_{close}$ of CCA-UPGMA was Higher	Number of Permutation Sets where $RAND_{close}$ of CCA-ROPPPA was Higher
1	0.9706 (0.0616)	95.71% (12.94%)	93.81% (17.50%)	9	5
2	0.9631 (0.0715)	94.15% (15.47%)	92.43% (17.93%)	5	14
3	0.9516 (0.0746)	91.41% (16.73%)	88.53% (20.86%)	7	14
4	0.8733 (0.1368)	93.04% (15.71%)	90.11% (19.40%)	5	25

Table 7.26 The Differences Between the Permutation Sets in Which the Results of CCA-UPGMA were Different than those of CCA-ROPPPA: by Partition Set Size

Number of Partitions in the Permutation Set	Mean (Std) ARS of the Permutation Set	Mean (Std) Percent of CCA-UPGMA that are Also Returned by CCA-ROPPPA	Mean (Std) Percent of CCA-ROPPPA that are Also Returned by CCA-UPGMA	Number of Permutation Sets where $RAND_{close}$ of CCA-UPGMA was Higher	Number of Permutation Sets where $RAND_{close}$ of CCA-ROPPPA was Higher
1	1 (0)	100.00% (0.00%)	100.00% (0.00%)	0	0
4	0.9057 (0.1141)	90.86% (17.73%)	87.84% (21.18%)	11	24
8	0.9132 (0.1044)	89.88% (18.15%)	85.82% (22.88%)	15	34

Table 7.27 The Differences Between the Permutation Sets in Which the Results of CCA-UPGMA were Different than those of CCA-ROPPPA: by Permutation Set Type

Permutation Set Type	Mean (Std) ARS of the Permutation Set	Mean (Std) Percent of CCA-UPGMA that are Also Returned by CCA-ROPPPA	Mean (Std) Percent of CCA-ROPPPA that are Also Returned by CCA-UPGMA	Number of Permutation Sets where $RAND_{close}$ of CCA-UPGMA was Higher	Number of Permutation Sets where $RAND_{close}$ of CCA-ROPPPA was Higher
Ordered	0.9396 (0.0989)	93.79% (15.22%)	91.37% (18.88%)	11	28
Unordered	0.9396 (0.0989)	93.37% (15.45%)	91.07% (19.21%)	15	30

Chapter 8

Discussion

In this chapter we discuss the rationale for using CCA-β particularly for gene expression profile data and we assess how effective a heuristic CCA-ROPPPA is for CCA-β.

In section 8.1 we discuss our findings from our numerical tests. In section 8.2 we show how the first two vector solutions ($\boldsymbol{x}^*(\lambda_1)$ and $\boldsymbol{x}^*(\lambda_2)$) returned by CCA-ROPPPA relate to the minimum cluster ratio cut [Yeh95]. Finally, in section 8.3 we discuss how CCA-ROPPPA is a natural extension of the minimum cluster ratio cut.

8.1 Discussion of Numerical Tests

8.1.1 Benefits and Drawbacks of CCA-ROPPPA

Our numerical tests in chapter 7 suggests that CCA-ROPPPA has the following benefits.

1. CCA-ROPPPA returned a CCA-β solution for 78.40% of the test input clustering sets (permutation sets).

2. In our numerical tests, on average 92.04% of the consecutive vectors pairs returned by CCA-ROPPPA ($\boldsymbol{x}^*(\lambda_{r-1})$, $\boldsymbol{x}^*(\lambda_r)$) defined two consensus clusterings (p^{r-1}, p^r) such that $\beta_C(p^r|p^{r-1})$ was the minimum β-ratio given $C(p^{r-1})$.

3. CCA-ROPPPA yields more options for consensus clustering solutions than CCA-GWLP and CCA-UPGMA.

4. CCA-ROPPPA outperforms CCA-GWLP in returning a consensus clustering with the desired or expected number of clusters.

5. CCA-ROPPPA outperforms CCA-UPGMA (especially when using real data sets) and CCA-GWLP when the researcher
   (a) first desires a consensus clustering where the number of clusters is as close as possible to the 'inherent' number of clusters and
   (b) secondly desires the RAND score of the consensus clustering and the 'correct clustering' as large as possible.

In addition, our numerical tests in chapter 7 suggests that CCA-ROPPPA has the following drawbacks.

1. CCA-UPGMA and CCA-GWLP on average have faster execution times than CCA-ROPPPA.

2. Unlike CCA-UPGMA, CCA-ROPPPA does not always return solutions that define consensus clusterings.

3. Unlike CCA-UPGMA, CCA-ROPPPA does not always return consensus clusterings that are nested in the preceding consensus clustering.

4. CCA-ROPPPA is a heuristic in that it does not always return a CCA-β solution.

8.1.2 Future Work to Address the Drawbacks

Addressing the drawbacks, we have the following suggestions for future research regarding CCA-ROPPPA.

If a researcher would like to use CCA-ROPPPA and it is important for it to more frequently return a CCA-β solution, there are additional constraints that could be added to the LPs in CCA-ROPPPA that may force the vector solutions returned by CCA-ROPPPA to achieve this. For instance, the theorem 6.1 shows us that the set of vectors returned by CCA-ROPPPA, $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$, define a CCA-$\beta$ solution if they satisfy conditions A and B (defined in section 7.5). Because each vector $\boldsymbol{x}^*(\lambda_r)\}$ returned by CCA-ROPPPA is found by solving an LP formulation, we could modify CCA-ROPPPA by adding binary constraints to this LP formulation. Then every solution in $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ would be binary and thus $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ would satisfy condition A. Furthermore, provided that $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ satisfies condition A, we could further modify CCA-ROPPPA by adding the following set of constraints to the LP formulation in CCA-ROPPPA that solves for $\boldsymbol{x}^*(\lambda_r)$ (for $r = 2, ..., R$):
$x_{ij} = 0 \quad \forall (i,j) \in \{(i,j) | \boldsymbol{x}^*(\lambda_{r-1}) = 0\}$

Because $\boldsymbol{x}^*(\lambda_{r-1})$ and $\boldsymbol{x}^*(\lambda_r)$ will both be binary (and thus represent consensus clusterings), this additional set of constraints will force the consensus clustering defined by $\boldsymbol{x}^*(\lambda_r)$ to be nested in the consensus clustering defined by $\boldsymbol{x}^*(\lambda_{r-1})$ for $r = 2, ..., R$. Thus $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ would satisfy conditions A and B. However, because CCA-ROPPPA has been modified, this does not necessarily mean that theorem 6.1 will still hold and thus that $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ will define a CCA-β solution. Thus an avenue for future research may be to determine whether theorem 6.1 still holds given these modification. If so, CCA-ROPPPA would no longer be a heuristic for CCA-β.

Furthermore, if a researcher would like to use CCA-ROPPPA and it is important for it to have a faster execution time, in sections 8.2 and 8.3 we discuss how the parametric LP formulation used in CCA-ROPPPA (P_λ) is a parametric extension of the dual of a LP formulation of the Uniform Multicommodity Flow problem (UMFP). We discuss how Shahrokhi and Matula [SM90] show that the dual of the UMFP is equivalent to the minimum weighted cluster ratio problem. There are several effective polynomial time heuristics found in the literature that approximate solutions to the UMFP and the minimum cluster ratio problem. Therefore, another avenue for future research may be to try to modify one of these polynomial time heuristics to instead approximate the optimal solution to the LP formulation used in CCA-ROPPPA, thus potentially decreasing the total execution time of CCA-ROPPPA.

8.2 How CCA-ROPPPA Relates to the Minimum Cluster Ratio Cut

The purpose of this section is to demonstrate how the first two vector solutions ($\boldsymbol{x}^*(\lambda_1)$ and $\boldsymbol{x}^*(\lambda_2)$) returned by CCA-ROPPPA relate to the minimum cluster ratio cut [Yeh95].

8.2.1 Primal and Dual Formulations Related to the UMFP

In describing the rationale for their proposed metric (the minimum cluster ratio) Yeh et. al. [Yeh95] discuss that clustering can also be considered as a uniform multi-commodity flow problem (UMFP) (defined in chapter 2).

For the graph $G = (V, E)$, the set of commodities $\rho = V \times V$, edge capacities $C : E \mapsto R^+$ they simplify the notation of the UMFP and formulate it as an equivalent problem in which from each node k $(1 \leq k \leq n)$ f units of flow is injected to every other node in the graph. The formulation that they give is below.

$$\max f \tag{8.1}$$

$$\text{st} \sum_{j=1}^{n} x_{ij}^k - \sum_{j=1}^{n} x_{ji}^k = \begin{cases} -f, \text{ if } i \neq k \\ (n-1)f, \text{if } i=k \end{cases} \qquad 1 \leq i, k \leq n \tag{8.2}$$

$$\sum_{j=1}^{n} x_{ij}^k + \sum_{j=1}^{n} x_{ji}^k \leq c_{ij} \qquad 1 \leq i, j, k \leq n \tag{8.3}$$

$$x_{ij}^k \geq 0 \qquad 1 \leq i, j, k \leq n \tag{8.4}$$

,

where x_{ij}^k is the flow originating at node k on edge (i, j), n is the total number of nodes and commodities, and c_{ij} is the capacity of edge (i, j).

To relate their proposed minimum cluster ratio problem to the UMFP, they use the following theorem from Shahrokhi and Matula [SM90].

Theorem 8.2.1 *The maximum uniform multicommodity flow equals the minimum weighted cluster ratio of any cut C, i.e., $max(f) = min(W_C)$.*

Thus from this theorem, the optimal solution of the UMFP problem does not correspond to the optimal solution of the cluster ratio problem. However, when $W_C^* = min_{r \geq 2} W_C(V_1, ..., V_r)$ and $w_{st}^* = w$ for some constant w for all $1 \leq s, t \leq r$, then the optimal solution of the UMFP does correspond to the optimal solution of the cluster ratio problem.

Theorem 8.2.1 follows from the fact that the weighted cluster ratio problem can be formulated as the dual of the UMFP. In this section we show that P_1 (the LP P_λ we use in CCA-ROPPPA in which $\lambda = 1$) is 1.) also the dual of the UMFP and 2.) equivalent to the weighted cluster ratio problem.

First the UMFP LP formulation in (8.1)-(8.4) is the node-arc LP formulation of the UMFP [SM90]. To show the dual relationship of P_1 and the UMFP, we use the edge-path LP formulation of the UMFP problem [SM90] that we give below.

$$
\begin{aligned}
\max\ & f && && (8.5)\\
\text{st}\ & \sum_{p \in P_{ij}} x_p - f \geq 0 && 1 \leq i < j \leq n && (8.6)\\
& \sum_{p:(i,j) \in p} x_p \leq c_{ij} && 1 \leq i < j \leq n && (8.7)\\
& x_p \geq 0 && \forall p && (8.8)\\
& f \geq 0 && && (8.9)
\end{aligned}
$$

,

where P_{ij} is the set of all paths from i to j in the graph $G(V, E)$, x_p is the units of flow from i to j on path $p \in P_{ij}$, and f is the total units of flow from each node i to each node j in $G(V, E)$.

Taking the dual of this LP formulation in (8.5)-(8.9) we get the following [L.11]:

$$
\begin{aligned}
\min\ & \sum_{1 \leq i < j \leq n} c_{ij} d_{ij} && && (8.10)\\
\text{st}\ & \sum_{1 \leq i < j \leq n} w_{ij} \geq 1 && && (8.11)\\
& -w_{ij} + \sum_{(u,v) \in p} d_{uv} \geq 0 && 1 \leq i < j \leq n, p \in P_{ij} && (8.12)\\
& d_{ij}, w_{ij} \geq 0 && 1 \leq i < j \leq n && (8.13)
\end{aligned}
$$

,

where d_{ij} corresponds to the capacity constraints in (8.7) and w_{ij} corresponds to the uniform flow constraints in (8.6).

The following lemmas help us show that the LP in (8.10)-(8.13) is equivalent to that of (8.18)-(8.21)

Lemma 8.2.1 *If* (w^*, d^*) *is an optimal solution of (8.10)-(8.13), then* $\min_{p \in P_{ij}} \{ \sum_{(u,v) \in p} d^*_{uv} \} = w^*_{ij}$ *for all* $1 \leq i < j \leq n$.

Proof Let (w^*, d^*) be an optimal solution of (8.10)-(8.13).

Considering the set of constraints in (8.12) it must be the case that a feasible solution of (8.10)-(8.13) satisfies $\min_{p \in P_{ij}} \{ \sum_{(u,v) \in p} d^*_{uv} \} \geq w^*_{ij}$ for all $1 \leq i < j \leq n$.

Now suppose (w^*, d^*) is an optimal solution and has a w^*_{ij} in which $\min_{p \in P_{ij}} \{ \sum_{(u,v) \in p} d^*_{uv} \} > w^*_{ij}$. Then it follows that the solution $(\frac{\bar{w}}{\epsilon}, \frac{\bar{d}}{\epsilon})$ in which a.) $\bar{w}_{ij} = \min_{p \in P_{ij}} \{ \sum_{(u,v) \in p} d^*_{uv} \}$, b.) $\bar{w}_{st} = w^*_{st}$ for all $(s,t) \neq (i,j)$, c.) $\bar{d}_{st} = d^*_{st}$ for all $1 \leq s < t \leq n$, and d.) $\epsilon = \sum_{s<t} \bar{w}_{st}$ is feasible and produces a smaller objective function value. Because this is the a contradiction to the optimality of (w^*, d^*), it must be the case that $\min_{p \in P_{ij}} \{ \sum_{(u,v) \in p} d^*_{uv} \} = w^*_{ij}$ for all $1 \leq i < j \leq n$

We can therefore think of w_{ij} as the shortest path from i to j in the graph weighted by the d_{uv} variables.

Lemma 8.2.2 *If (w^*, d^*) is an optimal solution of (8.10)-(8.13), then* $\sum_{1 \leq i < j \leq n} w^*_{ij} = 1$.

Proof Let (w^*, d^*) be an optimal solution of (8.10)-(8.13) where $\sum_{1 \leq i < j \leq n} w^*_{ij} = \epsilon > 1$. We can define a new solution in which $(\bar{w}, \bar{d}) = (\frac{w^*}{\epsilon}, \frac{d^*}{\epsilon})$ that satisfies the (8.11) constraint with equality, satisfies the (8.12) and (8.13) constraints, and produces a smaller objective function value than that of (w^*, d^*).

Thus we the following LP in (8.14)-(8.17) is equivalent to the LP in (8.10)-(8.13).

$$\min \sum_{1 \leq i < j \leq n} c_{ij} d_{ij} \tag{8.14}$$

$$\text{st} \sum_{1 \leq i < j \leq n} w_{ij} = 1 \tag{8.15}$$

$$\sum_{(u,v) \in p} d_{uv} \geq w_{ij} \qquad 1 \leq i < j \leq n, p \in P_{ij} \tag{8.16}$$

$$d_{ij}, w_{ij} \geq 0 \qquad 1 \leq i < j \leq n \tag{8.17}$$

,

Next we show the following.

Lemma 8.2.3 *If (w^*, d^*) is optimal for the LP in (8.14)-(8.17), then $w^* = d^*$.*

Proof Suppose that (w^*, d^*) is a feasible, optimal solution for the LP in (8.14)-(8.17), but $w^* \neq d^*$.

If there exists a $d^*_{uv} < w^*_{uv}$ then this violates constraint 8.16, which is a contradiction to the feasibility of (w^*, d^*).

Now suppose there exists a $d^*_{uv} > w^*_{uv}$. Let $(\hat{w}, \hat{d})$ be a solution in (8.14)-(8.17) such that $\hat{w} = w^*$, $\hat{d}_{ij} = d^*_{ij}$ for all $(i,j) \neq (u,v)$, and $\hat{d}_{uv} = w^*_{uv} = \hat{w}_{uv}$. Because (w^*, d^*) is feasible in (8.14)-(8.17), clearly constraints (8.15) and (8.17) still hold for $(\hat{w}, \hat{d})$ and the objective function value in (8.14) for $(\hat{w}, \hat{d})$ is smaller than that of (w^*, d^*). If a path $p \in P_{ij}$ does not contain the edge (u,v), then clearly constraint (8.16) still holds. If the path $p \in P_{ij}$ does contain the edge (u,v), then it follows that

$$\begin{aligned}
\sum_{(s,t)\in p} \hat{d}_{st} &\geq \hat{w}_{iu} + \hat{d}_{uv} + \hat{w}_{vj} \\
&= \hat{w}_{iu} + \hat{w}_{uv} + \hat{w}_{vj} \\
&= w^*_{iu} + w^*_{uv} + w^*_{vj} \\
&= \min_{p\in P_{iu}} \{ \sum_{(s,t)\in p} d^*_{st} \} + \min_{p\in P_{uv}} \{ \sum_{(s,t)\in p} d^*_{st} \} + \min_{p\in P_{vj}} \{ \sum_{(s,t)\in p} d^*_{st} \} \\
&\geq \min_{p\in P_{ij}} \{ \sum_{(s,t)\in p} d^*_{st} \} \\
&= w^*_{ij} \\
&= \hat{w}_{ij}
\end{aligned}$$

The third and the fourth line above follow from lemma 8.2.1. Thus $(\hat{w}, \hat{d})$ is feasible in (8.14)-(8.17) and produces a smaller objective function value than that of (w^*, d^*), which is a contradiction to the optimality of (w^*, d^*). Thus it must be the case that $w^* = d^*$.

Because of this lemma, the LP in (8.14)-(8.17) is equivalent to (8.18)-(8.21)

$$\min \sum_{1\leq i<j\leq n} c_{ij} d_{ij} \tag{8.18}$$

$$\text{st} \sum_{1\leq i<j\leq n} d_{ij} = 1 \tag{8.19}$$

$$\sum_{(u,v)\in p} d_{uv} \geq d_{ij} \qquad 1 \leq i < j \leq n, p \in P_{ij} \tag{8.20}$$

$$d_{ij} \geq 0 \qquad 1 \leq i < j \leq n \tag{8.21}$$

,

Now notice that the constraints in (8.20) are equivalent to those of the triangle inequality. Thus the LP in (8.18)-(8.21) is equivalent to that of (8.22)-(8.25).

$$\min \sum_{1 \le i < j \le n} c_{ij} d_{ij} \tag{8.22}$$

$$\text{st} \sum_{1 \le i < j \le n} d_{ij} = 1 \tag{8.23}$$

$$\begin{aligned} d_{ij} + d_{jk} &\ge d_{ik} \\ d_{jk} + d_{ik} &\ge d_{ij} \qquad 1 \le i < j < k \le n \\ d_{ik} + d_{ij} &\ge d_{jk} \end{aligned} \tag{8.24}$$

$$d_{ij} \ge 0 \qquad 1 \le i < j \le n \tag{8.25}$$

,

Because of constraint (8.23) and (8.25) it follows that $d_{ij} \le 1$ for all $1 \le i < j \le n$. Because of this we can let $x_{ij} = 1 - d_{ij}$ and attain the following equivalent LP in (8.26)-(8.29).

$$\max \sum_{1 \le i < j \le n} c_{ij} x_{ij} \tag{8.26}$$

$$\text{st} \sum_{1 \le i < j \le n} x_{ij} = \binom{n}{2} - 1 \tag{8.27}$$

$$\begin{aligned} x_{ij} + x_{jk} - x_{ik} &\le 1 \\ x_{ij} - x_{jk} + x_{ik} &\le 1 \qquad 1 \le i < j < k \le n \\ -x_{ij} + x_{jk} + x_{ik} &\le 1 \end{aligned} \tag{8.28}$$

$$x_{ij} \ge 0 \qquad 1 \le i < j \le n \tag{8.29}$$

,

We can now see that the dual of the UMFP is equivalent to P_1, where P_λ is the parametric LP we use in CCA-ROPPPA.

If we let $y_{ij}d_0 = d_{ij}$ for $1 \leq i < j \leq n$ where $d_0 = \frac{1}{\sum\limits_{1\leq i<j\leq n} y_{ij}}$, it follows that (8.22)-(8.25) is also equivalent to the following formulation.

$$\min \frac{\sum_{1\leq i<j\leq n} c_{ij}y_{ij}}{\sum_{1\leq i<j\leq n} y_{ij}} \tag{8.30}$$

$$\text{st} \quad \begin{aligned} y_{ij} + y_{jk} &\geq y_{ik} \\ y_{jk} + y_{ik} &\geq y_{ij} \\ y_{ik} + y_{ij} &\geq y_{jk} \end{aligned} \qquad 1 \leq i < j < k \leq n \tag{8.31}$$

$$y_{ij} \geq 0 \qquad 1 \leq i < j \leq n \tag{8.32}$$

Finally, Shahrokhi and Matula [SM90] show that (8.30)-(8.32) is equivalent to the weighted cluster ratio problem. Thus P_1 is equivalent to the weighted cluster ratio problem.

8.2.2 Relationship between the Cluster Ratio Problem and CCA-ROPPPA

Because the dual of the UMFP is the weighted cluster ratio problem [SM90] and P_1 and because the weighted cluster ratio problem is equivalent to P_1, then this demonstrates a clear relationship between the weighted cluster ratio problem and CCA-ROPPPA (which uses P_λ). Under the following conditions we can also demonstrate a relationship between the cluster ratio problem and CCA-ROPPPA.

- When the w_{ij} variables in the optimal weighted cluster ratio (equation 2.6) are all equal, then the minimum weighted cluster ratio is also the minimum cluster ratio.
- When $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ is the set of vectors returned by CCA-ROPPPA and $\boldsymbol{x}^*(\lambda_2)$ is binary, then $\boldsymbol{x}^*(\lambda_2)$ defines a consensus clustering that is the optimal solution to the minimum cluster ratio problem.

8.3 How CCA-ROPPPA is an Extension of the Minimum Cluster Ratio Problem

The purpose of this section is to show how CCA-ROPPPA is a natural extension of the minimum cluster ratio problem.

8.3.1 Primal and Dual Formulations Related to an Extension of the UMFP

First, we create a modification of the uniform multi-commodity flow problem (UMFP) and formulate it as the following parametric LP below.

$$\max \lambda f - \sum_{1 \le i < j \le n} e_{ij} \tag{8.33}$$

$$\text{st} \sum_{p \in P_{ij}} x_p - f \ge 0 \qquad 1 \le i < j \le n \tag{8.34}$$

$$\sum_{p|(i,j) \in p} x_p - e_{ij} \le c_{ij} \qquad 1 \le i < j \le n \tag{8.35}$$

$$x_p \ge 0 \qquad \forall p \tag{8.36}$$

$$e_{ij} \ge 0 \qquad \forall 1 \le i < j \le n \tag{8.37}$$

$$f \ge 0 \tag{8.38}$$

,

This is the same formulation as that represented in (8.5)-(8.9), except we have added a parametric component λ to the objective function and we have added e_{ij} variables to the capacity constraints and the objective function. Whereas the capacity constraint defined in (8.7) could not be violated, the introduction of the e_{ij} variables allows us to violate the capacity constraints. The variable e_{ij} indicates how much the capacity constraint c_{ij} was violated.

Notice that when $\lambda \le 0$, the optimal solution will have none of the capacity constraint violated, or $e_{ij} = 0$ for all $1 \le i < j \le n$. However, the optimal flow will have $f = 0$, so any feasible flow will end up being optimal. Notice that as λ increases, the reward of the maximized uniform flow will begin to counterbalance the penalty of violating some of the capacity constraints. Therefore, as λ increases, we may have more variables in which $e_{ij} > 0$.

Next we take the dual of the LP in (8.33)-(8.38) shown below in (8.39)-(8.43).

$$\min \sum_{1\leq i<j\leq n} c_{ij}d_{ij} \tag{8.39}$$

$$\text{st} \sum_{1\leq i<j\leq n} w_{ij} \geq \lambda \tag{8.40}$$

$$-w_{ij} + \sum_{(u,v)\in p} d_{uv} \geq 0 \qquad 1 \leq i < j \leq n, p \in P_{ij} \tag{8.41}$$

$$d_{ij} \leq 1 \qquad 1 \leq i < j \leq n \tag{8.42}$$

$$d_{ij}, w_{ij} \geq 0 \qquad 1 \leq i < j \leq n \tag{8.43}$$

,

Using the same logic from section 8.2, we prove the following lemmas to attain an equivalent formulation.

Lemma 8.3.1 *If (w^*, d^*) is an optimal solution of (8.39)-(8.43), then $\min_{p\in P_{ij}}\{\sum_{(u,v)\in p} d^*_{uv}\} = w^*_{ij}$ for all $1 \leq i < j \leq n$.*

Proof Let (w^*, d^*) be an optimal solution of (8.39)-(8.43).

Considering the set of constraints in (8.41) it must be the case that a feasible solution of (8.39)-(8.43) satisfies $\min_{p\in P_{ij}}\{\sum_{(u,v)\in p} d^*_{uv}\} \geq w^*_{ij}$ for all $1 \leq i < j \leq n$.

Now suppose (w^*, d^*) is an optimal solution and has a w^*_{ij} in which $\min_{p\in P_{ij}}\{\sum_{(u,v)\in p} d^*_{uv}\} > w^*_{ij}$. Then it follows that the solution $(\frac{\lambda\bar{w}}{\epsilon}, \frac{\lambda\bar{d}}{\epsilon})$ in which a.) $\bar{w}_{ij} = \min_{p\in P_{ij}}\{\sum_{(u,v)\in p} d^*_{uv}\}$, b.) $\bar{w}_{st} = w^*_{st}$ for all $(s,t) \neq (i,j)$, c.) $\bar{d}_{st} = d^*_{st}$ for all $1 \leq s < t \leq n$, and d.) $\epsilon = \sum_{s<t} \bar{w}_{st}$ is feasible and produces a smaller objective function value. Because this is the a contradiction to the optimality of (w^*, d^*), it must be the case that $\min_{p\in P_{ij}}\{\sum_{(u,v)\in p} d^*_{uv}\} = w^*_{ij}$ for all $1 \leq i < j \leq n$

Lemma 8.3.2 *If (w^*, d^*) is an optimal solution of (8.39)-(8.43), then $\sum_{1\leq i<j\leq n} w^*_{ij} = \lambda$.*

Proof Let (w^*, d^*) be an optimal solution of (8.39)-(8.43) where $\sum_{1\leq i<j\leq n} w^*_{ij} = \epsilon > \lambda$. We can define a new solution in which $(\bar{w}, \bar{d}) = (\frac{\lambda w^*}{\epsilon}, \frac{\lambda d^*}{\epsilon})$ that satisfies the (8.40) constraint with equality, satisfies the remaining constraints, and produces a smaller objective function value than that of (w^*, d^*).

Thus we get the following LP in (8.44)-(8.48) is equivalent to the LP in (8.39)-(8.43).

$$
\begin{aligned}
\min \quad & \sum_{1 \le i < j \le n} c_{ij} d_{ij} && && (8.44)\\
\text{st} \quad & \sum_{1 \le i < j \le n} w_{ij} = \lambda && && (8.45)\\
& \sum_{(u,v) \in p} d_{uv} \ge w_{ij} && 1 \le i < j \le n, p \in P_{ij} && (8.46)\\
& d_{ij} \le 1 && 1 \le i < j \le n && (8.47)\\
& d_{ij}, w_{ij} \ge 0 && 1 \le i < j \le n && (8.48)
\end{aligned}
$$

,

Next we show the following.

Lemma 8.3.3 *If (w^*, d^*) is optimal for the LP in (8.44)-(8.48), then $w^* = d^*$.*

Proof Suppose that (w^*, d^*) is a feasible, optimal solution for the LP in (8.44)-(8.48), but $w^* \ne d^*$.

If there exists a $d^*_{uv} < w^*_{uv}$ then this violates constraint 8.46, which is a contradiction to the feasibility of (w^*, d^*).

Now suppose there exists a $d^*_{uv} > w^*_{uv}$. Let $(\hat{w}, \hat{d})$ be a solution in (8.44)-(8.48) such that $\hat{w} = w^*$, $\hat{d}_{ij} = d^*_{ij}$ for all $(i, j) \ne (u, v)$, and $\hat{d}_{uv} = w^*_{uv} = \hat{w}_{uv}$. Because (w^*, d^*) is feasible in (8.44)-(8.48), clearly constraints (8.45), (8.47) and (8.48) still hold for $(\hat{w}, \hat{d})$ and the objective function value in (8.44) for $(\hat{w}, \hat{d})$ is smaller than that of (w^*, d^*). If a path $p \in P_{ij}$ does not contain the edge (u, v), then clearly constraint (8.46) still holds. If the path $p \in P_{ij}$ does contain the edge (u, v), then it follows that

$$
\begin{aligned}
\sum_{(s,t)\in p} \hat{d}_{st} \quad &\geq \hat{w}_{iu} + \hat{d}_{uv} + \hat{w}_{vj} \\
&= \hat{w}_{iu} + \hat{w}_{uv} + \hat{w}_{vj} \\
&= w^*_{iu} + w^*_{uv} + w^*_{vj} \\
&= \min_{p\in P_{iu}} \{ \sum_{(s,t)\in p} d^*_{st} \} + \min_{p\in P_{uv}} \{ \sum_{(s,t)\in p} d^*_{st} \} + \min_{p\in P_{vj}} \{ \sum_{(s,t)\in p} d^*_{st} \} \\
&\geq \min_{p\in P_{ij}} \{ \sum_{(s,t)\in p} d^*_{st} \} \\
&= w^*_{ij} \\
&= \hat{w}_{ij}
\end{aligned}
$$

The third and the fourth line above follow from lemma 8.3.1.

Thus $(\hat{w}, \hat{d})$ is feasible in (8.44)-(8.48) and produces a smaller objective function value than that of (w^*, d^*), which is a contradiction to the optimality of (w^*, d^*). Thus it must be the case that $w^* = d^*$.

Because of this lemma, we can write the LP in (8.49)-(8.52) that is equivalent to (8.44)-(8.48)

$$
\begin{aligned}
\min \quad & \sum_{1\leq i<j\leq n} c_{ij} d_{ij} && && (8.49) \\
\text{st} \quad & \sum_{1\leq i<j\leq n} d_{ij} = \lambda && && (8.50) \\
& \sum_{(u,v)\in p} d_{uv} \geq d_{ij} && 1 \leq i < j \leq n, p \in P_{ij} && (8.51) \\
& 0 \leq d_{ij} \leq 1 && 1 \leq i < j \leq n && (8.52)
\end{aligned}
$$

,

The constraints in (8.51) are equivalent to those of the triangle inequality. Thus the LP in (8.49)-(8.52) is equivalent to that of (8.53)-(8.56).

$$
\begin{aligned}
\min \sum_{1\le i<j\le n} c_{ij}d_{ij} && (8.53)\\
\text{st} \sum_{1\le i<j\le n} d_{ij} = \lambda && (8.54)\\
d_{ij}+d_{jk}\ge d_{ik} && \\
d_{ik}+d_{jk}\ge d_{ij} \qquad & 1\le i<j<k\le n & (8.55)\\
d_{ij}+d_{ik}\ge d_{jk} && \\
0\le d_{ij}\le 1 \qquad & 1\le i<j\le n & (8.56)
\end{aligned}
$$

,

Now we can let $x_{ij} = 1 - d_{ij}$ and attain the following equivalent LP.

$$
\begin{aligned}
\max \sum_{1\le i,j\le n} c_{ij}x_{ij} && (8.57)\\
\text{st} \sum_{1\le i<j\le n} x_{ij} = \binom{n}{2} - \lambda && (8.58)\\
x_{ij}+x_{jk}-x_{ik}\le 1 && \\
x_{ij}-x_{jk}+x_{ik}\le 1 \qquad & 1\le i,j,k\le n & (8.59)\\
-x_{ij}+x_{jk}+x_{ik}\le 1 && \\
x_{ij}\ge 0 \qquad & 1\le i<j\le n & (8.60)
\end{aligned}
$$

,

We can now see that the dual of our modified UMFP represented in the parametric LP in (8.33)-(8.38) is equivalent to P_λ, which we use in CCA-ROPPPA.

Because our modified UMFP problem is a parametric extension of the UMFP problem, we can think of P_λ (the dual of this parametric extension of the UMFP problem) as an extension of P_1 (the dual of the UMFP problem). And because the weighted cluster ratio problem is equivalent to P_1, we can think of P_λ as a parametric extension of the weighted cluster ratio problem.

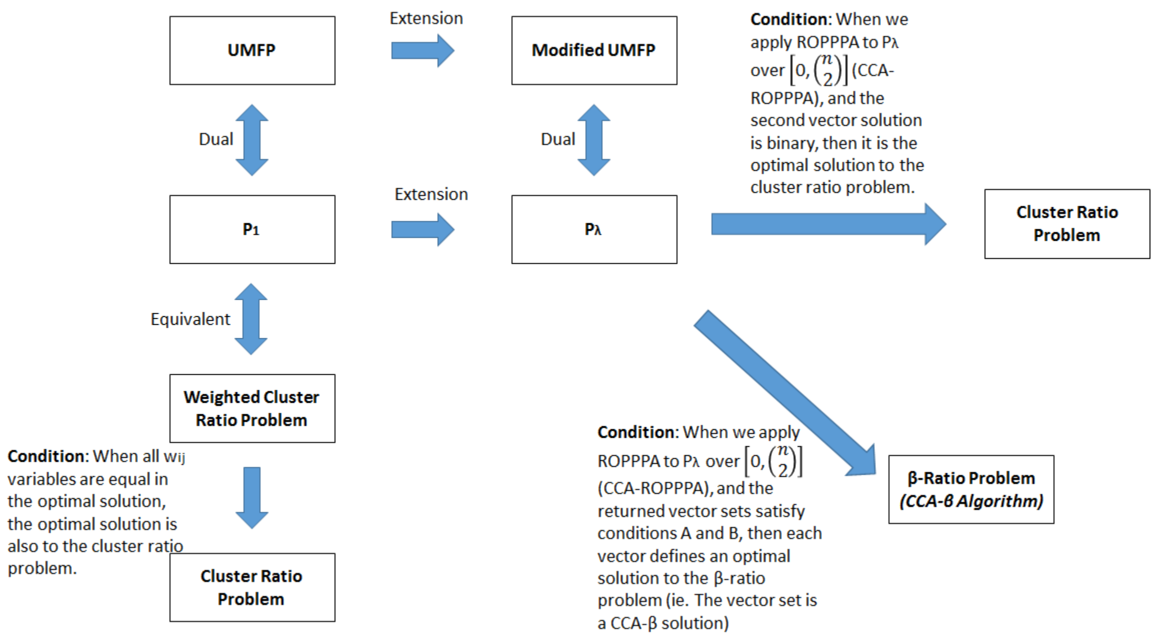


Figure 8.1 LP Relationships

Like we showed in section 8.2, an avenue for future research may be to prove the formulation in (8.53)-(8.56) is equivalent to that of our minimum β-ratio problem (or CCA-β.

However, the fact that a.) CCA-ROPPPA uses P_λ and b.) returns a CCA-β solution when conditions A and B are met (theorem 6.1), also shows that CCA-β is related to the cluster ratio problem and thus contributes to the literature surrounding the cluster ratio problem.

Figure 8.1 summarizes these relationships that we discussed in sections 8.2 and 8.3.

Several fast and efficient heuristics [Yeh95] have been proposed to find an approximate solution to the P_1 (the weighted cluster ratio problem) and the cluster the ratio problem. Because P_λ is a quite a natural parametric extension of the P_1, future work may involve modifying these heuristics to solve for the approximate solutions of P_λ in the CCA-ROPPPA algorithm.

Chapter 9

Conclusion

In this dissertation we set out to create a novel consensus clustering algorithm. The aim of this consensus clustering algorithm that we created was to have it return a *series* of consensus clusterings that achieved a high degree of consensus with a set of input clusterings of a data set.

9.1 Summary of Contributions

To summarize, our proposed consensus clustering algorithm is called CCA-β. It is a co-occurence consensus clustering algorithm that creates a similarity matrix from the input clusterings and reclusters the similarity matrix using a divisive hierarchical clustering algorithm. Divisive hierarchical clustering algorithms typically use a metric that determines which and how a cluster(s) is split in each iteration of the algorithm. We used our proposed minimum β-ratio for this metric. We show that the minimum β-ratio cut is an extension of the well-researched minimum cluster ratio cut.

We also created a heuristic for CCA-β, which we call CCA-ROPPPA. We call it a heuristic of CCA-β, because it does not always return a set of results that *would* have been returned by CCA-β. CCA-ROPPPA applies a parametric programming algorithm to parametric linear programming formulation that we propose and call P_λ in this dissertation.

Our numerical tests showed that CCA-ROPPPA returned a CCA-β solution for 78.40% of our input clustering sets. Our tests also indicated that the following factors may have affected the percent of input clusterings sets that had CCA-ROPPPA return a CCA-β solution: a.) the original data set the input clustering set was generated from, b.) the number of input clusterings in the set, and c.) the number of clusters that the input clusterings in the set had.

We also proved that if CCA-ROPPPA returns a set of *vectors* $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ that define a series of nested consensus clusterings $\{p^1, ..., p^R\}$, then $\{p^1, ..., p^R\}$ is also a CCA-β solution. In chapter 8 we introduced several ideas for future work that may force the set of solutions $\{\boldsymbol{x}^*(\lambda_1), ..., \boldsymbol{x}^*(\lambda_R)\}$ returned by CCA-ROPPPA to define a series of nested consensus clusterings $\{p^1, ..., p^R\}$. However, because this modification would change CCA-ROPPPA, this does not necessarily mean that $\{p^1, ..., p^R\}$ would be a CCA-β solution. Therefore, an idea for future work may be to determine if this result still holds if we force CCA-ROPPPA to return a series of nested consensus clusterings $\{p^1, ..., p^R\}$.

9.2 Goals of our Consensus Clustering Algorithm and Heuristic

In addition to returning a *series* of consensus clusterings that achieve a high degree of consensus with the set of input clusterings, we also sought to have our consensus clustering algorithm CCA-β and its heuristic CCA-ROPPPA particularly meet the needs of a researcher when the input clusterings were created by clustering a gene expression profile data set.

Because it is often important for a researcher analyzing gene expression profile data to be able to choose the number of clusters returned by a clustering algorithm, we sought to create a consensus clustering algorithm that returned more than one consensus clustering, so the researcher can choose a consensus clustering that has the closest number of clusters to what he or she desired or expected. One of the most common consensus clustering algorithms, the median partition problem (which can be modeled as the integer program GWIP) does not offer this capability of producing more than one consensus clustering or preselecting the number of clusters returned. In addition, 87.35% of the test input clustering sets had CCA-GWLP (which uses GWIP) return a consensus clustering having the number of clusters that the researcher would desire or expect. On the other hand, 99.85% of the test input clustering sets had CCA-ROPPPA return a consensus clustering having the number of clusters that the researcher would desire or expect. Furthermore, the minimum cluster ratio problem can only return one consensus clustering and the user cannot preselect the number of clusters returned. Because GWIP and the mathematical formulation of the minimum cluster ratio problem have very similar feasible regions, we created CCA-ROPPPA which uses P_λ that a.) uses these feasible regions, b.) is an extension of an LP formulation used to find the minimum cluster ratio cut, and c.) has the ability to produce more than one consensus clustering.

Because gene expression profile data is often nested, we sought to create a consensus clustering algorithm that returned a series of nested consensus clusterings. Our numerical tests showed that the average percent of consecutive pairs in each vector set returned by CCA-ROPPPA that had both vectors define a consensus clustering with the second nested in the first was 92.04%. Our tests also indicated that the following factors may have affected this metric: a.) the original data set the input clustering set was generated from, b.) the number of input clusterings in the set, and c.) the number of clusters that the input clusterings in the set had. Note that co-occurence consensus clustering algorithms that use agglomerative hierarchical clustering algorithms such as UPGMA (CCA-UPGMA) will always return a series of nested consensus clusterings.

Next, gene expression profile data can be used to classify tissue samples into well researched tissue sub-types (clusters). Many gene expression profile data sets found in the literature come with a 'correct clustering' in order for a researcher to test the performance of his or her clustering (or consensus clustering) algorithm. Because of this, we sought to create a consensus clustering algorithm that returned a consensus clustering that was as close as possible to the 'correct clustering.' Our numerical tests show that if the researcher primarily desires a consensus clustering where the number of clusters is as close as possible to the 'correct' number of clusters and secondarily desires a consensus clustering that has a high RAND score with the correct clustering, then CCA-ROPPPA outperforms CCA-UPGMA and CCA-GWLP.

Next, the minimum cluster ratio problem has been used in the literature for a.) its ability to create a *single* clustering with clusters that are roughly balanced in size, and b.) its rigorous mathematical structure. This mathematical structure has allowed for researchers to theoretically prove relationships to other well-studied problems to which there exist fast heuristics, and in doing so this has subsequently allowed them to create fast heuristics for the minimum cluster ratio problem. Because of this, we sought to create a consensus clustering algorithm that was an extension of the minimum cluster ratio problem. Specifically we designed our CCA-β algorithm such that it provides a *series* of nested consensus clusterings that were attained using a metric that is an extension of the minimum cluster ratio cut (which we call the minimum β ratio cut.)

Finally, because CCA-ROPPPA is a linear programming based algorithm that uses a linear programming formulation with $3\binom{n}{3}$ constraints, clearly the execution time of consensus clusterings algorithms and heuristics that are polynomially time solvable (like CCA-UPGMA) will be much faster than that of CCA-ROPPPA especially as the number of n objects increases. However, there does exist quite a bit of research dedicated to finding binary optimal solutions to GWIP in a more time efficient way. Because the feasible region of P_λ that we use in CCA-ROPPPA is very similar to that of GWIP and because we specifically use a parametric programming algorithm found in the literature (OPPPA) that allows for not the whole feasible region of P_λ to be needed in order to execute the algorithm, we are able to apply some of these techniques to OPPPA, thus creating ROPPPA. In chapter 5 for future work we suggest also applying column generation techniques in ROPPPA to allow for even faster execution time. In addition, there exist many other fast solving techniques applied to GWIP mentioned in section 1.4 that we did not incorporate into CCA-ROPPPA. Thus, future work may involve incorporating these techniques in order to be able to solve consensus clusterings problems with $n > 1000$ objects like the researchers are able to do in some instances. Finally, because we were able to mathematically show a strong relationship between CCA-β, CCA-ROPPPA, and the minimum cluster ratio problem; and because the literature offers several fast heuristics for the minimum cluster ratio problem, future work may involve trying to modify one or more of these heuristics to find fast approximate solutions to CCA-β.

[Ban] Bandyopadhyay, S. *Data Sets*. URL: `http://www.isical.ac.in/~sanghami/data.html` (visited on 09/12/2016).

[BP10] Bandyopadhyay, S. & Pal, S. *Classification and Learning Using Genetic Algorithms: Applications in Bioinformatics and Web Intelligence*. Natural Computing Series. Springer Berlin Heidelberg, 2010.

[BM01] Bandyopadhyay, S. & Maulik, U. "Nonparametric genetic clustering: comparison of validity indices". *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* **31**.1 (2001), pp. 120–125.

[BM04] Bandyopadhyay, S. & Maulik, U. "Genetic clustering for automatic evolution of clusters and application to image classification." *Pattern Recognition* **35**.6 (19, 2004), pp. 1197–1208.

[Bar15] Barton, T. *Clustering Benchmark*. `https://github.com/deric/clustering-benchmark`. 2015.

[BD99] Ben-Dor, A. et al. "Clustering Gene Expression Patterns". *Journal of Computational Biology* **6**.3/4 (1999), pp. 281–297.

[Ber96] Berkelaar, A. et al. *The Optimal Set and Optimal Partition Approach to Linear and Quadratic Programming*. Tech. rep. in Advances in Sensitivity Analysis and Parametric Programming, 1996.

[BT97] Bertsimas, D. & Tsitsiklis, J. *Introduction to Linear Optimization*. 1st. Athena Scientific, 1997.

[Dow10] Downing, N. et al. "Improved consensus clustering via linear programming." *ACSC*. Ed. by Mans, B. & Reynolds, M. Vol. 102. CRPIT. Australian Computer Society, 2010, pp. 61–70.

[Eis98] Eisen, M. B. et al. "Cluster analysis and display of genome-wide expression patterns". **95** (1998), pp. 14863–14868.

[FS03] Filkov, V. & Skiena, S. "Integrating microarray data by consensus clustering". *Tools with Artificial Intelligence, 2003. Proceedings. 15th IEEE International Conference on*. 2003, pp. 418–426.

[GG97] Gal, T. & Greenberg, H. *Advances in Sensitivity Analysis and Parametric Programming*. International Series in Operations Research & Management Science. Springer US, 1997.

[GW89] Grötschel, M. & Wakabayashi, Y. "A Cutting Plane Algorithm for a Clustering Problem". *Math. Program.* **45**.1 (1989), pp. 59–96.

[Han] Handl, J. *Cluster Generators*. URL: `http://personalpages.manchester.ac.uk/mbs/Julia.Handl/generators.html` (visited on 09/12/2016).

[HK04] Handl, J. & Knowles, J. "Multiobjective clustering with automatic determination of the number of clusters" (2004).

[HS00] Hartuv, E. & Shamir, R. "A Clustering Algorithm Based on Graph Connectivity". *Inf. Process. Lett.* **76**.4-6 (2000), pp. 175–181.

[Her01] Herrero, J. et al. "A hierarchical unsupervised growing neural network for clustering gene expression patterns." *Bioinformatics* **17**.1 (2001), pp. 126–136.

[HA85] Hubert, L. & Arabie, P. "Comparing partitions". English. *Journal of Classification* **2**.1 (1985), pp. 193–218.

[Jia04] Jiang, D. et al. "Cluster Analysis for Gene Expression Data: A Survey". *IEEE Trans. on Knowl. and Data Eng.* **16**.11 (2004), pp. 1370–1386.

[KK08] Kashef, R. & Kamel, M. S. "Cooperative Partitional-Divisive Clustering and Its Application in Gene Expression Analysis." *BIBE*. IEEE, 27, 2008, pp. 116–122.

[Koh84] Kohonen, T. *Self-Organization and Associative Memory*. First. Berlin; New York: Springer, 1984.

[KM86] Krivnek, M. & Morvek, J. "NP-hard problems in hierarchical-tree clustering". *Acta Informatica* **23**.3 (1986), pp. 311–323.

[L.11] L., T. "Lecture 16". *Computer Science 261: Optimization, Standford University* (2011).

[Lec94] Leclerc, B. "Medians for weight metrics in the covering graphs of semilattices". *Discrete Applied Mathematics* **49**.13 (1994). Special Volume Viewpoints on Optimization, pp. 281 –297.

[LR99] Leighton, T. & Rao, S. "Multicommodity Max-flow Min-cut Theorems and Their Use in Designing Approximation Algorithms". *J. ACM* **46**.6 (1999), pp. 787–832.

[Llo82] Lloyd, S. P. "Least squares quantization in pcm". *IEEE Transactions on Information Theory* **28** (1982), pp. 129–137.

[Man08] Manning, C. D. et al. *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008.

[Mir96] Mirkin, B. G. *Mathematical classification and clustering*. Nonconvex optimization and its applications. Dordrecht, Boston, London: Kluwer academic publ, 1996.

[Mon03] Monti, S. et al. “Consensus Clustering: A Resampling-Based Method for Class Discovery and Visualization of Gene Expression Microarray Data”. *Mach. Learn.* **52**.1-2 (2003), pp. 91–118.

[Pro15] Program, B. I. C. *Broad Institute Cancer Program Database*. `http://www.broadinstitute.org/cgi-bin/cancer/datasets.cgi`. 2015.

[Ran71] Rand, W. “Objective criteria for the evaluation of clustering methods”. *Journal of the American Statistical Association* **66**.336 (1971), pp. 846–850.

[SM90] Shahrokhi, F. & Matula, D. W. “The Maximum Concurrent Flow Problem”. *J. ACM* **37**.2 (1990), pp. 318–334.

[SS00] Shamir, R. & Sharan, R. “CLICK: A Clustering Algorithm for Gene Expression Analysis”. AAAI Press, 2000.

[She01] Sherlock, G. “Analysis of large-scale gene expression data”. *Briefings in Bioinformatics* **2**.4 (2001), p. 350. eprint: `/oup/backfile/Content_public/Journal/bib/2/4/10.1093/bib/2.4.350/2/350.pdf`.

[Sme02] Smet, F. D. et al. “Adaptive quality-based clustering of gene expression profiles.” *Bioinformatics* **18**.5 (2002), pp. 735–746.

[SM58] Sokal, R. R. & Michener, C. D. “A statistical method for evaluating systematic relationships”. *University of Kansas Scientific Bulletin* **28** (1958), pp. 1409–1438.

[Tam99] Tamayo, P. et al. “Interpreting Patterns of Gene Expression with Self-Organizing Maps: Methods and Application to Hematopoietic Differentiation”. *Proceedings of the National Academy of Sciences, USA* **96** (1999), pp. 2907–2912.

[Tüs83] Tüshaus, U. *Aggregation binärer Relationen in der qualitativen Datenanalyse*. Verlag Anton Hain, 1983.

[VPRS11] Vega-Pons, S. & Ruiz-Shulcloper, J. “A Survey of Clustering Ensemble Algorithms.” *IJPRAI* **25**.3 (2011), pp. 337–372.

[Wak86] Wakabayashi, Y. *Aggregation of Binary Relations: Algorithmic and Polyhedral Investigations*. 1986.

[XK01] Xing, E. P. & Karp, R. M. “CLIFF: clustering of high-dimensional microarray data via iterative feature filtering using normalized cuts”. *Bioinformatics* **17**.suppl 1 (2001), S306–S315.

[Yeh95] Yeh, C. et al. “Circuit clustering using a stochastic flow injection method”. *IEEE Trans. on CAD of Integrated Circuits and Systems* **14**.2 (1995), pp. 154–162.

APPENDIX

Appendix A

Appendix

The following additional items can found at:
https://victoriaellison.wordpress.com/research.

- the visualizations of all the consensus clustering results returned by CCA-UPGMA, CCA-GWLP, and CCA-ROPPPA in the numerical tests chapter.
- the 648 permutation sets used as input for CCA-UGMA, CCA-GWLP, and CCA-ROPPPA in the numerical tests chapter
- ”label” files for each of the 648 permutation sets. (The second row in the label files indicates the ”index” or the original order of the object in the original data file. The first row indicates the correct cluster label assigned to the object index in the row above.)
- the indices of the 50 objects that were sampled from the following data sets:
  - spiral
  - elliptical-10-2
  - banana
  - 2d-4c-no4
  - 2d-3c-no123
  - dlbcl
  - tissuemulti
- a table of the average number of consensus clusterings returned by CCA-ROPPPA and CCA-UPGMA (grouped by by original data set, the number of input clusterings in the permutation set, and the input clustering trial set)