Abstract

KULKARNI, GIRISH. Exact and Heuristic Algorithms for the q-mode Problem. (Under the direction of Dr. Yahya Fathi.)

In this dissertation we focus on the development of exact and inexact (i.e., heuristic) algorithms for the q-mode problem. The exact algorithms are based on integer programming models for the q-mode problem. We discuss the theoretical properties of an existing IP model and propose several enhancements. We also propose a new IP model for the problem and investigate these models through a comprehensive computational experiment. The computational experiment reveals that, in practice, the IP models are more effective for instances with strong natural clusters but less effective for instances containing weak natural clusters. We also propose exact algorithms based on the Benders decomposition for one of the IP models.

The heuristic algorithm that we propose for the q-mode problem is a local improvement algorithm that is based on a very large scale neighborhood structure. We evaluate the algorithm through a computational experiment and empirically demonstrate its effectiveness.

Exact and Heuristic Algorithms for the q-mode problem.

BY

Girish Kulkarni

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
MAY 2005

सुखार्थी वा त्यजेत विद्या, विद्यार्थी वा त्यजेत सुखम् ।
सुखार्थिनः कुतो विद्या, विद्यार्थिनः कुतः सुखम् ॥

A seeker of material comforts should not expect to gain knowledge
while a seeker of knowledge has to avoid material comforts,
as the comfort seeker does not gain knowledge
whereas a knowledge seeker does not get material comforts.

- a sanskrit "pearl of wisdom"

Biography

Girish Kulkarni was born on the the 1st of November, 1976 in the beautiful coastal city of Margao in the state of Goa, India. After graduating from high school in Mumbai, he obtained the Bachelor of Technology degree in Metallurgical Engineering and Materials Science from IIT Bombay in 1999. After completing his undergraduate studies, he worked as a software engineer at Infosys Technologies Ltd. at Pune, India for a year and then joined the Operations Research program at NC State University.

As a graduate student Girish undertook courses in Operations Research and Computer Science and the primary focus of his research was on integer programming models and exact and heuristic algorithms for combinatorial optimisation problems. During his stay at NC State University he was a member of INFORMS and served as the president of its student chapter at NC State during 2003-2004. He was invited to be a member of the intenational honor society for Operations Research, *Omega Rho*.

On the completion of his Ph.D., Girish plans to embark on a career in the industry as an Operations Research analyst.

Acknowledgments

I would like to thank my advisor, Dr. Yahya Fathi for helping me develop an interest in the field of combinatorial optimisation. Dr. Fathi has been a great mentor and working with him has been a memorable learning experience. I am indebted to him for his ideas and advice. I had the opportunity to work with him on academic and industrial problems and I benefited greatly from his insight and experience. I also wish to acknowledge his tremendous contribution in preparing this dissertation.

I wish to thank Dr. Shu-Cherng Fang, Dr. Stephen Roberts and Dr. Carla Savage for taking time out of their busy schedules to serve on my committee. They are skillful teachers with great command over their subjects. A large part of what I have imbibed during my graduate studies comes from the courses they taught. I am grateful to Dr. Savage and Dr. Roberts for their suggestions and comments, and to Dr. Fang for his insightful questions during my proposal and my final exam.

My parents, Dr. Suneela Kulkarni and Dr. Mangesh Kulkarni, have always been a great source of encouragement and support and that has motivated me to pursue doctoral research. I owe my success to their well rounded upbringing and their emphasis on discipline and excellence.

I was lucky to have Leena Wagle by my side throughout the journey of my doctoral studies. Her love and affection helped me face the challenges and focus on my priorities. She has brought joy and colour to my life.

I would like to mention some of my fellow students at NC state, Daniel Finkel, Girish Ramachandra, Rajesh Nagarajan and Chuan Lin. They were great team mates and office mates and from whom I received invaluable help while studying for the courses and qualifying exams. I am thankful to Barbara Walls who is a great program assistant for the OR program.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The q-mode problem is a combinatorial optimization problem that arises in several contexts. Clustering of categorical data as required in Data mining and manufacturing of switching cabinets in the telecommunications industry are two scenarios about which published literature exists. The q-mode problem requires the partitioning of objects into clusters and hence can be considered a special case of the partitioning problem. In the first two sections of this chapter we briefly introduce the two scenarios that we mentioned above and describe how the q-mode problem arises in these contexts. In the third section we provide a formal problem definition for the q-mode problem along with the terminology and notation that we use throughout this dissertation. In section 4 we discuss an integer programming formulation for the q-mode problem. The last section contains an overview of this dissertation.

1.1 Electronic Manufacturing.

The q-mode problem arises in electronic manufacturing in the context of a proposed scheme described in [16] and in [17], for manufacturing of *switching cabinets*. A switching cabinet is a piece of electronic equipment that consists of a collection of circuit packs mounted on slots on a frame. Every switching cabinet is configured to perform a specific task based on customer requirements. Each slot on a frame is mounted with a circuit pack from a given collection of circuit packs and the type of circuit pack assigned to the different slots determine the configuration and hence the functionality of the switching cabinet. Each switching cabinet ordered by the customer requires a specific configuration that depends on its application and hence the number of distinct configurations manufactured at a given time is extremely large.

The manufacturing of switching cabinets involves two distinct operations, circuit pack assembly and cabinet configuration. We are only concerned with cabinet configuration and testing. In order to streamline the production process a manufacturing scheme is proposed in [16, 17]. This scheme is based on manufacturing a collection of fully configured *model* switching cabinets and then modifying these model configurations so as to match the configurations demanded by the customer. This scheme is expected to improve the quality of the switching cabinets produced and also reduce the cost of manufacturing. We can maintain one or more distinct configurations among these model cabinets. Every switching cabinet configuration demanded by the customer is compared with the model configurations that we have maintained and is assigned to the model configuration that is *closest* to it.

If a particular slot in the model configuration has a circuit pack type that is different from the type required on the demanded cabinet, then the circuit pack on the model cabinet is replaced with the circuit pack required on the demanded cabinet. Thus, slot by slot the model cabinet is reconfigured by replacing circuit packs so as to obtain the configuration of the demanded cabinet. If a cabinet k is assigned to model configuration l and in reconfiguring this model cabinet to obtain the demanded cabinet k, a removal of a circuit pack is required on a particular slot j of the model cabinet, then we say that a *replacement* is required for slot j of cabinet k. The number of replacements required so that the model cabinet having configuration l is reconfigured into cabinet configuration k is known as the *replacement number* of cabinet k for model configuration l. Thus when the replacement number of a cabinet for a model is less, then time spent on reconfiguration is less and the cabinet configuration is said to be closer to the said model.

The model configuration problem is defined to be a problem of determining a collection of distinct model configurations to be used in this manufacturing scheme. For the purpose of determining the model configurations historical data on actual cabinet configurations ordered over a fixed time horizon is used. The model configurations are determined so as to minimize the total number of replacements for all the cabinet configurations in the selected historical data. In the context of the model configuration problem each switching cabinet present in the customer demand is referred to as an *order* and each slot is referred to as a *position*. The configuration of an order is determined by the circuit pack type assigned to each position on it. A cabinet having one of the model configurations is referred to as a *model*.

1.2 Data Mining.

In this section we give a brief introduction to data mining. We limit our discussion to those aspects of data mining that are relevant to the q-mode problem. Detailed explanations and descriptions of the various aspects of data mining are available in [10]. Data mining, also known in more general terms as knowledge discovery, is a term given to a collection of tasks that are performed in obtaining useful information from available data. Knowledge discovery involves the discovery of patterns in the given data allowing us to describe general properties of the data or derive inference from the current data in order to make predictions. The discovery of patterns may further lead to the discovery of association rules, i.e., attribute-value conditions that may occur frequently together in a given set of data. Two important processes involved in data mining are *classification* and *prediction*. Classification is the process of finding a set of functions that distinguish and describe the classes in the given data. This set of functions can be used to predict the class of records whose class label is unknown.

The set of functions is derived on the basis of a set of training records (i.e., a data set of records whose class labels are known). Further the functions found can be used to predict missing values in some records whose class label is known. In data mining terminology this would be known as prediction.

Unlike classification and prediction which deal with records whose classes are known, in *clustering* we investigate the records without consulting known class labels. The class labels are not present in the training data set simply because they may not be known to begin with. Clustering can be used to generate such labels. The records are clustered or grouped by maximizing intraclass similarity or minimizing interclass similarity. Each cluster that is formed can be viewed as a class of objects from which rules can be derived. Clustering also facilitates taxonomy formation, i.e., organization of the data in a hierarchy of classes that group similar events together.

Data available for data mining can be in various formats. We assume that information on which data mining needs to be done describes several entities. Thus the information available, that we call a data set, contains a set of records, and each record describes various attributes of a single entity. As an example, we may have a data set that describes the physical characteristics of mushrooms. This data set would have one record per mushroom and each record would consist of a fixed set of attributes. Further, each attribute can take values from a fixed set of categories. Thus, in the data set, every record may have attributes like *shape*, *size*, *color* etc, and the attribute *shape* can take one value from the categories *conical*, *bell* and *convex* etc.

We can formally define categorical data in the following manner. Let A_j be the j^{th} out of a total of n attributes that describe a space Ω. Each attribute has a domain $D(A_j)$. The domain of an attribute will be regarded as categorical if it is finite and unordered, i.e., if $a, b \in D(A_j)$ then either $a = b$ or $a \neq b$. Also, all $a \in D(A_j)$ will be referred to as the categories (or values) of the attribute A_j. The space Ω will be considered categorical if the domain of each of the attributes A_j

describing each record is categorical. Every record X in the categorical space Ω is described by a conjunction of attribute-value pairs $(A_1 = x_1) \cap \ldots \cap (A_n = x_n)$ where $x_1 \in D(A_1), \ldots, x_n \in D(A_n)$. Thus, each record can be represented as a vector $X = (x_1, \ldots, x_n)$ having exactly n elements.

Depending on our objective in clustering a given collection of records, different approaches can be employed to cluster these records. In this thesis we focus on an approach which is based on creating q *data modes*. The data mode has the same number of attributes as the records in the data set and the configuration of the data mode is defined by an assignment of a value, chosen from the appropriate domain, to every attribute on the data mode. Each attribute of a record can be compared to the corresponding attribute of a data mode. Every attribute in the record that does not have the same value as the corresponding attribute of the mode is said to correspond to a replacement. The total number of replacements for the record is the replacement number of that record for the mode that it was compared to. The record is assigned to the mode which corresponds to the least replacement number. Thus based on the criterion of minimizing the sum of the replacement numbers for all records the clustering of the given categorical data into q clusters effectively reduces to finding the configurations of q data modes.

One can clearly perceive that there is a one to one correspondence between the two scenarios of clustering of switching cabinets and clustering of categorical data. In fact, a collection of switching cabinets can be thought of as a data set where each switching cabinet is a record, every slot on the cabinet is an attribute and the circuit pack types that can be assigned to each slot constitute a set of categories that form the domain of that particular attribute. The model configuration problem in the manufacturing of switching cabinets and the problem of finding data modes in the context of clustering of categorical data are two applications of the q-mode problem. In the following section we present a formal definition of the q-mode problem along with a notation and terminology that is independent of the contexts in which this problem may arise.

1.3 Definition of the q-mode Problem and Notation.

In the context of q-mode problem our objective is to partition a given collection of objects into q mutually exclusive and collectively exhaustive groups so as to minimize the total *distance* from the objects to the "mode" of the cluster to which each object is assigned. We refer to each object in this context as a *record*, and each record is represented by a vector of size n. Each element (position) of this vector corresponds to an attribute of the object, and we assume that all attributes are categorical in nature, i.e., each attribute can take one of m different values (or categories) for that attribute. Without loss of generality we assume that all attributes have the same number m of possible values or categories, but this assumption can be easily removed with minor adjustments. We refer to the space of all such vectors of size n as CV^{nm}.

For a given collection (group) of records Φ in CV^{nm}, and for each value of $i = 1$ to m and $j = 1$ to n, let $F(i,j) =$ Number of records in the collection Φ that have value i in position j, and let $F^{max}(j) = \max_i F(i,j)$ and $i^*(j) = \arg\max_i F(i,j)$. Clearly $i^*(j)$ represents the category that is most frequently observed in position j among all members of the collection Φ. If more than one category tie at achieving this maximum value at position j we break the tie arbitrarily and select any one of these categories as $i^*(j)$. We now define *mode* of Φ as a vector of size n where its j^{th} element is $i^*(j)$, for all $j = 1$ to n. We denote this vector by $mod(\Phi)$ and its j^{th} element by $mod_j(\Phi) = i^*(j)$ for all j. Note that given a collection of p vectors $\Phi \subseteq CV^{nm}$ its mode vector can be determine efficiently and the corresponding computational requirement is $o(np)$ [17].

Given two vectors U^1 and U^2 of the same size we define the distance between these vectors $D(U^1, U^2)$ as the number of positions at which the two vectors are not identical; i.e., letting $d_j(U^1, U^2) = 1$ for all j such that $u_j^1 \neq u_j^2$, and $d_j(U^1, U^2) = 0$ otherwise, it follows that $D(U^1, U^2) = \sum_{j=1}^{n} d_j(U^1, U^2)$. In these expressions the notation u_j is used to represent the j^{th} element of the vector U.

For a given collection of p vectors in CV^{nm} (i.e., vectors of categorical data as defined above), say $\Phi = \{U^1, \cdots, U^p\}$, it can be shown that [17] the total distance between these vectors and their mode is smaller than or equal to the total distance between these vectors and any other vector of the same size in CV^{nm}. In other words

$$\sum_{k=1}^{p} D[U^k, mod(\Phi)] = \min_{V \in CV^{nm}} \left\{ \sum_{k=1}^{p} D[U^k, V) \right\}$$

Clearly $\sum_{k=1}^{p} D[U^k, mod(\Phi)]$ is a characteristic value of the collection Φ; we denote this value by $MD(\Phi)$ and refer to it as the *total distance of the collection* Φ *from its mode*. In context, this value is comparable to the total distance of a collection of vectors in $\Re^n$ from their geometric center.

We can interpret the distance between a vector $U^k \in \Phi$ and its mode, i.e., $D[U^k, mod(\Phi]$, as the total number of positions where a replacement of the value (change of category) is required to make this vector identical to the mode vector. This interpretation is particularly useful in the context of the switching cabinet manufacturing as described earlier or in a similar manufacturing environment. In the context of the data mining, an interpretation for $MD(\Phi)$ depends on the specific application at hand, but this value serves as a metric to measure the similarity of objects in the given collection of vectors. Obviously a smaller value for $MD(\Phi)$ implies that the members of the collection Φ are more similar to each other, and a larger value implies otherwise. For a comprehensive discussion of various measures that can be employed in this context see [12].

We are now prepared to give a formal definition of the $q-$mode problem.

The q-mode problem: Given a collection of p vectors in CV^{nm} and a positive integer q, partition these vectors into q mutually exclusive and collectively exhaustive groups (clusters) Φ_1 through Φ_q so as to minimize $\sum_{\ell=1}^{q} MD(\Phi_\ell)$.

In this context, and for ease of discussion, throughout this paper we refer to $\sum_{\ell=1}^{q} MD(\Phi_\ell)$ as *the total number of replacements associated with partitions* Φ_1 *through* Φ_q *of* Φ. Note that the 1-mode problem is simply the problem of finding the mode of the given collection of vectors, and hence it can solved efficiently as discussed earlier. For $q \geq 2$ the q-mode problem is conjectured to be NP-hard, although no proof is available in the open literature as of this writing.

1.4 An MIP Model for the q-mode Problem.

An integer programming formulation for the q-mode problem is discussed in [17]. We have included this formulation, which we refer to as (IP), below for ease of reference. A collection of p records and an integer q define an instance of the q-mode problem. Here q represents the number of clusters into which the given collection of records is to be partitioned. The records are represented in the form of a three dimensional array a_{ijk} where

$$a_{ijk} = \begin{cases} 1 & \text{if value } i \text{ is present at position } j \text{ in record } k. \\ 0 & \text{otherwise.} \end{cases}$$

The decision variables in the formulation are

$$\begin{aligned} v_{ijl} &= \begin{cases} 1 & \text{if value } i \text{ is assigned to position } j \text{ in mode } l. \\ 0 & \text{otherwise.} \end{cases} \\ y_{kl} &= \begin{cases} 1 & \text{if record } k \text{ is assigned to cluster } l. \\ 0 & \text{otherwise.} \end{cases} \\ t_{jk} &= \begin{cases} 1 & \text{if a replacement is required in position } j \text{ for record } k. \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \tag{1.1}$$

$$
(IP) \qquad
\begin{aligned}
\min \sum_j \sum_k t_{jk} & & \quad (1.2)\\
\text{s. t. } a_{ijk} y_{kl} - v_{ijl} \leq t_{jk} & \qquad \forall i, j, k, l & \quad (1.3)\\
\sum_i v_{ijl} = 1 & \qquad \forall j, l & \quad (1.4)\\
\sum_l y_{kl} = 1 & \qquad \forall k & \quad (1.5)\\
v_{ijl} \in \{0,1\}, y_{kl} \in \{0,1\}, t_{jk} \geq 0 & \qquad \forall\, i, j, k, l & \quad (1.6)
\end{aligned}
$$

In this formulation equation (1.4) ensures that exactly one value is assigned to each position in every mode, and equation (1.5) ensures that each record is assigned to exactly one cluster. Equation (1.3) is the constraint that allows for counting of replacements. Whenever a record k having value i at the j^{th} position is assigned to cluster l, i.e., $a_{ijk}y_{kl} = 1$, and the mode for cluster l does not have value i in position j, i.e., $v_{ijl} = 0$ then t_{jk} is forced to be equal to 1. The objective is to minimize the total number of replacements for the entire collection of records. This formulation has $mnq + pq + np$ variables and $mnpq + nq + p$ constraints.

A special case of the q-mode problem arises when $q = 1$, i.e., we need to find the mode of a given collection of records, i.e., the 1-mode problem. As mentioned earlier an efficient procedure with computational requirement of the order of $o(np)$, is described in Morgan et. al. [17] for solving the 1-mode problem. In this thesis we focus on the problem with $q \geq 2$.

1.5 Organization of Thesis.

In this dissertation we focus on the development of exact and heuristic algorithms for the q-mode problem. The exact algorithms are based on integer programming models for the q-mode problem. The heuristic algorithm is a local improvement algorithm that is based on a very large scale neighborhood structure. We have already introduced the q-mode problem, provided a brief explanation of the applications where this problem arises, and also discussed an integer programming model for it. Chapter 2 is a review of the literature pertaining to the q-mode problem. In Chapter 3 we propose a relaxation of the integer programming model (IP) discussed earlier in this chapter and describe a Benders' decomposition based on this relaxed model. We also propose an alternative model for the q-mode problem and further modifications to the relaxed IP model. Chapter 4 discusses the collection of problem instances that we use in the computational experiments. Chapter 5 contains the details of a computational experiment performed to evaluate the exact algorithms. Chapter 6 presents the design of a local improvement algorithm for the q-mode problem along with details of a very large scale neighborhood that it is based on. The discussion and details regarding the computational experiments performed to evaluate the local improvement algorithm are included in chapter 7. Finally, chapter 8 contains the conclusion and ideas for future research.

Chapter 2

Literature Review

In this chapter we review literature related to both contexts in which the q-mode problem arises as described earlier i.e., in electronic manufacturing and also in clustering of data having categorical attributes. There is a large body of work in the area of clustering but most of this literature is primarily related to clustering of numeric data. Several ideas originating in the literature regarding clustering of numeric data have been used in developing algorithms for clustering of categorical data but the algorithms developed for numeric data have not shown great success in clustering categorical data. In this chapter, we limit our attention to literature that pertains to algorithms for the clustering of categorical data. After this survey, we include here a brief overview of the use of large scale neighborhood structures in search algorithm for combinatorial optimization problems.

In the first section, we mainly discuss algorithms developed for the 2-model problem arising in electronic manufacturing. This section has five subsections, four different algorithms are described in the first four sections and the fifth subsection describes results of a computational experiment. Section 2.2 describes the K-modes algorithm along with a formal definition of categorical data. This algorithm is basically an extension of the K-means algorithm [15] for categorical data. In section 2.3 we review an algorithm referred to as ROCK, that has been developed for transaction data. Section

2.4 briefly describes an algorithm referred to as CACTUS and presents the ideas of summarization of the categories used in the development of this algorithm. Finally, section 2.5 we briefly introduce very large scale neighborhood search techniques.

2.1 Algorithms for the 2-model Problem.

A comprehensive study of the 2-model problem in the context of manufacturing of switching cabinets is presented in [16]. This work includes description of two main schemes for manufacturing of switching cabinets and proposes several algorithms for the 2-model problem. In the context of the two model problem every switching cabinet configuration that is ordered is referred to as an *order*. All algorithms are based on the objective function of minimizing the total number of replacements. We describe these algorithms in the following subsections.

2.1.1 Constructive Clustering Scheme.

The constructive clustering scheme begins by selecting any two of the given orders as models. Each of the remaining orders are compared to these two models. An order is assigned to the model that corresponds to less number of replacements for that order. The assignment of every given order in this manner creates two clusters. We then find the model corresponding to each of the two clusters. The total number of replacements required for both the clusters is noted. This procedure is now repeated for every possible pair of orders within the given collection of orders and the clustering corresponding to the smallest value of the total number of replacements is selected.

In this scheme, the major computational step is in selecting the best clusters. For every pair of orders in the given collection we need to calculate two models, this requires computation of the order of $o(np)$. There are a possible of $\binom{p}{2} = \frac{p(p-1)}{2}$ distinct pairs of orders that can be used as models, where n is the total number of positions on each order and p is the total number of orders in the given collection.

Thus the computational requirement of this algorithm is of the order of $o(np^3)$.

2.1.2 Iterative Clustering Scheme.

The first step in this algorithm is to select two orders from the given collection of orders that are as unlike each other as possible. This is done by comparing every possible pair of orders to each other. The pair which has a minimum number of identical positions in common is chosen and designated as the initial pair of models. The remaining orders are compared to both of these models and each order is assigned to the model which corresponds to less number of replacements for that order. This gives us two initial clusters. We now obtain the models for each cluster. If the new models obtained are different from the older models then the new models are selected as the current models and all the orders are reassigned by comparing each order with the current models and changing its assignment only if it results in less number of replacements for that order. We obtain two new models corresponding to the two new clusters and check if the models have changed. The procedure stops when the models obtained in two consecutive iterations are the same. This procedure is finite because every iteration results in a clustering that has strictly less number of replacements than the clustering in the last iteration and the number of replacements is lower bounded by 0. This procedure can, in theory, take a large number of iterations but computational experiments have shown that the algorithm generally terminates within 2 or 3 iterations.

2.1.3 Single Switch Neighborhood (SSN) Algorithm.

The SSN algorithm is a local search algorithm. In this algorithm a feasible solution of the 2-model problem is represented by a vector V. The k^{th} element of this vector indicates the cluster to which order k is assigned in the current solution. Thus every element of this vector takes value either 1 or 2. A neighbor V' of a feasible solution V is obtained by switching the cluster assignment of one of the orders. All the neighbors of a solution V constitute its neighborhood. Any feasible solution V corresponds to two clusters and the models for these clusters can be efficiently obtained. Once models are obtained the total number of replacements required for all the orders in the current solution is known and this is the cost of the solution V. The algorithm begins by selecting an initial feasible solution V^0, i.e., every order in the given collection is assigned to one of the two clusters. We now look for a neighbor of V^0 that has less cost i.e., a neighbor that has a clustering of the orders that correspond to total number of replacements that is less than that for V^0. The first neighbor found that has less cost becomes the current feasible solution. This procedure terminates when the current feasible solution has lesser cost than all its neighbors. This algorithm finds a local optimal solution to the problem and the quality of solution obtained depends upon the choice of the starting solution.

2.1.4 Variable Depth Search (VDS) Algorithm.

This algorithm is based on a neighborhood search strategy that has been introduced by Kernighan and Lin for the uniform graph partitioning problem [13]. Here, instead of switching the cluster assignment of just one of the orders (as in the SSN algorithm) a sequence of switches is performed to obtain a neighbor of the current feasible solution. Thus the neighborhood consists of the solutions obtained by all possible sequences of switches that can be performed on the current feasible solution. This neighborhood definition is less restrictive in the sense that all interim switches in the sequence of switches selected may not individually lead to an improved neighbor. But, the switches taken together in a sequence must lead to a lower cost neighbor.

This algorithm also starts with a randomly generated initial feasible solution. The cluster assignment of an order k is switched. The total number of replacements for the solution obtained after the switch, i.e., $D(k)$ is calculated. The value of $D(k)$ is calculated for all the orders in the collection. The order corresponding to the lowest value of $D(k)$ is selected, its cluster assignment is tentatively switched and the change in the total number of replacements (which we refer to as the *replacement change* in this subsection) is noted. This order becomes the first order in the sequence of switches that is built. Now we recalculate the values of $D(k)$ for all remaining orders (i.e., those that are not already in the sequence) and select the next order that gets added to the sequence. The replacement change for each order that gets added to the sequence is also noted. This procedure continues until all orders have been included as part of the current sequence. Note that the total change in the number of replacements when the first two orders in the sequence are actually switched can be calculated by adding the first two replacement changes that we have noted. Similarly, the total change in the number of replacements when the first r orders are actually switched equals the sum of the first r insertion changes that we have stored. Also, note that the sum of the replacement changes for all p orders in the sequences is zero, as performing all p switches will lead us back to the original solution.

The algorithm selects the size of the sequence to be the one that corresponds to the largest total decrease in the total number of insertions. Once the size of the sequence of switches to be performed is selected, the switches are performed as indicated in the sequence to obtain an improved neighbor. This procedure continues until the current solution is such that the associated sequences of all sizes correspond to a net increase in the number of replacements.

2.1.5 Computational Results.

Extensive computational experiments have been carried out to evaluate the performance of these four algorithms and a complete discussion of the results is given in [16]. The instances used for this computational experiment are randomly generated and the authors give details about these instances and the ideas used in generating them.

The computational experiment shows that the VDS algorithm is the best of the four algorithms in terms of the quality of solutions obtained and the SSN algorithm is a close second. The solutions obtained using VDS and SSN algorithms were considerably better than the constructive algorithms. In many of the instances, both VDS and SSN obtained identical results. In very few instances VDS was slightly better while in one instance SSN obtained a better solution. In terms of solution time the iterative clustering algorithm was the fastest, while the time taken by the enumerative clustering algorithm was comparable with the time taken by the search algorithms (VDS and SSN). Also, the VDS and SSN algorithms were used on 90 very small sized instances for which the optimal solution was obtained by exhaustive enumeration of all the solutions. In all these instances, both algorithms were able to obtain the optimal solution.

2.2 The K-modes Algorithm.

The K-means algorithm proposed in [15] for clustering of numeric data has clustering cost based on the Euclidean distance between the records. The K-modes algorithm [11] is a clustering algorithm that extends the paradigm of the K-means algorithm to domains that are categorical. The letter K in this phrase refers to the number of clusters in the data which is the same as the letter q that we use in the context of the q-mode problem. The clustering cost function used in the K-modes algorithm is the total number of mismatches of the corresponding attribute categories of the two records. Smaller number of mismatches correspond to greater similarity between the two records. This is the same cost function that we use in the q-mode problem.

Formally, for two records $X = (x_1, \ldots, x_n)$, and $Y = (y_1, \ldots, y_n)$ the degree of dissimilarity is given by $d(X,Y) = \sum_j \delta(x_j, y_j)$ where

$$\delta(x_j, y_j) = \begin{cases} 0 & x_j = y_j \\ 1 & x_j \neq y_j \end{cases}$$

The dissimilarity measure $d(X,Y)$ gives equal importance to each category of an attribute. A cost function $d_{\chi^2}(X,Y)$ that takes into account the frequencies of the categories in a data set is also defined

$$d_{\chi^2}(X,Y) = \sum_{j=1}^{n} \frac{(f_{x_j} + f_{y_j})}{f_{x_j} f_{y_j}} \delta(x_j, y_j)$$

where, f_{x_j} and f_{y_j} are the number of objects in the data set that have values x_j and y_j for attribute j. This dissimilarity measure gives more importance to rare categories than frequent ones.

In the K-modes algorithm each cluster has a representative known as the *mode* which is conceptually close to the idea of a centroid. In fact, the value of each attribute on the *mode* is the statistical mode of the values of that attribute of all the records in that cluster. The cost of each cluster C_k, i.e., $I(C_k)$, in the K-modes algorithm is the sum of the dissimilarity measure calculated for each record X_i in cluster k and the mode of that cluster Q_k i.e., $I(C_k) = \sum\limits_{\forall X_i \in C_k} \sum_j d(x_j^i, q_j^k)$, where x_j^i is the value of the j^{th} attribute of record X_i and q_j^k is the value of the j^{th} attribute of mode Q_k. The K-modes algorithm proceeds so as to create clusters that minimize the sum of the cost of all clusters i.e.,

$$\min \sum_k I(C_k) = \min \sum_{k=1}^{K} \sum_{\forall X_i \in C_k} \sum_{j=1}^{n} d(x_j^i, q_j^k)$$

The K-modes algorithm explained briefly consists of the following steps

1. Select K initial modes one for each cluster. There are several strategies that can be used to select these K initial modes, two of these strategies are explained in the paper.

2. Compare each order in the given collection with each of the K modes and calculate the dissimilarity measure $d(x_j^i, q_j^k)$. Allocate record X to that cluster k where the corresponding mode Q_k has the least value of the dissimilarity measure. Once an order is assigned to a particular cluster, the mode of that

cluster is recalculated.

3. After all records have been allocated to clusters, compare all records with all the current modes. If a record is found such that its nearest mode belongs to another cluster rather than its current one, reallocate the record to that cluster and update the modes of both clusters.

4. Repeat 3 until no record has changed clusters after a full cycle test of the whole data set.

The K-modes algorithm obtains locally optimal solutions and a solution found from this algorithm is dependent on the initial modes selected and the order of records in the dataset. This algorithm is very similar to the iterative clustering scheme that is proposed in [16] for the 2-model problem, as discussed earlier.

Experimental results show that K-modes algorithm shows very good classification performance when tested with the well known soybean data base. The authors have also included an experiment to demonstrate the scalability of the algorithm for large data set.

2.3 The ROCK Algorithm.

ROCK (**RO**bust **C**lustering using lin**K**s) proposed in [9] is an agglomerative hierarchical clustering algorithm that uses the concept of links to cluster transactional data. Transactional data is a special type of categorical data where the data set consists of records that have variable number of attributes. A cluster in a transactional data set may have a very large number of attributes present among all the records in it. But individual records in the cluster, on the average, may have a very small number of these attributes. Example of a transaction is a set of items purchased by a single individual at a grocery store. A database of such records is called a market basket database. Records that belong to a particular cluster because they represent individuals that have similar buying patterns, may contain a very small number of items from a very large set of items that defines the cluster. Thus every transaction in a cluster does not contain all the attributes but a small subset of them. Thus it is possible that a pair of items in a cluster have very few items in common but they are *linked* by a number of other transactions in the cluster that have a substantial number of items in common with the two transactions. Further the number of items that define different clusters may not have uniform sizes.

The notion of links between the various transactions can be explained as follows. A pair of transactions are defined to be neighbors if the degree of their similarity is greater than some threshold value. The degree of similarity can be quantified using any function or also from some sort of a similarity table given by a domain expert. The number of links ($links(t_s, t_r)$) between a pair of transactions, t_s and t_r, is then the number of common neighbors for those transactions. Transactions belonging to the same cluster will have in general a large number of common neighbors and consequently more links.

This concept of links is used to develop a cost function on the basis of which clustering is performed. A cluster should be such that all pairs of transactions in it must have a large number of links. In other words, two transactions t_s and t_r that belong to the same cluster should have a high value of $link(t_s, t_r)$. At the same time two transactions t_s and $t_{r'}$ that belong to different clusters should have a low value of $link(t_s, t_{r'})$. This leads the authors to propose the following cost function

$$E_l = \sum_{l=1}^{q} n_l \sum_{t_s, t_r \in C_l} \frac{link(t_s, t_r)}{n_l^{1+2f(\theta)}}$$

The cost function not only tries to maximize $link(t_s, t_r)$ between all pairs of transactions in a cluster but it is also able to distinguish between clusters that need to be merged together and those that need to be separate in an agglomerative algorithm. This is very important because the cost function is used in the cost of a hierarchical algorithm which proceeds by merging of clusters. Thus a cost function that simply maximizes the sum of the links between pairs of transactions would not prevent all the transactions to become part of a single cluster. To prevent this from happening the total number of links from any pair of transactions in a cluster l is divided by the total expected pairs of links in the cluster which is estimated to be $n_l^{1+2f(\theta)}$. Here $f(\theta)$ is a function such that the cluster C_k has approximately $n_l^{f(\theta)}$ links. The function $f(\theta)$ depends on the data set as well as on the type of clusters that we are trying to obtain. Dividing by the expected number of links in C_k prevents points with very few links between them from getting assigned to the same cluster. The other main issue here is the determination of the function $f(\theta)$. The authors have found that if the clusters in the instance are well defined then even inaccurate but reasonable estimates work well in practice.

The ROCK algorithm begins by accepting as input a set of sampled records randomly drawn from the original data set and the number of desired clusters K. The ROCK algorithm finds optimal clusters based on these sample records. Initially, each record is a cluster. At each iteration the clusters are stored in decreasing order of the value of the cost function. The algorithm proceeds by selecting the cluster with the largest value of the cost function and merging it with another cluster that it is closest to. The closeness of the two clusters is decided based on the value of a *goodness measure* which itself is based on the cost function. The number of clusters thus decrease at each iteration and the algorithm is stopped when there are K clusters left or when there are no links between the clusters formed at the end of a particular iteration. The clusters formed using just the sampled records are then used to assign the remaining records to the appropriate clusters.

The ROCK algorithm is run on three data sets taken from the UCI machine learning repository: Mushroom data set, Congressional Votes data set and the US mutual Funds data set. The ROCK algorithm has limited success in the Congressional votes data set. This data set contains two natural classes of records (Democrats and Republicans). The ROCK algorithm is able to cluster this data set into two clusters, one containing a large number of democrats and the other containing a large number of republicans, but it is unable to obtain clusters that separate out the two classes. The Mushroom data set has two classes Edible and Poisonous. The ROCK algorithm clusters the data set into 21 clusters of which 20 clusters contain either all poisonous mushrooms or all edible mushrooms, only one cluster contains mushrooms of both classes.

2.4 The CACTUS Algorithm.

The CACTUS algorithm is proposed in [7]. This paper introduces a novel formalization of a cluster for categorical attributes by generalizing the definition of a cluster for numerical attributes. This algorithm is unique in the sense that the set of values that the attributes can take are partitioned into disjoint clusters and these clusters are used to assign the records in the data set to different clusters.

CACTUS (**CA**tegorical **C**lus**T**ering **U**sing **S**ummaries) has two important characteristics. First, the algorithm requires only two scans of the dataset, and hence is very fast and scalable. Second the algorithm creates *summaries* and each record is assigned to clusters based on these summaries.

To define the summaries of the given dataset the authors introduce and define various terms which we explain briefly below. Let $A_1, \ldots, A_n$ be a set of categorical attributes with domains $D_1, \ldots, D_n$, respectively. Let the dataset D be a set of records where each record t contains values for each of the n attributes. i.e., $t : t \in D_1 \times \cdots \times D_n$. We call $S = S_1 \times \cdots \times S_n$ an *interval region* if for all $i \in \{1, \ldots, n\}, S_i \subseteq D_i$. Let $a_i \in D_i$ and $a_j \in D_j$, $i \neq j$ then the *support* $\sigma(a_i, a_j)$ of the attribute value pair (a_i, a_j) with respect to dataset D is defined as the number of records in the data set D that have the i^{th} attribute equal to a_i and the j^{th} attribute equal to a_j, i.e., $\sigma(a_i, a_j) = |\{t \in D : t.A_i = a_i \text{ and } t.A_j = a_j\}|$. A record $t = \langle t.A_1, \ldots, t.A_n \rangle \in D$ is said to *belong* to the region S if for all $i \in \{1, \ldots, n\}$, $t.A_i \in S_i$. The *support* $\sigma(S)$ of

S is the number of records in D that belong to S.

Now, we define the concepts *strong connection* and *similarity*. If all the attributes are independent then all attribute values within an attribute are equally likely and an expected value can be calculated for the number of records which have a particular combination of attribute values. Intuitively clusters occur in a dataset because the attribute values are not equally likely. If there are a large number of records have attribute value pair (a_i, a_j) then they are said to be *connected* and if the support $\sigma(a_i, a_j)$ exceeds a certain threshold then these two attribute values are said to be *strongly connected*. Further, let $S_i \subset D_i$ and $S_j \subset D_j$, $i \neq j$, be two sets of attribute values. An element $a_i \in S_i$ is strongly connected with S_j if, for all $x \in S_j$, a_i and x are strongly connected. S_i and S_j are said to be strongly connected if each $a_i \in S_i$ is strongly connected with S_j and each $a_j \in S_j$ is strongly connected with S_i. Also, let $a_1, a_2 \in D_i$. The similarity $\gamma^j(a_1, a_2)$ between a_1 and a_2 with respect to $A_j (j \neq i)$ is defined as follows. $\gamma^j(a_1, a_2) = |\{x \in D_j : \sigma(a_1, x) > 0 \text{ and } \sigma(a_2; x) > 0\}|$. These concepts of strong connection and similarity between attribute values is used to determine *summaries* required for the CACTUS algorithm.

The summary information is of two types: (1) inter-attribute summaries and (2) intra-attribute summaries. The inter-attribute summaries (Σ_{IJ}) consist of all strongly connected attribute value pairs where each pair has attribute values from different attributes, i.e.,

$\Sigma_{IJ} = \{\Sigma_{ij} : i, j \in \{1, \ldots, n\} \text{ and } i \neq j\}$ where,

$\Sigma_{ij} = \{(a_i, a_j, \sigma^*(a_i, a_j) : a_i \in D_i, a_j \in D_j, \text{ and } \sigma^*(a_i, a_j) > 0\}$.

The intra-attribute summaries consist of similarities between attribute values of the same attribute i.e.,

$\Sigma_{II} = \{\Sigma_{ii}^j : i, j \in \{1, \ldots, n\} \text{ and } i \neq j\}$ where,

$\Sigma_{ii} = \{(a_{i1}, a_{i2}, \gamma^j(a_{i1}, a_{i2}) : a_{i1}, a_{i2} \in D_i, \text{ and } \gamma^j(a_{i1}, a_{i2}) > 0\}$.

Further, for $i = 1, \ldots, n$, $C_i \subseteq D_i$, $|C_i| > 1$, and $\alpha > 1$, $C = \langle C_1, \ldots, C_n \rangle$ is a cluster over $\{A_1, \ldots, A_n\}$ if the following three conditions are satisfied.

1. For all $i, j \in \{1, \ldots, n\}, i \neq j, C_i$ and C_j are strongly connected.

2. For all $i, j \in \{1, \ldots, n\}, i \neq j$, there exists no $C'_i \supset C_i$ such that for all $j \neq i$, C'_i and C_j are strongly connected.

3. The support $\sigma(C)$ of C is at least α times the expected support of C under the attribute-independence assumption.

The CACTUS algorithm is based on the above concepts and consists of three phases, Summarization, Clustering and Validation.

1. Summarization: This phase consists of calculation of inter-attribute and intra-attribute summaries and the paper describes effecient techniques to calculate these summaries for a given data set.

2. Clustering: This phase involves the use of summaries to develop disjoint sets of values which have been defined as *interval regions*. These interval regions have to satisfy the three conditions given above. Thus in this step the domain of attributes is partitioned into disjoint clusters.

3. Validation: In this phase each record from the data set is individually scanned. The interval regions to which the values present in the record belong determines the cluster to which the record should be assigned. If membership of some some cluster is below a threshold value then that cluster is dropped.

The CACTUS algorithm is tested on a synthetic data set having 1 million records. Each record has 10 attributes and the number of attribute values for each attribute equal 100. The records are created in such a manner that there are strong clusters in the data set. The CACTUS algorithm is able to identify the clusters of each record correctly and in a reasonable amount of computer time. CACTUS is also applied to a data set that contains a combination of two sets of bibliographic entries. The results from this application show that CACTUS finds intuitively meaningful clusters in the data set.

Here we conclude our survey pertaining to algorithms that can be used for solving the q-mode problem. In the next section we present a brief overview of very large scale neighborhood structures and the techniques used for searching these neighborhoods in the context of designing search algorithms for combinatorial optimization problems.

2.5 Very Large Scale Neighborhood Search Techniques.

The structure of the neighborhood is by far the most important feature that affects the quality of the local optima obtained via neighborhood search algorithms in combinatorial optimization problems (also referred to in literature as local search algorithms). It is generally desirable to have large neighborhoods so as to improve the possibility of having good quality local optima. On the other hand, a large amount of time may have to be spent on searching large neighborhoods leading to large amount of time being spent for every run. One generally desires to perform several runs of an algorithm with different starting points so as to increase the chance of finding a global optimal solution, and having large execution times for each run leads to fewer runs. Thus a large neighborhood may not necessarily produce an effective heuristic unless we can search the neighborhood efficiently.

Very large scale neighborhood (VLSN) search algorithms can be considered to belong to three broad classes : (1) variable-depth methods in which large neighborhoods are searched heuristically, (2) large neighborhoods in which the neighborhoods are searched using network flow techniques or dynamic programming, and (3) large neighborhoods induced by restrictions of the original problem that are solvable in polynomial time. A comprehensive survey of algorithms belonging to all three classes can be found in Ahuja et al. [5]. Below, we introduce relevant literature on search algorithms for the first two of the above three classes.

The heuristic proposed by Lin and Kernighan [14] is one of the earliest variable depth methods and many variable depth methods for the traveling salesman problem can be considered as generalizations of this heuristic. Further extensions and generalizations of the ideas of Lin and Kernighan by Glover [8] have lead to a structured class of variable depth methods called *ejection chains*. Variable depth methods and ejection chains have been used to develop heuristic methods that have been successful in obtaining good quality solutions for a wide variety of combinatorial optimization problems. The second class of VLSN search algorithms are those where the neighborhoods are searched using network flow techniques. The techniques used can be classified as follows (i) minimum cost cycle finding methods (ii) shortest path or dynamic programming based methods and (iii) methods based on minimum cost matchings or assignments. Thompson and Orlin [19] have proposed a cyclic exchange neighborhood for solving partitioning problems, a large sub-class of problems that find significant applications in logistics, manufacturing telecommunications and scheduling. The large neighborhood formed out of cyclic exchanges has been used by Ahuja et al. [3], along with techniques to identify an improved neighbor (based on finding the minimum cost cycle), for solving the capacitated minimum spanning tree problem. A more detailed discussion on search algorithms based on cyclic exchange neighborhoods along with a network optimization based methodology to efficiently search the neighborhood is given in [2].

Chapter 3

MIP models and Benders' Decomposition.

In this chapter we present theoretical developments related to MIP formulations for the q-mode problem. These developments are directed towards designing an exact algorithm for the q-mode problem that can solve relatively large size instances. In the first section of this chapter we demonstrate that the MIP formulation (IP) can be relaxed and the resulting formulation (IPR) can be used to obtain an optimal solution of the q-mode problem. In the second section we discuss a Benders decomposition for the formulation (IPR). In the third section we propose an alternate MIP formulation for the q-mode problem. In the same section we also present a modification for the formulation (IPR) with the aim of improving the optimal objective value of its LP relaxation. This modification leads to considerable computational efficiency when solving this model using a branch and bound algorithm.

3.1 A Relaxed MIP Formulation for the q-mode Problem.

We consider the mixed integer programming (MIP) formulation (IP) discussed in section 1.4 and replace the binary restriction on v_{ijl}, i.e., $v_{ijl} \in \{0, 1\}$, with a simple non-negativity restriction, $v_{ijl} \geq 0$. The resulting model is still an MIP formulation as it has y_{kl} variables that are constrained to be binary. Clearly this MIP model is a relaxation of the formulation (IP) and we refer to this formulation as (IPR). The model (IPR) is given below and the notation used in this model is the same as that used for model (IP).

$$(IPR) \qquad \begin{aligned} \min \sum_j \sum_k t_{jk} & & & (3.1) \\ \text{s. t. } a_{ijk} y_{kl} - v_{ijl} \leq t_{jk} & \qquad \forall i, j, k, l & & (3.2) \\ \sum_i v_{ijl} = 1 & \qquad \forall j, l & & (3.3) \\ \sum_l y_{kl} = 1 & \qquad \forall k & & (3.4) \\ v_{ijl} \geq 0, t_{jk} \geq 0, y_{kl} \in \{0, 1\} & \qquad \forall\ i, j, k, l & & (3.5) \end{aligned}$$

In this section we show that (IPR) has an optimal solution in which all v_{ijl} variables are integer (binary); hence this solution is also an optimal solution for (IP). We further show that every basic feasible solution of a corresponding LP model, obtained from (IPR) by fixing the y_{kl} variables at any feasible values, has integer values for all v_{ijl} variables. This property is important in the context of obtaining an optimal solution of (IPR) using a branch and bound algorithm.

3.1.1 Separability of (IPR) for a Fixed Value of y.

We begin by assuming that we have an instance of the q-mode problem and also an appropriate set of binary values for the y_{kl} variables, i.e., we fix the values of all variables of type y_{kl} such that they satisfy constraints (3.4) and (3.5). In other words, we have $y_{kl} \in Y \ \forall\ k, l$ where Y is defined as $Y = \{y_{kl} | y_{kl} \in \{0, 1\}, \sum_l y_{kl} = 1, \forall\ k, l\}$. Henceforth, we shall use y to collectively refer to all the variables of type y_{kl}. Once we fix $y \in Y$, all constraints of type (3.2) can be rewritten as $t_{jk} + v_{ijl} \geq a_{ijk}y_{kl}$, where $a_{ijk}y_{kl}$ is either 0 or 1. Also, since constraint (3.4) is already satisfied, it can be dropped. The reduced formulation (IPR_y) thus obtained is given below. We use $\nu(y)$ to represent the optimal value of the problem as a function of y.

$$
(IPR_y) \qquad
\begin{aligned}
\nu(y) = min \sum_j \sum_k t_{jk} & & \qquad (3.6)\\
\text{s.t. } t_{jk} + v_{ijl} \geq a_{ijk}y_{kl} & \qquad \forall\ i, j, k, l & \qquad (3.7)\\
\sum_i v_{ijl} = 1 & \qquad \forall\ j, l & \qquad (3.8)\\
v_{ijl} \geq 0, t_{jk} \geq 0 & \qquad \forall\ i, j, k, l & \qquad (3.9)
\end{aligned}
$$

It can be seen by inspection that (IPR_y) is separable in variable j as shown below. Hence, we can split the formulation (IPR_y) into n *subproblems*, i.e., for each $j = 1$ to n we have

$$
(IPR_y^j) \qquad
\begin{aligned}
\nu_j(y) = min \textstyle\sum_k t_{jk} & \\
\text{s.t. } t_{jk} + v_{ijl} \geq a_{ijk}y_{kl} & \qquad \forall\ i, k, l\\
\textstyle\sum_i v_{ijl} = 1 & \qquad \forall\ l\\
v_{ijl} \geq 0, t_{jk} \geq 0 & \qquad \forall\ i, k, l
\end{aligned}
$$

and $\nu(y) = \sum\limits_j^n \nu_j(y)$.

As defined before, let Φ represent the set of all records and let Φ_l represent the set of all records assigned to cluster l. We know that $y \in Y$ implies that each record k is assigned to exactly one cluster l. Thus, $\Phi_1 \cup \Phi_2 \cup \cdots \cup \Phi_q = \Phi$ and $\Phi_{l_1} \cap \Phi_{l_2} = \emptyset \ \forall\ l_1, l_2$. We can re-write the objective function $\sum_k t_{jk}$ as $\sum_l \sum_{k \in \Phi_l} t_{jk}$ by bringing together t_{jk} variables corresponding to records belonging to the same cluster.

Let record k be assigned to cluster l', then $y_{kl'} = 1$ and $y_{kl} = 0 \ \forall\ l \neq l'$. Therefore, for all clusters $l \neq l'$, we can replace $a_{ijk}y_{kl}$ with 0 and the constraints $t_{jk} + v_{ijl} \geq a_{ijk}y_{kl}$ become $t_{jk} + v_{ijl} \geq 0$. Since, $v_{ijl} \geq 0$ and $t_{jk} \geq 0$, these constraints are satisfied by all feasible values of v_{ijl} and t_{jk}. Thus these constraints are redundant and can be dropped. After dropping all constraints of the form $t_{jk} + v_{ijl} \geq a_{ijk}y_{kl}$ where $y_{kl} = 0$ i.e., $k \notin \Phi_l$, and rewriting the remaining constraints that correspond to $y_{kl} = 1$, i.e., $k \in \Phi_l$, as $t_{jk} + v_{ijl} \geq a_{ijk}$, we can write IPR_y^j as

$$\begin{aligned} & min \textstyle\sum_l \sum_{k \in \Phi_l} t_{jk} & \\ \text{s.t. } & t_{jk} + v_{ijl} \geq a_{ijk} & \forall\ i, l, k \in \Phi_l \\ & \textstyle\sum_i v_{ijl} = 1 & \forall\ l \\ & v_{ijl} \geq 0, t_{jk} \geq 0 & \forall\ i, k, l \end{aligned}$$

Clearly the above model is separable in variable l. Hence, each of the subproblems (IPR_y^j) is separable into q subproblems which implies that (IPR_y) is separable into nq subproblems. Thus for each $j = 1$ to n and each $l = 1$ to q we have a subproblem (IPR_y^{jl}) as follows

$$(IPR_y^{jl}) \qquad \begin{aligned} \nu_{jl}(y) = & min \textstyle\sum_{k \in \Phi_l} t_{jk} & \\ \text{s.t. } & t_{jk} + v_{ijl} \geq a_{ijk} & \forall\ i, k \in \Phi_l \\ & \textstyle\sum_i v_{ijl} = 1 & \\ & v_{ijl} \geq 0, t_{jk} \geq 0 & \forall\ i, k \end{aligned}$$

where $\nu(y) = \sum_j \sum_l \nu_{jl}(y)$. In its fully separated form (IPR_y) can now be written as

$$(IPR_y) \qquad \sum_j \sum_l \left(\begin{array}{cc} min \sum_{k \in \Phi_l} t_{jk} & \\ \text{s.t. } t_{jk} + v_{ijl} \geq a_{ijk} & \forall\, i, l, k \in \Phi_l \\ \sum_i v_{ijl} = 1 & \forall\, j, l \\ v_{ijl} \geq 0, t_{jk} \geq 0 & \forall\, i, j, k, l \end{array} \right)$$

3.1.2 Formulation (IPR_y^{jl}) has an Integer Optimal Solution.

In this subsection we prove that (IPR_y^{jl}) has an integer optimal solution. We also derive a formula to calculate the optimal objective value for any subproblem (IPR_y^{jl}) and finally show that (IPR) is a valid formulation for the q-mode problem. These results are particularly useful in the context of the Benders' decomposition algorithm discussed in section 3.2. First let us consider the subproblem (IPR_y^{jl}) that we have introduced earlier.

$$\begin{array}{lll} & min \sum_{k \in \Phi_l} t_{jk} & \\ (IPR_y^{jl}) & \text{s.t. } t_{jk} + v_{ijl} \geq a_{ijk} & \forall\, i, k \in \Phi_l \\ & \sum_i v_{ijl} = 1 & \\ & v_{ijl} \geq 0, t_{jk} \geq 0 & \forall\, i, k \end{array}$$

We know that a_{ijk} takes values 0 or 1 and hence the constraints $t_{jk} + v_{ijl} \geq a_{ijk}$ can be written as one constraint of type $t_{jk} + v_{ijl} \geq 1$ and $m - 1$ constraints of type $t_{jk} + v_{ijl} \geq 0$. The proof proceeds by showing that we can transform problem (IPR_y^{jl}) by dropping constraints of type $t_{jk} + v_{ijl} \geq 0$ and rewriting the remaining "≥ 1" constraints as "$= 1$" constraints without affecting the optimal solution. Then we show that the optimal solution to this transformed problem is integer. We also need the following definition for $F_l(i, j)$. For all records assigned to cluster l, $F_l(i, j)$ is the number of records having value i in position j. Therefore, $F_l(1, j) + F_l(2, j) + \cdots + F_l(m, j) = |\Phi_l|$ = number of records assigned to cluster l, and $F_l(i, j) = \sum_{k \in \Phi_l} a_{ijk}$.

Theorem 3.1. *(IPR_y^{jl}) has an integer optimal solution and the optimal objective value is $|\Phi_l| - \max_i F_l(i,j)$.*

Proof : Every record k corresponds to a group of m constraints of which exactly one constraint is of the form $t_{jk} + v_{ijl} \geq 1$ and $m-1$ constraints are of the form $t_{jk} + v_{ijl} \geq 0$. In problem IPR_y^{jl}, apart from these m constraints no other constraints affect the value of the corresponding variable t_{jk}. Since $t_{jk} \geq 0$ and $v_{ijl} \geq 0$, it follows that constraints of type $t_{jk} + v_{ijl} \geq 0$ are satisfied by every feasible value of t_{jk} and v_{ijl} and hence such constraints are redundant. We focus on the only remaining constraint in the group, i.e., $t_{jk} + v_{ijl} \geq 1$, and show that this constraint must be satisfied as an equality at every optimal solution for this problem.

Let us assume that $\{t_{jk}^*, v_{ijl}^*\}$ represents an optimal solution to IPR_y^{jl} and that for some i' and k' the optimal solution is such that we have $t_{jk'}^* + v_{i'jl}^* > 1$. $\therefore t_{jk'}^* > 1 - v_{i'jl}^*$.

Let, $t'_{jk'} = 1 - v_{i'jl}^*$. It follows that $t'_{jk'} < t_{jk'}^*$, hence we have

$$\sum_{\substack{k\in\Phi_l,\\ k\neq k'}} t_{jk}^* + t'_{jk'} < \sum_{\substack{k\in\Phi_l,\\ k\neq k'}} t_{jk}^* + t_{jk'}^* = \sum_{k\in\Phi_l} t_{jk}^*$$

Also, $t'_{jk'} \geq 0$ and hence satisfies all constraints. Thus $t_{jk'}^*$ cannot be part of an optimal solution, and this implies that for any record k if the corresponding variable t_{jk}^* is optimal, then we can write $t_{jk}^* = 1 - v_{ijl}^*$, where i is such that $a_{ijk} = 1$.

We can thus transform problem IPR_y^{jl} by dropping the "≥ 0" constraints and rewriting the remaining "≥ 1" constraints as equality. Let us call this transformed problem as (IPD_y^{jl}).

We have already shown that if t_{jk}^* is optimal then for every record k assigned to cluster l there is a value i_k s.t. $t_{jk}^* = 1 - v_{i_k jl}^*$ and $a_{i_k jk} = 1$. It follows that

$$\begin{aligned}\sum_{k\in\Phi_l} t^*_{jk} &= \sum_{k\in\Phi_l}(1-v^*_{i_kjl})\\ &= \sum_{\substack{k\in\Phi_l,\\ a_{1jk}=1}}(1-v^*_{1jl}) + \sum_{\substack{k\in\Phi_l,\\ a_{2jk}=1}}(1-v^*_{2jl}) + \ldots + \sum_{\substack{k\in\Phi_l,\\ a_{mjk}=1}}(1-v^*_{mjl})\\ &= \sum_i F_l(i,j)(1-v^*_{ijl})\end{aligned}$$

Problem (IPD^{jl}_y) becomes $\begin{pmatrix} \min & \sum_i F_l(i,j)(1-v_{ijl}) \\ \text{s.t.} & \sum_i v_{ijl}=1, v_{ijl}\geq 0 \end{pmatrix}$

$$\begin{aligned}\text{Now, } \min\sum_i F_l(i,j)(1-v_{ijl}) &= \sum_i F_l(i,j) - \max\sum_i F_l(i,j)v_{ijl}\\ &= |\Phi_l| - \max\sum_i F_l(i,j)v_{ijl}\end{aligned}$$

Since, $v_{ijl}\geq 0$ and $\sum_i v_{ijl}=1$, $\sum_i F_l(i,j)v_{ijl}$ is actually a convex combination of the m numbers represented as $F_l(i,j)$, for $i=1$ to m. Thus $\max\sum_i F_l(i,j)v_{ijl} = max_i F_l(i,j)$ and $|\Phi_l| - \max\sum_i F_l(i,j)v_{ijl} = |\Phi_l| - \max_i F_l(i,j)$. Thus the optimal solution to (IPD^{jl}_y) is such that $v^*_{ijl}=1$ when $i = arg\max_i F_l(i,j)$ and 0 otherwise. The optimal objective value is $|\Phi_l| - \max_i F_l(i,j)$. Thus (IPD^{jl}_y) has an integer optimal solution, which in turn indicates that (IPR^{jl}_y) has an integer optimal solution.
■

Further, we can see that when $arg\max_i F_l(i,j)$ is unique, (IPR^{jl}_y) has a unique optimal solution and it is integer. When $arg\max_i F_l(i,j)$ is not unique then the total number of integer optimal solutions for (IPR^{jl}_y) equals the number of values i such that $i = arg\max_i F_l(i,j)$. In this case (IPR^{jl}_y) has an infinite number of non-integer optimal solutions as any convex combination of the integer optimal solutions will be optimal. This result directly leads to the following theorem stating that (IPR) has an integer optimal solution.

Theorem 3.2. *(IPR) has an optimal solution in which all v_{ijl} variables are binary.*

Proof: From section 3.1.1 we know that problem (IPR_y) is separable into nq subproblems represented as (IPR_y^{jl}). Hence the collection of variables in every optimal solution of (IPR_y) corresponding to the subproblem IPR_y^{jl} will be an optimal solution to that subproblem. Further from theorem 3.1 we know that every subproblem (IPR_y^{jl}) has an optimal solution where all variables are integer. Hence we can conclude that IPR_y for $y \in Y$, has an optimal solution where all variables have integer values.

Further, we know that (IPR_y) is obtained by fixing the values of the y_{kl} variables with a feasible set of values. Hence $v(IPR) = \min_{y \in Y} v(IPR_y) = v(IPR_{y*})$, where $y* = \arg\min_{y \in Y} v(IPR_y)$. We have already shown that IPR_{y*} has an integer optimal solution, hence IPR has an optimal solution in which all the variables are integer. ■

3.1.3 Integrality of Basic Feasible Solutions of (IPR_y)

In this subsection we show that every basic feasible solution of the formulation (IPR_y) has integer values for all variables. We begin by first proving a property of the subproblem (IPR_y^{jl}). Then we use this property to show that that all basic feasible solutions of (IPR_y^{jl}) have integral values for all variables. These two results are used to demonstrate the integrality of basic feasible solutions of (IPR_y), and further to show that a branch and bound algorithm that is based on the LP relaxation of (IPR) finds an optimal solution for the IP model of the q-mode problem. We begin by re-writing the formulation (IPR_y^{jl}) in "*standard form*" by adding slack variables r_{ijkl}.

$$
(IPR_y^{jl}) \qquad
\begin{aligned}
& \min \sum_{k\in\Phi_l} t_{jk} && && (3.10)\\
& \text{s.t. } t_{jk} + v_{ijl} - r_{ijkl} = a_{ijk} && \forall\, i,k \in \Phi_l && (3.11)\\
& \sum_i v_{ijl} = 1 && && (3.12)\\
& v_{ijl} \geq 0, t_{jk} \geq 0, r_{ijkl} \geq 0 && \forall\, i,k && (3.13)
\end{aligned}
$$

Clearly, this model has $m|\Phi_l| + 1$ constraints and $m|\Phi_l| + |\Phi_l| + m$ variables ($|\Phi_l|$ variables of type t_{jk}, m variables of type v_{ijl} and $m|\Phi_l|$ variables of type r_{ijkl}).

Lemma 3.1. *Every basic feasible solution of problem (IPR_y^{jl}) has exactly one variable $v_{i'jl}$ such that $v_{i'jl} = 1$ and $v_{ijl} = 0\ \forall\ i \neq i'$.*

Proof : There are two possible cases, the first case, where the basic feasible solution has at least one variable of type t_{jk} that is equal to 0 and the second case where all variables of type t_{jk} are strictly greater than 0.

Case 1 : At least one variable of type $t_{jk} = 0$. i.e., $t_{jk} = 0$ for at least one $k \in \Phi_l$

It is clear from constraint (3.11) that every variable of type t_{jk} is present in m constraints of type $t_{jk} + v_{ijl} - r_{ijkl} = a_{ijk}$. One of these constraints has $a_{i'jk} = 1$, and the remaining $m-1$ constraints, that correspond to all $i \neq i'$, have $a_{ijk} = 0$. Hence, corresponding to the variable of type t_{jk} that is equal to zero, we have one constraint of type $v_{i'jl} - r_{i'jkl} = 1$. Since $v_{ijl} \geq 0, r_{ijkl} \geq 0\ \forall\ i,k$ and $\sum_i v_{ijl} = 1$, the only feasible solution is $v_{i'jl} = 1, r_{i'jkl} = 0$.

Therefore, if $t_{jk} = 0$ for at least one $k \in \Phi_l$, then the only feasible solution is such that $v_{i'jl} = 1$ and $v_{ijl} = 0\ \forall\ i \neq i'$.

Case 2 : $t_{jk} > 0\ \forall k \in \Phi_l$.

As mentioned before, for each variable of type t_{jk} we have $(m-1)$ constraints of type $t_{jk} + v_{ijl} - r_{ijkl} = 0$. Hence for each variable of type t_{jk} such that $t_{jk} > 0$, we have $(m-1)$ variables of type r_{ijkl} such that $r_{ijkl} > 0$. Thus each $t_{jk} > 0$ corresponds to m variables that are strictly greater than 0. Therefore when all variables of type $t_{jk} > 0$ then we must have $m|\Phi_l|$ variables strictly greater than 0 at every feasible solution.

We know that for $\{t_{jk}, v_{ijl}, r_{ijkl}\}$ to be a basic feasible solution of (IPR_y^{jl}) we need at least $(m|\Phi_l|+|\Phi_l|+m)$ - $(m|\Phi_l|+1)$ = $(m+|\Phi_l|$-1) variables in the solution to be equal to 0. In other words, we can have no more than $(m|\Phi_l|+1)$ non-zeros. Since we already have $m|\Phi_l|$ non-zeros (all the t_{jk} and r_{ijkl} variables) and we have to satisfy constraint (3.12), we can have exactly one of the variables of type $v_{ijl} > 0$. Hence, when all $t_{jk} > 0$ every basic feasible solution will have exactly one $v_{i'jl} = 1$ and $v_{ijl} = 0 \; \forall \; i \neq i'$. ■

Lemma 3.2. *Every basic feasible solution of (IPR_y^{jl}) has integer values for all the variables.*

Looking at the standard form formulation of (IPR_y^{jl}) one can see that every record k corresponds to m constraints of type (3.11). We refer to this set of m constraints as C_k. From the results of lemma 3.1 we see that for a record k in any solution of (IPR_y^{jl}) exactly two possibilities exist. The first case is where record k is such that $a_{i'jk} = 1$, $v_{i'jl} = 1$ and $a_{ijk} = v_{ijl} = 0 \; \forall i \neq i'$. The second case is where the record k is such that, there are distinct i', i'' such that $a_{i''jk} = 1$ and $a_{ijk} = 0 \; \forall i \neq i''$, and $v_{i'jl} = 1$ and $v_{ijl} = 0 \; \forall i \neq i'$. We look at both cases separately.

Case 1 : $a_{i'jk} = 1$, $v_{i'jl} = 1$ and $a_{ijk} = v_{ijl} = 0 \; \forall i \neq i'$.

In this case, each constraint in the constraint set C_k will take the form $t_{jk} = r_{ijkl}$. Any $t_{jk} \geq 0$ satisfies all these constraints. Let A be a feasible solution such that the variable t_{jk} takes the value $t > 0$. Note that when $t_{jk} = t$ then $r_{ijkl} = t \; \forall i$. Let B be a solution where $t_{jk} = r_{ijkl} = 2t$ and C be a solution where $t_{jk} = r_{ijkl} = 0$. All other corresponding variables in solutions A, B and C have identical values. From the definitions of the solutions A, B and C, it is clear that solution A is a convex combination of solutions B and C and hence it cannot be a basic solution. Thus, if a basic feasible solution has a record k such that $a_{ijk} = v_{ijl} = 1$, then the corresponding variable t_{jk} and variables r_{ijkl} are equal to 0.

Case 2 : $a_{i''jk} = 1$ and $a_{ijk} = 0 \ \forall i \neq i''$ and $v_{i'jl} = 1$ and $v_{ijl} = 0 \ \forall i \neq i'$.
In this case, the constraints from the constraint set C_k will be of three types

1. One constraint of type $t_{jk} - r_{i'jkl} + 1 = 0$, that corresponds to $v_{i'jl} = 1, a_{i'jk} = 0$.
2. One constraint of type $t_{jk} - r_{i''jkl} - 1 = 0$ that corresponds to $v_{i''jl} = 0, a_{i''jk} = 1$ and
3. A set of constraints of type $t_{jk} = r_{ijkl} \ \forall i \neq i', i''$ that correspond to $v_{ijl} = 0, a_{ijk} = 0$.

Any $t_{jk} \geq 1$ satisfies these constraints. We consider a solution A such that $t_{jk} = t > 1$, a solution B such that $t_{jk} = 2t - 1$ and a solution C such that $t_{jk} = 1$. The table below indicates the value of the other variables. The second column gives the values of variables in solution A, while the third and the fourth column give the values of the variables in solutions B and C respectively. The second row labeled "$t_{jk} =$" gives the values that the variable t_{jk} takes in solutions A, B and C. Similarly the third, fourth and the fifth row give the values that variables $r_{i'jkl}$, $r_{i''jkl}$ and r_{ijkl} respectively take in solutions A, B and C. Note that r_{ijkl} actually represents all the variables r_{ijkl} where $i \neq i', i''$

	A	B	C
$t_{jk} =$	t	$2t-1$	1
$r_{i'jkl} =$	$t+1$	$2t$	2
$r_{i''jkl} =$	$t-1$	$2t-2$	0
$r_{ijkl} =$ $(\forall i \neq i', i'')$	t	$2t-1$	1

All other corresponding variables take identical values in the three solutions. We see that solution A is a convex combination of solutions B and C and thus it cannot be basic for any $t > 1$. It follows that if a basic feasible solution has a record k such that $a_{i''jk} = 1$ and $a_{ijk} = 0 \ \forall i \neq i''$ and $v_{i'jl} = 1$ and $v_{ijl} = 0 \ \forall i \neq i'$ then the corresponding t_{jk} variable takes value 1 and the corresponding r_{ijkl} variables are integer. ■

Theorem 3.3. *For any fixed $y \in Y$, every basic feasible solution of formulation (IPR_y) has integer values for all the variables.*

Proof: From the results of lemma 3.2 we know that every basic feasible solution of each subproblem (IPR_y^{jl}) has integer values for all variables. From the separability of (IPR_y) into nq subproblems (IPR_y^{jl}), that we demonstrated in subsection 3.1.1, it follows that in every basic feasible solution of (IPR_y) the collection of variables associated with each subproblem (IPR_y^{jl}) will form a basic feasible solution for that subproblem. Further in lemma 3.2 we have proved that every basic feasible solution of subproblem (IPR_y^{jl}) has integer values. Thus, we can conclude that all variables in every basic feasible solution of (IPR_y) have integer values. ■

Corollary 3.1. *If formulation (IPR) is solved using a branch and bound algorithm that uses the LP relaxation of (IPR), the solution obtained has integer values for all variables and hence is optimal for (IP).*

Proof: A branch and bound algorithm based on the LP relaxation of (IPR) will find basic feasible solutions to (IPR_y) for some $y \in Y$ and hence from theorem 3.3, the optimal solution found by a branch and bound algorithm will have integer values for all variables and hence would be optimal for (IP). ■

3.2 Benders Decomposition for the q-mode problem.

J. F. Benders developed an algorithmic strategy to solve mixed integer programming problems. In section 3.1 we have described a formulation (IPR) where only the y_{kl} variables are binary and the remaining variables are linear. The algorithmic strategy developed by Benders can be used to iteratively solve this mixed integer programming formulation. The details of our implementation are presented in the subsections that follow. The theory and detailed analysis of Benders decomposition is available in [18].

3.2.1 Benders' Reformulation for the q-mode Problem.

We look at the problem (IPR_y) created by fixing the values of the y_{kl} variables in the formulation (IPR) of the q-mode problem. As discussed in the earlier section, there exists a $y^* \in Y$ that solves (IPR) and for any fixed value of $y \in Y$, the q-mode problem reduces to a linear programming problem in variables t_{jk} and v_{ijl}. In the context of Benders' decomposition we shall refer to this linear programming problem as (BP_y) or the Benders' primal subproblem.

$$(BP_y)\left(\begin{array}{ccc} \min & \sum_j \sum_k t_{jk} & \\ s.t. & t_{jk} + v_{ijl} \geq a_{ijk} y_{kl} & \forall\, i,j,k,l \\ & \sum_i v_{ijl} = 1 & \forall\, j,l \\ & v_{ijl} \geq 0, t_{jk} \geq 0 & \forall\, i,j,k,l \end{array}\right)$$

This linear programming problem has a dual which we refer to as (BD_y) or the Benders' dual subproblem. The formulation of (BD_y), consisting of dual variables α_{ijkl} and β_{jl} $\forall\, i,j,k,l$, is given below.

$$(BD_y)\left(\begin{array}{lcr} max & \sum_i \sum_j \sum_k \sum_l \alpha_{ijkl} a_{ijk} y_{kl} + \sum_j \sum_l \beta_{jl} & \\ s.t. & \sum_i \sum_l \alpha_{ijkl} \leq 1 & \forall\, j,k \\ & \sum_k \alpha_{ijkl} + \beta_{jl} \leq 0 & \forall\, i,j,l \\ & \alpha_{ijkl} \geq 0, \beta_{jl} \in \Re^{nq} & \forall\, i,j,k,l \end{array}\right)$$

Before proceeding further in our discussion of the Benders decomposition of (IPR) we discuss an efficient procedure for solving (BP_y) and (BD_y) for any fixed value of y.

Proposition 3.1. *Given a valid data set (a_{ijk}), i.e., each record k has exactly n positions and each position has one of the m available values, and a valid assignment of records to clusters, i.e., $y \in Y$, the Benders primal problem (BP_y) and its dual (BD_y) are bounded and feasible and the optimal objective value is $np - \sum_j \sum_l \max_i F_l(i,j)$.*

Proof: The Benders' primal subproblem was obtained by fixing the values of the y_{kl} variables and we showed in Section 3.1.2 that the optimal solution of a subproblem (IPR_y^{jl}) is $|\Phi_l| - \max_i F_l(i,j)$, where $|\Phi_l|$ is the total number of records assigned to cluster l and $F_l(i,j)$ is the number of records in cluster l having value i in position j.

The optimal objective value for BP_y is the sum of the optimal values of each of the subproblems that constitute BP_y.

$$\begin{aligned} \text{i.e., } v(BP_y) &= \textstyle\sum_j \sum_l (|\Phi_l| - \max_i F_l(i,j)) \\ &= \textstyle\sum_j \sum_l |\Phi_l| - \sum_j \sum_l \max_i F_l(i,j) \\ &= np - \textstyle\sum_j \sum_l \max_i F_l(i,j) \end{aligned} \tag{3.14}$$

Given a problem instance (a_{ijk}) and a set of values $y \in Y$ we can now construct a feasible solution $(\bar{\alpha}_{ijkl}, \bar{\beta}_{jl})$ for the Benders dual subproblem (BD_y). Let $\bar{\alpha}_{ijkl} = a_{ijk} y_{kl}$ and $\bar{\beta}_{jl} = -\max_i \sum_k \bar{\alpha}_{ijkl}$ $\forall i,j,k,l$. From the definition of the variables y_{kl}, we know that for each record k, $y_{kl} = 1$ for a unique cluster $l = l^*$ and $y_{kl} = 0 \;\forall\; l \neq l^*$. Hence

$\sum_i \sum_l \bar{\alpha}_{ijkl} = \sum_i \sum_l a_{ijk} y_{kl} = \sum_i a_{ijk}$. Moreover, for a given position j of a given record k, $a_{ijk} = 1$ for a unique value i^* and $a_{ijk} = 0 \; \forall \; i \neq i^*$. Hence $\sum_i a_{ijk} = 1$ and it follows that $\sum_i \sum_l \bar{\alpha}_{ijkl} = 1$.

Also from the definition of $\bar{\beta}_{jl}$ all constraints of type $\sum_k \alpha_{ijkl} + \beta_{jl} \leq 0$ are satisfied. Thus, the solution $(\bar{\alpha}_{ijkl}, \bar{\beta}_{jl})$ that we have constructed is feasible for the dual problem (BD_y). Further, for the corresponding value of the objective function we have

$$\begin{aligned}\sum_i \sum_j \sum_k \sum_l \bar{\alpha}_{ijkl} a_{ijk} y_{kl} &= \sum_i \sum_j \sum_k \sum_l a_{ijk} y_{kl} \text{ (from definition of } \alpha_{ijkl}) \\ &= \sum_j \sum_k \left(\sum_i \sum_l a_{ijk} y_{kl} \right) = \sum_j \sum_k (1) = np\end{aligned} \tag{3.15}$$

$$\text{and,} \sum_k \bar{\alpha}_{ijkl} = \sum_k a_{ijk} y_{kl} = \sum_{k \in \Phi_l} a_{ijk} = F_l(i,j) \quad \therefore -\max_i \sum_k \alpha_{ijkl} = -\max_i F_l(i,j)$$

$$\therefore \sum_j \sum_l \beta_{jl} = -\sum_j \sum_l \max_i F_l(i,j) \tag{3.16}$$

From equations (3.14), (3.15) and (3.16) it is clear that the objective value of the Benders' dual subproblem corresponding to the solution $(\bar{\alpha}_{ijkl}, \bar{\beta}_{jl})$ equals the optimal objective value of the Benders' primal. Hence the feasible solution for the dual problem that we have constructed is in fact optimal for (BD_y). ■

In the next section we resume the discussion of Benders decomposition for (IPR) by explaining the formulation of the master problem.

3.2.2 The Benders Master Problem.

From the definition of the primal problem (BP_y) and its dual (BD_y) it is clear that $v(IP) = \min_{y \in Y} v(BP_y) = \min_{y \in Y} v(BD_y)$.

Let $\mathcal{A}$ be the set of extreme points of the region defined by the constraints of the Benders' dual (BD_y) and let $(\tilde{\alpha}^r_{ijkl}, \tilde{\beta}^r_{jl}) \in \mathcal{A}$ for $r = 1$ to $|\mathcal{A}|$. From linear programming theory we know that the optimal solution of (BD_y) lies at one of the extreme points of its feasible region hence we can rewrite $\min_{y \in Y} v(BD_y)$ as

$$\min_{y\in Y}\left\{\max_{r=1 \text{ to } |\mathcal{A}|}\left\{\sum_i\sum_j\sum_k\sum_l \tilde{\alpha}^r_{ijkl}a_{ijk}y_{kl}+\sum_j\sum_l\tilde{\beta}^r_{jl}\right\}\right\}$$

or,

$$(BM)\left(\begin{array}{ll}\min & \Omega \\ \text{s.t.} & \sum_i\sum_j\sum_k\sum_l \tilde{\alpha}^r_{ijkl}a_{ijk}y_{kl}+\sum_j\sum_l\tilde{\beta}^r_{jl} \le \Omega \quad \forall r=1 \text{ to } |\mathcal{A}| \\ & y\in Y\end{array}\right)$$

The above formulation (BM) is called the master problem of the Benders decomposition and from the above discussion it is clear that the optimal value of the master problem (BM) is the same as the optimal value for (IPR). Next we present an iterative algorithm for solving the Benders' master problem.

3.2.3 Details of the Algorithm based on Benders Decomposition.

The master problem formulation (BM) involves a large number of constraints as it has one constraint for every extreme point in the set $\mathcal{A}$. Solving such a large problem may be computationally very expensive. Moreover, the master problem solution involves the issue of generating all the extreme points in set the $\mathcal{A}$ which by itself is a daunting task. This is resolved by using an iterative framework developed by Benders in which constraints of the master problem are generated one at a time (one constraint at each iteration). At each iteration in this iterative algorithm we have a relaxed version of the master problem (BM_r) containing only the constraints generated in the past r iterations. Thus (BM_r) has only a subset of the total constraints present in (BM). We solve the relaxed problem (BM_r) to obtain a solution y^{r+1} and then solve the

```
0  $\Omega_0 \leftarrow -\infty$, $r \leftarrow 1$, $A^0 \leftarrow \emptyset$, and $y^1 \in Y$
1  Solve $(BD_{y^r})$, obtain $(\tilde{\alpha}^r_{ijkl}, \tilde{\beta}^r_{jl})$
2  **If** $\Omega_{r-1} \geq v(BD_{y^r})$ **then**
3      STOP. $y^r$ is an optimal solution.
4  **else** $A^r \leftarrow A^{r-1} \cup (\tilde{\alpha}^r_{ijkl}, \tilde{\beta}^r_{jl})$
5  Create and add constraint corresponding to $(\tilde{\alpha}^r_{ijkl}, \tilde{\beta}^r_{jl})$ to $BM_{r-1}$ to get $(BM_r)$
6  Solve $(BM_r)$ to get solution $y^{r+1}$ and $\Omega_r$, let $r \leftarrow r+1$. **goto** 1.
```

Figure 3.1: Pseudo Code for Algorithm B_{opt}

dual problem $(BD_{y^{r+1}})$. The solution to the dual problem will give rise to a new constraint which is then added to (BM_r) to obtain (BM_{r+1}) and so on. Below, we describe in detail this iterative algorithm, which we shall refer to as B_{opt}. The major steps of this algorithm are explained in figure 3.1.

Line 0 of the algorithm B_{opt} is an initialization step. The iteration counter r is initialized to 1 and we let Ω_0 be a negative number with a large absolute value. The initial solution y^1 can be chosen by randomly allotting the records to clusters or we can use the solution of a constructive heuristic algorithm. In line 1, we use the method shown in proposition 3.1 to find the optimal solution $(\tilde{\alpha}^r_{ijkl}, \tilde{\beta}^r_{jl})$ of problem (BD_{y^r}). In line 2 we evaluate the optimality criteria. If it is satisfied then we go to line 3 where the iterative procedure terminates with an optimal vector y^r. We can now find the corresponding mode of each of the q collections of records and also obtain the corresponding optimal solution to the problem instance. If the optimality criteria is not satisfied, i.e., $\Omega_{r-1} < v(BD_{y^r})$, then we go to line 4 where the optimal solution $(\tilde{\alpha}^r_{ijkl}, \tilde{\beta}^r_{jl})$ is added to the set of extreme points $\mathcal{A}^{r-1}$ to obtain $\mathcal{A}^r$. Further, in Line 5 we use the optimal solution $(\tilde{\alpha}^r_{ijkl}, \tilde{\beta}^r_{jl})$ to create the r^{th} constraint and add it to the relaxed master problem (BM_{r-1}) to obtain (BM_r). In line 6 we solve the relaxed master problem (BM_r) to get new solution y^{r+1} and its corresponding lower bound, Ω_r, for the optimal value, $v(IP)$. We also increase the value of the iteration counter by 1 and then go to line 1. An important point to notice here is that the optimal solution of the earlier iteration, i.e., y^r, can be used as a starting solution for problem BM_r. This gives the branch and bound algorithm that is used to solve BM_r a good quality integer feasible starting solution.

This algorithm can be applied for two purposes. The first one is to obtain an integer optimal solution to the q-mode problem. The second purpose is to obtain a high quality lower bound in reasonable time for large problem instances which cannot be optimally solved. The lower bound obtained from B_{opt} can be used in the context of a branch and bound algorithm or to serve as a benchmark to evaluate the quality of solutions obtained from a heuristic algorithm.

Each iteration of the B_{opt} algorithm involves solving the relaxed master problem, (BM_r), which by itself is a combinatorial optimization problem. Further (BM_r) contains r constraints at the r^{th} iteration and hence it grows in size with each iteration. Preliminary computational experiments reveal that the value of r does become quite large before the algorithm terminates. We modify algorithm B_{opt} so as to reduce its computational requirements. This modification involves solving of the relaxed master problem sub-optimally so that the corresponding computational requirements at every iteration are considerably reduced.

From the theory of Benders' decomposition we know that at every iteration of B_{opt} where $\Omega_{r-1} = v(BM_{r-1}) < v(BD_{y^r})$, a distinct extreme point of the feasible region of the dual subproblem is identified. This in turn ensures that the algorithm B_{opt} does not cycle and terminates in a finite number of iterations. Thus, if we solve the relaxed master problem sub-optimally, but obtain a solution, $\bar{y}^r$ such that $\bar{v}(BM_{r-1}) < v(BD_{\bar{y}^r})$ then we have essentially identified a distinct extreme point and the algorithm can proceed without any theoretical difficulty. Here $\bar{v}(BM_{r-1})$ refers to the objective value of BM_{r-1} corresponding to solution $\bar{y}^r$. This idea is used to modify B_{opt} and we refer to the modified algorithm as B_{Nopt}. At the r^{th} iteration of B_{Nopt} we use the CPLEX MIP solver to find an integer solution, $\bar{y}^r$ (not necessarily optimal), to BM_{r-1}. We use the solution, $\bar{y}^r$, to obtain $v(BD_{\bar{y}^r})$ and evaluate the stopping criterion. If the stopping criterion is not satisfied, i.e., $\bar{v}(BM_{r-1}) < v(BD_{\bar{y}^r})$, then we calculate the corresponding optimal solution to the dual subproblem $(\tilde{\alpha}^r_{ijkl}, \tilde{\beta}^r_{jl})$ using $\bar{y}^r$. The solution $(\tilde{\alpha}^r_{ijkl}, \tilde{\beta}^r_{jl})$ identifies a distinct extreme point of the dual feasible region and we can use it to create the next constraint for the Benders master problem. This constraint is added to the relaxed master problem to obtain BM_r and then we proceed to the next iteration.

Note that when BM_{r-1} is solved sub-optimally the objective value $\bar{\Omega}_{r-1} = \bar{v}(BM_{r-1})$ is no longer a valid lower bound for the q-mode problem. If the stopping criterion is satisfied i.e., $\bar{v}(BM_{r-1}) \geq v(BD_{\bar{y}^r})$, then we go back to the relaxed master problem BM_{r-1} and solve it to obtain a better integer solution $\hat{y}^r$. This process of finding better solutions to BM_{r-1} goes on until we find a solution that violates the stopping criterion or we find an optimal solution to BM_{r-1}. If the stopping criterion is not satisfied then we can obtain a distinct extreme point and if it is satisfied with an optimal solution to BM_{r-1} then we have an optimal integer solution to the q-mode problem. Further, if algorithm B_{Nopt} is terminated using a different stopping criterion before it finds the optimal solution (e.g., it is forced to terminate after a fixed time period) then we have to re-solve the relaxed master problem in the last iteration optimally so that we obtain a valid lower bound for the q-mode problem. We shall refer to this modified algorithm as B_{Nopt}. Flowcharts of both algorithms, B_{opt} and B_{Nopt}, are displayed in figure 3.2.

Both algorithms B_{opt} and B_{Nopt} are such that we obtain a solution y^r for the relaxed Benders' master problem BM_{r-1} at each iteration. Then the optimal value of (BD_{y^r}) is obtained using the procedure discussed in proposition 3.1 and an efficient procedure to calculate the mode of a collection of records. Next we calculate the optimal solution $(\tilde{\alpha}^r_{ijkl}, \tilde{\beta}^r_{jl})$ to the Benders dual subproblem BD_{y^r}. This is required so that we can create a new constraint to be added to BM_{r-1} to obtain BM_r. The computation of the optimal solution of BD_r can be avoided by re-writing the constraints of the relaxed Benders' master problem BM_r in terms of the solutions of the relaxed

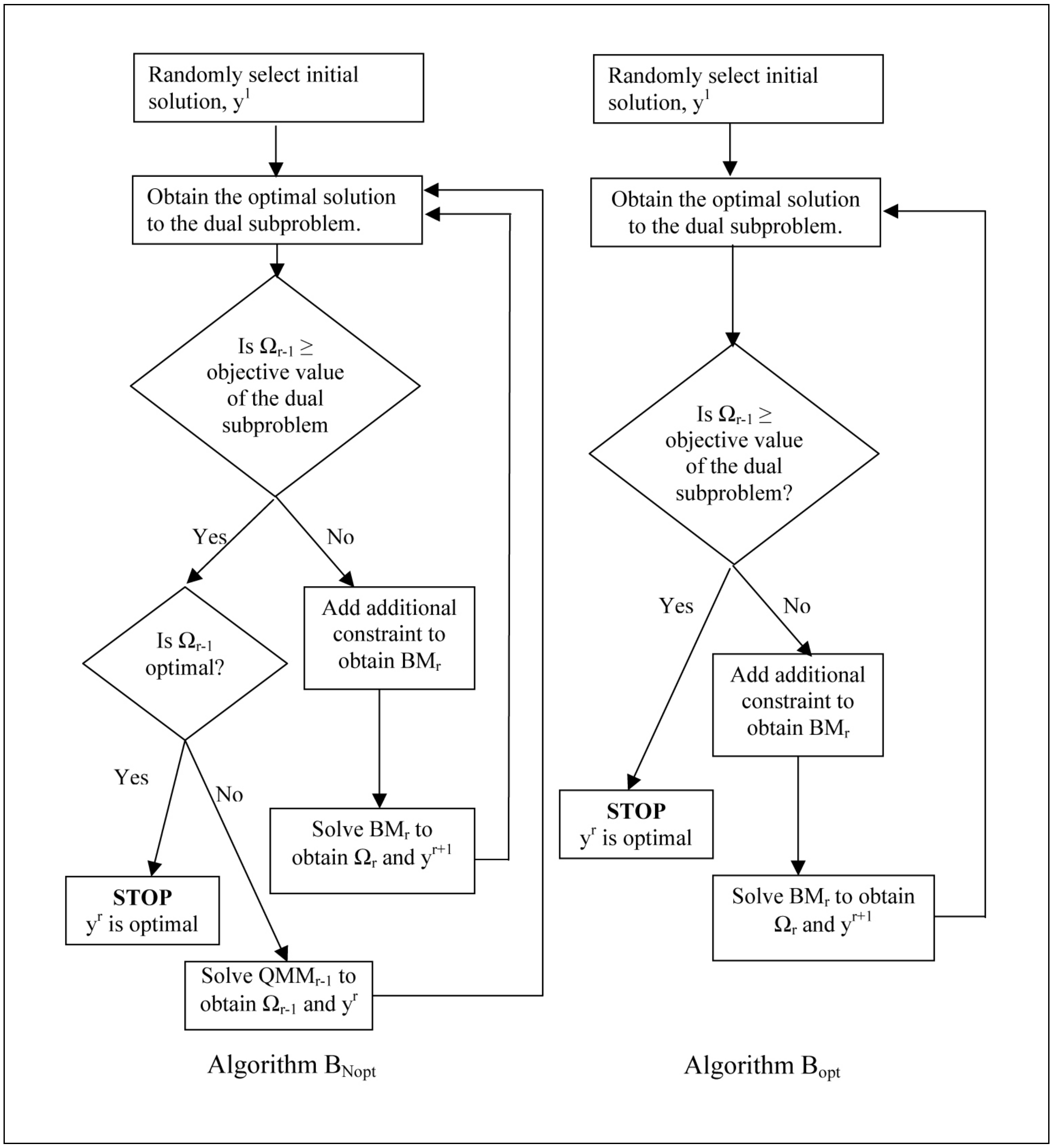
Randomly select initial solution, y^1
Obtain the optimal solution to the dual subproblem.
Is $\Omega_{r-1} \geq$ objective value of the dual subproblem
Yes
No
Is Ω_{r-1} optimal?
Add additional constraint to obtain BM_r
Yes
No
Solve BM_r to obtain Ω_r and y^{r+1}
STOP y^r is optimal
Solve QMM_{r-1} to obtain Ω_{r-1} and y^r
Algorithm B_{Nopt}
Randomly select initial solution, y^1
Obtain the optimal solution to the dual subproblem.
Is $\Omega_{r-1} \geq$ objective value of the dual subproblem?
Yes
No
Add additional constraint to obtain BM_r
STOP y^r is optimal
Solve BM_r to obtain Ω_r and y^{r+1}
Algorithm B_{opt}

Figure 3.2: Flowcharts for algorithms B_{Nopt} and B_{opt}.

Benders' master problem from the earlier iterations, $y^1, \ldots, y^{r-1}$, instead of writing in terms of the optimal solutions of the Benders' dual subproblem. We demonstrate this in the proposition below.

Proposition 3.2. *The relaxed master problem (BM_r) can be written as*

$$(BM'_r)\left(\begin{array}{lll} min & \Omega & \\ s.t. & n\sum_k \sum_l y^e_{kl} y_{kl} - \sum_j \sum_l \max_i \sum_k a_{ijk} y^e_{kl} \leq \Omega & \forall\ e = 1\ to\ r \\ & \sum_l y_{kl} = 1 & \forall\ k \\ & y_{kl} \in \{0,1\} & \forall\ k,l \end{array}\right)$$

where y^e_{kl} represents the assignment of records to clusters that correspond to the extreme point point $(\tilde{\alpha}^e_{ijkl}, \tilde{\beta}^e_{kl})$ of the feasible region of the dual subproblem $\forall\ e = 1$ to r.

Proof: The relaxed master problem is

$$(BM_r)\left(\begin{array}{lll} min & \Omega & \\ \text{s.t.} & \sum_i \sum_j \sum_k \sum_l \tilde{\alpha}^e_{ijkl} a_{ijk} y_{kl} + \sum_j \sum_l \tilde{\beta}^e_{jl} \leq \Omega & \forall\ e = 1 \text{ to } r \\ & \sum_l y_{kl} = 1 & \forall\ k \\ & y_{kl} \in \{0,1\} & \forall\ k,l \end{array}\right)$$

Consider the left hand side of the e^{th} constraint of problem BM_r.

$\sum_i \sum_j \sum_k \sum_l (\tilde{\alpha}^e_{ijkl}) a_{ijk} y_{kl} + \sum_j \sum_l \tilde{\beta}^e_{jl}$

$= \sum_i \sum_j \sum_k \sum_l (a_{ijk} y^e_{kl}) a_{ijk} y_{kl} + \sum_j \sum_l \tilde{\beta}^e_{jl}$ $\quad (\because$ by def. $\tilde{\alpha}^e_{ijkl} = a_{ijk} y^e_{kl})$

$$= \sum_i \sum_j \sum_k \sum_l (a_{ijk} y^e_{kl}) a_{ijk} y_{kl} - \sum_j \sum_l \max_i \sum_k a_{ijk} y^e_{kl} \quad \left(\begin{aligned} \because \tilde{\beta}^e_{jl} &= -\max_i \tilde{\alpha}^e_{ijkl} \\ &= -\max_i \sum_k a_{ijk} y^e_{kl} \end{aligned} \right)$$

$= \sum_k \sum_l \sum_i \sum_j a_{ijk} y^r_{kl} y_{kl} - \sum_j \sum_l \max_i \sum_k a_{ijk} y^r_{kl}$ $\quad (\because a^2_{ijk} = a_{ijk})$

$= \sum_k \sum_l y^r_{kl} y_{kl} \left(\sum_j \sum_i (a_{ijk}) \right) - \sum_j \sum_l \max_i \sum_k a_{ijk} y^r_{kl}$

$= \sum_k \sum_l y^r_{kl} y_{kl} (n) - \sum_j \sum_l \max_i \sum_k a_{ijk} y^r_{kl}$

$= n \sum_k \sum_l y^r_{kl} y_{kl} - \sum_j \sum_l \max_i \sum_k a_{ijk} y^r_{kl}$

Hence we can rewrite the relaxed master problem (BM_r) as (BM'_r). ■

3.3 Additional MIP formulations for the q-mode Problem.

3.3.1 An Alternate MIP Formulation.

Here we present an alternate MIP formulation for the q-mode problem. This formulation is based on the definition of the mode of a collection of records. Given a collection of records and an assignment of every record to one of q clusters then the q-mode problem is transformed into "q" 1-mode problems. In other words, the assignment of every record to a unique cluster creates q different sub-collections of records from the original collection, and solution to the q-mode problem involves calculating a mode for each of the q sub-collections. We use this concept to propose an alternative integer programming model for the q-mode problem.

We keep the same notation that we used in the previous section and repeat some of the notation below for easy reference. An instance of the q-mode problem is given in the form of a three dimensional array a_{ijk}.

$$a_{ijk} = \begin{cases} 1 & \text{if value } i \text{ is present at position } j \text{ in record } k. \\ 0 & \text{otherwise.} \end{cases}$$

We also need the decision variable y_{kl}.

$$y_{kl} = \begin{cases} 1 & \text{if record } k \text{ is assigned to cluster } l. \\ 0 & \text{otherwise.} \end{cases}$$

Since every record is to be assigned to a unique cluster, we have $\sum_l y_{kl} = 1 \ \forall \ k$. Further, the total number of records assigned to cluster l that have value i in position j is given by $\sum\limits_k a_{ijk}y_{kl}$, i.e., $F_l(i,j) = \sum\limits_k a_{ijk}y_{kl}$. By definition, the value at position j in the mode corresponding to cluster l is given by $\arg\max_i F_l(i,j) = \arg\max_i \sum_k a_{ijk}y_{kl}$. All the records that do not have value i in position j will require a replacement for that position. Hence the total number of replacements required for position j in cluster l is given by number of records not having value i at position j, i.e., $|\Phi_l| - F_l(i,j) = \sum_k y_{kl} - \max_i \sum_k a_{ijk}y_{kl}$.

Proceeding in similar fashion for each slot j and each cluster l we can say that the total number of replacements required for the complete set of given records is

$\sum_l \sum_j (\sum_k y_{kl} - \max_i \sum_k a_{ijk}y_{kl})$.

Further, the q-mode problem requires the assignment of each record to a unique cluster so as to minimize the sum of the replacements corresponding to all the clusters i.e.,

$$\min \left\{ \sum_l \sum_j (\sum_k y_{kl} - \max_i \sum_k a_{ijk} y_{kl}) \middle| \sum_l y_{kl} = 1 \ \forall \ k \right\} \qquad (\star)$$

We know that at every feasible solution $\sum_l y_{kl} = 1$, hence $\sum_l \sum_j \sum_k y_{kl} = \sum_j \sum_k (1) = np$. Thus we can rewrite the objective function as

$$np + \min(-\sum_l \sum_j (\max_i \sum_k a_{ijk} y_{kl})) = np - \max(\sum_l \sum_j (\max_i \sum_k a_{ijk} y_{kl}))$$

Let us now refer to $F_l^{max}(i,j) = \max_i \sum_k a_{ijk} y_{kl}$ as w_{jl}. Thus w_{jl} represents the frequency count of the most frequently occurring value at position j in all the records assigned to cluster l. It follows that, for each j and l, the corresponding w_{jl} can be obtained by enforcing the following constraints

$$\begin{array}{lll}
\sum\limits_k a_{ijk} y_{kl} \le w_{jl} & \forall i & \\
\sum\limits_k a_{ijk} y_{kl} \ge w_{jl} - A_j(1 - v_{ijl}) & \forall i & \text{where } A_j > \sum\limits_k a_{ijk} y_{kl} \ \forall i \text{ e.g. } A_j = p \\
\sum\limits_i v_{ijl} = 1 & & \\
v_{ijl} \in \{0,1\} & \forall i &
\end{array}$$

Using this constraint structure, the formulation marked $(\star)$ reduces to the following integer programming model.

$$np - \max \sum_l \sum_j w_{jl}$$

$$\text{s.t.} \sum_k a_{ijk} y_{kl} \leq w_{jl} \quad \forall i, j, l \tag{3.17}$$

$$\sum_k a_{ijk} y_{kl} \geq w_{jl} - A_j(1 - v_{ijl}) \quad \forall i, j, l \tag{3.18}$$

$$\sum_i v_{ijl} = 1 \quad \forall j, l \tag{3.19}$$

$$\sum_l y_{kl} = 1 \quad \forall k \tag{3.20}$$

$$w_{jl} \geq 0, v_{ijl} \in \{0,1\}, y_{kl} \in \{0,1\} \quad \forall i, j, k, l \tag{3.21}$$

In the above model, constraints (3.17), (3.18) and (3.19) collectively ensure that w_{jl} is equal to the frequency of the most frequently occurring value i in position j of all records assigned to cluster l, i.e, $w_{jl} = F_l^{max}(i,j) = \sum\limits_k a_{ijk}y_{kl} \ \forall\ j, l$. Since the objective function in our model involves maximizing $(\sum_l \sum_j w_{jl})$, the constraints (3.18) and (3.19) would select w_{jl} such that $w_{jl} = \max\limits_i \sum_k a_{ijk}y_{kl}$ and constraints (3.17) become redundant. It is interesting to note here that when $w_{jl} = \max_i \sum_k a_{ijk}y_{kl}$ the corresponding $v_{ijl} = 1$. Thus every v_{ijl} variable that is equal to 1 identifies the value i that occurs most frequently in position j for all records in cluster l. Thus the v_{ijl} variables have the same significance here as they had in the formulation (IPR), i.e., they determine the configuration of the modes. The integer programming model, which we refer to as (IPA) henceforth, is given below.

$$(IPA)\quad \begin{array}{lr} np - \max \sum_l \sum_j w_{jl} & \\ \text{s.t.} \sum_k a_{ijk} y_{kl} \geq w_{jl} - A_j(1 - v_{ijl}) & \forall i, j, l \\ \sum_i v_{ijl} = 1 & \forall j, l \\ \sum_l y_{kl} = 1 & \forall k \\ w_{jl} \geq 0, v_{ijl} \in \{0, 1\}, y_{kl} \in \{0, 1\} & \forall i, j, k, l \end{array}$$

The value assigned to the set of numbers A_j determines how tight the formulation is. We select the value of A_j to be $\max_i \sum_{k=1}^{p} a_{ijk} \quad \forall j = 1, \ldots, n$.

The formulation (IPA) has $mnq+nq+p$ constraints and $nq+mnq+np$ variables as compared to $mnpq+nq+p$ constraints and $np+mnq+pq$ variables in formulation (IPR). Thus the LP relaxation of (IPA) will have a far smaller size of the basis as compared to the LP relaxation of (IPR). On the other hand (IPR) has only np binary variables while (IPA) has $np + mnq$ binary variables [1]. We conduct a computational experiment to compare the LP relaxations of IPA and IPR. We present the details of this experiment and the related discussion later in section 5.2.

3.3.2 A Modification of the Formulation (IPR).

In this subsection, we present a modification to the formulation (IPR) such that the linear programming relaxation of the resulting formulation can potentially have a higher objective value than the linear programming relaxation of (IPR). This modification involves replacing the decision variable t_{jk} with the variables t_{jkl} which are defined as

$$t_{jkl} = \begin{cases} 1 & \text{if a replacement is required at position } j \text{ of record } k \text{ at cluster } l \\ 0 & \text{otherwise.} \end{cases}$$

[1]Through a numerical example we can show that, unlike (IP), in this model if the binary constraints on the v_{ijl} variables are relaxed then the resulting formulation might have fractional v_{ijl} values in the optimal solution.

The objective value is also changed to $\sum_j \sum_k \sum_l t_{jkl}$. We shall refer to this modified formulation as (IP'). It is trivial to show that the formulation (IP') is a valid formulation for the q-mode problem. Further, we can relax the integer requirements on the variables v_{ijl} to obtain formulation (IPT) which is given below. Next we show that the formulation (IPT) can be used to obtain an optimal solution to the q-mode problem and also prove that the LP relaxation of (IPT) is greater than or equal to the LP relaxation of (IPR).

$$
\text{(IPT)} \quad
\begin{aligned}
&\min \sum_j \sum_k \sum_l t_{jkl} && && (3.22)\\
&\text{s. t. } a_{ijk} y_{kl} - v_{ijl} \le t_{jkl} && \forall i,j,k,l && (3.23)\\
&\sum_i v_{ijl} = 1 && \forall j,l && (3.24)\\
&\sum_l y_{kl} = 1 && \forall k && (3.25)\\
&y_{kl} \in \{0,1\}, v_{ijl} \ge 0, t_{jkl} \ge 0 && \forall\, i,j,k,l && (3.26)
\end{aligned}
$$

Using arguments similar to those presented in section 3.1.1 for (IPR), we can show that for any fixed $y \in Y$ in formulation (IPT), the corresponding model (IPT_y) is fully separable in index variables j and l. The fully separated form of (IPT_y) is given below.

$$
(IPT_y) \qquad \sum_j \sum_l \left(\begin{array}{cc}
\min \sum_k t_{jkl} & \\
\text{s.t. } t_{jkl} + v_{ijl} \ge a_{ijk} y_{kl} & \forall\, i,k, \\
\sum_i v_{ijl} = 1 & \\
v_{ijl} \ge 0, t_{jkl} \ge 0 & \forall\, i,k
\end{array} \right)
$$

If record k does not belong to cluster l, i.e., $k \notin \Phi_l$, then $y_{kl} = 0$. Thus the constraint $t_{jk} + v_{ijl} \geq a_{ijk}y_{kl}$ becomes $t_{jk} + v_{ijl} \geq 0$ and since $v_{ijl} \geq 0$, $t_{jkl} \geq 0$, $\forall$ i, j, k, l, this constraint is satisfied by any feasible values of v_{ijl} and t_{jkl} and hence it is redundant. This implies that the constraints $t_{jk} + v_{ijl} \geq a_{ijk}y_{kl}$ can be dropped for all $k \notin \Phi_l$ and the formulation (IPT_y) can be written as

$$(IPT_y) \qquad \sum_j \sum_l \left(\begin{array}{cc} \min \sum_{k\in\Phi_l} t_{jkl} & \\ \text{s.t. } t_{jk} + v_{ijl} \geq a_{ijk} & \forall\, i, k \in \Phi_l, \\ \sum_i v_{ijl} = 1 & \\ v_{ijl} \geq 0, t_{jk} \geq 0 & \forall\, i, k \end{array} \right)$$

We can also use arguments similar to those presented in section 3.1.3 to demonstrate that every basic feasible solution of (IPT_y) has integer values for all variables. Similar to Theorem 3.3 and Corollary 3.1 corresponding to (IPR), we have Theorem 3.4 and corollary 3.2 corresponding to (IPT). We state these below without a proof.

Theorem 3.4. *For any fixed $y \in Y$, every basic feasible solution of formulation (IPT_y) has integer values for all the variables.*

Corollary 3.2. *If formulation (IPT) is solved using a branch and bound algorithm that uses the LP relaxation of (IPT), the solution obtained has integer values for all variables and hence is optimal for (IP).*

Next we prove that the optimal objective value of the LP relaxation of the formulation (IPT) is greater than or equal to the optimal objective value of the LP relaxation of (IPR). Let LPT and LPR be the linear programming models obtained from IPR and IPT respectively, by replacing the binary restrictions, i.e., $y_{kl} \in \{0, 1\}$, with simple non-negativity restrictions, i.e., $y_{kl} \geq 0$ for all k, l. Further, let $v(LPT)$ and $v(LPR)$ be the optimal objective values of LPT and LPR respectively.

Theorem 3.5. *The optimal objective value of LPT is no worse than that of LPR, i.e., $v(LPR) \leq v(LPT)$.*

Proof : Let $S_T^* = \{y_{kl}^*, t_{jkl}^*, v_{ijl}^*\}$ be an optimal solution for (LPT) and let its corresponding objective value be given by $v(S_T^*)$, i.e.,

$$v(LPT) = v(S_T^*) = \sum_j \sum_k \sum_l t_{jkl}^* \tag{3.27}$$

Since S_T^* is a feasible solution for (LPT), it follows that

$$a_{ijk} y_{kl}^* - v_{ijl}^* \leq t_{jkl}^* \; \forall \; i, j, k, l, \quad \sum_l y_{kl}^* = 1 \; \forall \; k, \quad \sum_i v_{ijl}^* = 1 \; \forall \; j, l$$

Now let $t'_{jk} = \max_l t_{jkl}^* \; \forall \; j, k$. It follows that, $a_{ijk} y_{kl}^* - v_{ijl}^* \leq t'_{jk} \; \forall \; i, j, k, l$ and $S_R = \{y_{kl}^*, t'_{jk}, v_{ijl}^*\}$ is a feasible solution for (LPR). The objective value of (LPR) corresponding to solution S_R is given by

$$v(S_R) = \sum_j \sum_k t'_{jk} \tag{3.28}$$

Let S_R^* be an optimal solution to (LPR). Since S_R is merely a feasible solution for LPR, it follows that

$$v(LPR) = v(S_R^*) \leq v(S_R) \tag{3.29}$$

From equations (3.28) and (3.29), and from the definition of t'_{jk}, it follows that

$$v(S_R^*) \leq v(S_R) = \sum_j \sum_k t'_{jk} = \sum_j \sum_k \max_l t_{jkl}^* \tag{3.30}$$

Since it is clearly true that $\max_l t_{jkl}^* \leq \sum_l t_{jkl}^* \; \forall \; j, k$, it follows that

$$\sum_j \sum_k \max_l t_{jkl}^* \leq \sum_j \sum_k \sum_l t_{jkl}^* = v(S_T^*) \tag{3.31}$$

Therefore equations (3.27), (3.29), (3.30) and (3.31) imply that $v(LPR) \leq v(LPT)$.
■

The formulation (IPT) has the same number of constraints as (IPR) but the np variables of type t_{jk} are replaced by npq variables of type t_{jkl}. Hence (IPT) is a larger formulation as compared to (IPR). But preliminary computational experiments reveal that $v(LPT)$ is indeed much larger than $v(LPR)$ for many instances of the q-mode problem. Later, in section 5.2, we present a computational experiment in which we empirically compare the LP relaxations of the (IPT), (IPR) and (IPA).

Chapter 4

Random generation of problem instances for the q-mode problem.

4.1 Introduction.

An instance of the q-mode problem is characterized by the parameters m, n, p and q. Given these parameters, an instance of the q-mode problem consists of p records, each record is described by n attributes. Further, we need to partition these p records into q distinct clusters. We number the values that a specific attribute can take using integers and hence each attribute in every record contains a value i lying between 1 and m. Here we are assuming that each attribute has m possible values but this assumption can be removed with minor changes. For clarity of presentation, we shall refer to the attribute value as "a-value" throughout this chapter. Effectively, a record in an instance of the q-mode problem is a vector with n positions, and each position contains an a-value lying between 1 and m.

The instances that we create have subsets of vectors such that vectors belonging to the same subset have a lower degree of dissimilarity, i.e., they tend to be *closer* to each other, as compared to the vectors from other subsets. In other words, we create artificial instances that have natural clusters. We then use different algorithms to "discover" these natural clusters in the instances and report the findings. To obtain a problem instance that has natural clusters we use the concept of *profiles* with non-uniform probability mass functions. This idea has been used by Morgan [16] to create problem instances for the 2-model problem.

4.2 Notation.

Let an instance of the q-mode problem be represented as an $n \times p$ matrix V, where V_{jk} represents the a-value present at position j in vector k. Each column of this matrix, i.e., vector k, represents a separate record from the given collection. A solution to the q-mode problem is an assignment of every record k ($k = 1$ to p) to a distinct cluster l ($l = 1$ to q). In this chapter we shall use the terms *vector* and *record* interchangeably.

We represent a probability mass function associated with the position j in the l^{th} subset as $f_{jl} = (\pi_{1jl}, \ldots, \pi_{mjl})$, where π_{ijl} is the probability that a-value i is assigned to position j of vector k that belongs to subset l, i.e., $Pr\{V_{jk} = i\} = \pi_{ijl}$, for all $i \in (1, \ldots, m)$ and $\sum_i \pi_{ijl} = 1$. A profile Π_l is defined as a vector consisting of n probability mass functions, i.e., $\Pi_l = (f_{1l}, \ldots, f_{nl})$. To create an instance with q subsets we first identify q different profiles. A vector k corresponding to profile Π_l can be created by using the j^{th} pmf, f_{jl}, to identify an a-value $i \in (1, \ldots, m)$ that will be assigned to the j^{th} position of the vector for every value of $j = 1$ to n. Thus each profile is used to create a subset of vectors and these q subsets together constitute a problem instance.

The degree of dissimilarity of the q profiles and hence the degree of dissimilarity of the vectors created using the different profiles is influenced by the type of pmfs used in the profiles. A uniform pmf will give no preference to any a-value i and the dissimilarity between records created using different profiles will be low giving rise to *weak* clusters in the instance created. On the other hand, a pmf such as (1,0,0,0,0) will cause the a-values assigned in a particular position in all records corresponding to that profile to be identical. Thus we will have strong dissimilarity between records of different subsets leading to *strong* clusters in the instance.

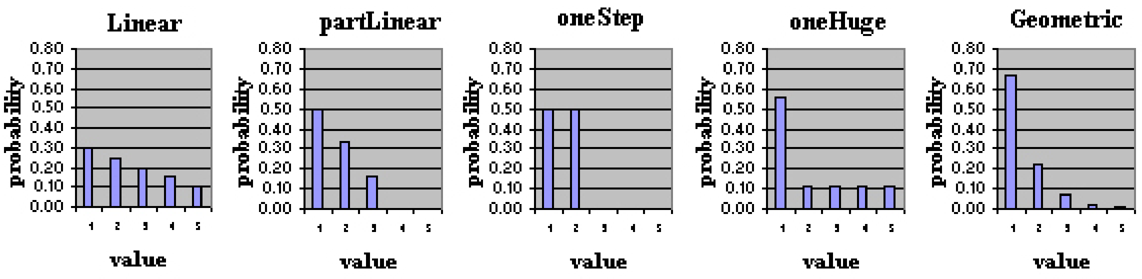

Figure 4.1: The type of pmfs used in the selection of profiles.

The probability mass functions that we use in creating instances can be classified into 6 pmf *types*. We refer to these six pmf types as Uniform, Linear, partLinear, oneStep, oneHuge and Geometric. These pmf types (apart from the Uniform pmf type) are illustrated in figure 4.1. In the first pmf type, Uniform, all a-values that can be assigned to a position of a vector have the same positive probability. To evaluate the probabilities of all the a-values in an Uniform pmf type we need just one parameter, m, which is the total number of a-values. The Linear pmf type also has all a-values having positive probabilities. We refer to the largest probability as π_a and the smallest one as π_b. Furthermore, when the a-values are arranged in the decreasing order of the probabilities the difference between any two adjacent probabilities is constant. A pmf of type Linear is defined using two parameters m and r. Here, m is the total number of a-values that a position in a vector can have and r is the ratio of the largest probability to the smallest probability, i.e., $r = \frac{\pi_a}{\pi_b}$. The third pmf type is referred to as partLinear. This pmf type is similar to the Linear pmf type, the difference is that some values in the partLinear type can have probabilities equal to zero, while in the case of pmf type Linear the probabilities for all values are positive. The partLinear pmf type also requires two parameters $k(< m)$ and r. The parameter k represents the number of a-values that have positive probabilities and r is the ratio of the largest probability to the smallest probability. It is to be noted that when $k = m$, all probabilities in type partLinear will be identical to probabilities in pmf type Linear.

The fourth pmf type is referred to as oneStep. This pmf is similar to the Uniform pmf type discussed earlier. The difference between the two is that some a-values in the oneStep pmf type have probabilities equal to zero while the uniform pmf type has all the a-values with positive probabilities. This pmf consists of some values having probability 0 and the other values having the same positive probability. The oneStep pmf type requires only one parameter $k(< m)$ which is equal to the number of values that have positive probabilities. Obviously, when $k = m$, the oneStep pmf type is the same as the Uniform pmf type. The next pmf type is referred to as oneHuge. This pmf type consists of exactly one a-value that has a large probability while the other a-values have the same positive but relatively smaller probability. This pmf type requires two parameters m, the total number of possible a-values and r, the ratio of the larger probability to the smaller probability.

The last pmf type in our classification is referred to as Geometric. This pmf type consists of all a-values having a positive probability. Further, when the a-values are arranged in the decreasing order of the probability then the ratio of adjacent probabilities (i.e., the larger probability divided by the smaller probability) is constant. The Geometric pmf type requires two parameters m, the total number of possible a-values and r, the ratio between two adjacent probabilities when the values are arranged in the decreasing order of the probabilities.

A summary of the pmf types is given in table 4.1. The table consists of 6 rows each row corresponding to a pmf type. The first column identifies the pmf type. The second column displays the parameters that are required and the third column gives the formula which can be used to calculate the probabilities of the corresponding pmf type once the values of the parameters are given. The fourth and the fifth column are used to display an example of each of the pmf types; the fourth column contains

Table 4.1: The 6 pmf types.

pmf type	param	formula	e.g.	pmf
Uniform	m	$a = b = \frac{1}{m}$	5	(0.2,0.2,0.2,0.2,0.2)
Linear	(m, r)	$\frac{2}{m(r+1)}\left(1 + \frac{\mu(r+1)}{m-1}\right)$ $0 \le \mu \le m-1.$	$(5, 3)$	(0.30,0.25,0.20,0.15,0.10)
partLinear	(k, r)	$\frac{2}{k(r+1)}\left(1 + \frac{\mu(r+1)}{k-1}\right)$ $0 \le \mu \le k-1.$	$(3, 3)$	(0.50,0.33,0.17,0,0)
oneStep	k	$a = \frac{1}{k}.$	2	(0.5,0.5,0,0,0)
oneHuge	(m, r)	$a = \frac{r}{r+m-1}$, $b = \frac{1}{r+m-1}.$	(5,5)	(0.55,0.11,0.11,0.11,0.11)
Geometric	(m, r)	$r^{\mu}\left(\frac{r-1}{r^m-1}\right)$ $0 \le \mu \le m-1$	(5,3)	(0.67,0.22,0.07,0.02,0.01)

the values of the parameters for each example and the fifth column contains the pmfs calculated based on the parameters in column 4. The pmfs displayed in figure 4.1 are the same as the those displayed in column 5.

4.3 Creation of Profiles.

To create the q profiles corresponding to an instance of the q-mode problem we use the following strategy. The first step in this strategy is selecting a pmf type. In the second step, we identify n sets of pmfs, one set of pmfs corresponding to each position. Each set has at least q pmfs. Finally, in the third step we create the profiles one position at a time by assigning one pmf from the corresponding set of pmfs to every profile. Below, we explain the steps in this strategy in more detail.

The first step in this strategy is very clear. We have to choose one pmf type from the six types that we have described earlier. The pmf type Uniform assigns equal probability to all a-values, and vectors created using profiles that have the Uniform pmf type will not contain natural clusters. Hence we eliminate Uniform from the possible choices of pmf types. Note that according to this strategy all pmfs belonging to the q profiles that are used to create an instance have the same pmf type.

The second step in the strategy is to identify n sets of pmfs such that each set contains at least q pmfs. To create natural clusters in the instance, we need to create profiles which are dissimilar. We create dissimilar profiles by creating each set of pmfs in such a manner that the pmfs belonging to a set are as *different* from each other as possible. This is achieved by ensuring that no two pmfs belonging to the same position assign the highest probability to the same a-value. Below we describe a method that we shall refer to as **scheme I**, that we use to create a set of pmfs.

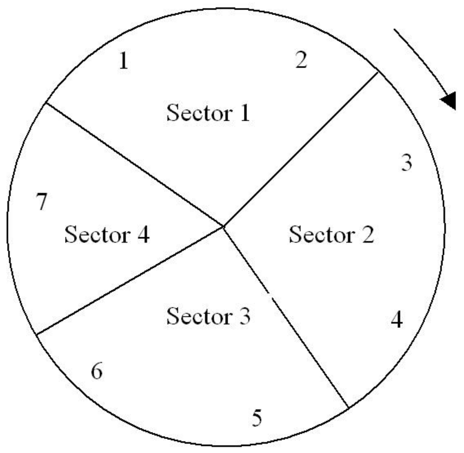

Figure 4.2: Creation of pmf sets.

Scheme I: This scheme expects as input the values of the parameters m, n, p and q, the pmf type, and the values of the parameters corresponding to that pmf type. This scheme creates q different permutations of the m a-values such that the same a-value does not have highest probability in more than one pmf. If more than one a-value have the same "highest" probability in a particular pmf than we make sure that all such a-values for each position would be different in different subsets. We begin by arranging the a-values in a circle in clockwise fashion. We divide the circle into sectors. The first q sectors have $\left\lfloor \frac{m}{q} \right\rfloor$ $(= f)$ a-values each and the last sector has $(m - f \times q)$ a-values. If $m = f \times q$ then we have only q sectors otherwise we will have $(q{+}1)$ sectors. We now identify q permutations of the a-values in the following manner. To obtain the l^{th} permutation we start from the first a-value in the l^{th} sector and keep adding the a-values to the permutation vector as we traverse the circle in clockwise direction. The first element in the l^{th} permutation will be the $((l-1)f+1)^{th}$ a-value. Next, we use the appropriate formula given in table 4.1 to calculate the probabilities corresponding to the pmf type that has been selected. We arrange these probabilities in decreasing order. The l^{th} pmf is obtained by assigning the first probability to the first a-value in the l^{th} permutation, the second probability to the second value and so on. Using this scheme the maximum number of different pmfs that can be created for each of the pmf type is equal to the maximum number of different permutations that we can get with the above procedure, which is m (and in case of pmf type oneStep it is $\lfloor \frac{m}{k} \rfloor$). Thus this scheme is valid only if $q \leq m$ (and in case of pmf type oneStep, $q \leq \lfloor \frac{m}{k} \rfloor$).

As an example, let us consider that we have to create $q(=3)$ pmf sets and there are $m(=7)$ possible values, and we have chosen pmf type Geometric, with parameters $(m=7, r=3)$. We will demonstrate the steps in scheme I to obtain one pmf set. The total number of a-values in each sector is given by $f = \left\lfloor \frac{m}{q} \right\rfloor = \lfloor \frac{7}{3} \rfloor = 2$. We arrange the a-values in a circle and divide the circle into four sectors as shown in figure 4.2. The first sector has a-values 1 and 2, the second sector has a-values 3 and 4, the third sector has a-values 5 and 6, and the fourth sector has a-value 7. The three permutations that we get along with the three corresponding pmfs are displayed in figure 4.3. We apply scheme I n times to obtain n pmf sets, one pmf set for each of the n positions in a vector. This is the end of step 2.

In the third step, using these n pmf sets the q profiles are created one position at a time. For each position j, we randomly select one pmf from the j^{th} pmf set and assign it as the j^{th} pmf of the first profile. This pmf is then removed from the pmf set. From the remaining pmfs in the pmf set we randomly choose another pmf and assign it to the j^{th} position of the second profile. We continue in this manner until the j^{th} position of each one of the q profiles has been assigned a pmf. We repeat this procedure for all n positions i.e., until each of the q profiles has n pmfs. Presently,

	permutation	pmf
1	$(1, 2, 3, 4, 5, 6, 7)$	$(0.666, 0.222, 0.074, 0.024, 0.0082, 0.0027, 0.009)$
2	$(3, 4, 5, 6, 7, 1, 2)$	$(0.0027, 0.009, 0.666, 0.222, 0.074, 0.024, 0.0082)$
3	$(5, 6, 7, 1, 2, 3, 4)$	$(0.024, 0.0082, 0.0027, 0.009, 0.666, 0.222, 0.074)$

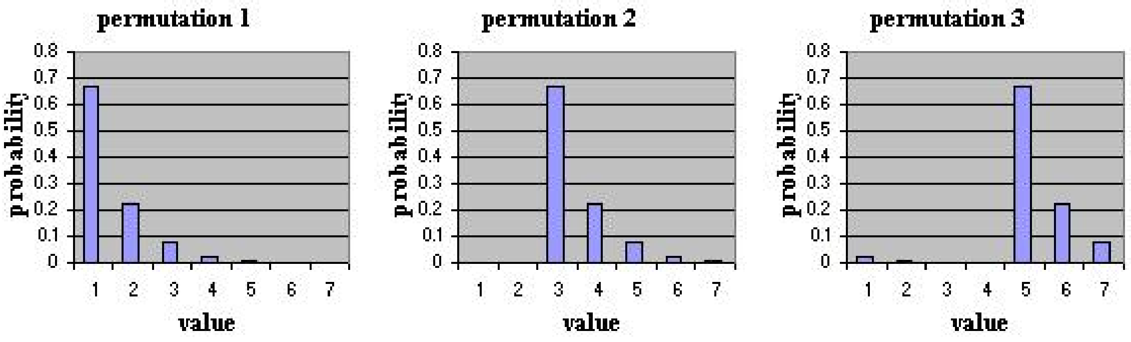

Figure 4.3: Creation of pmf sets.

the n pmf sets that we create are exact copies of each other because of two reasons. The first reason is that we use one pmf type for every instance and the second reason is the use of scheme I for creating a pmf set. We can create different pmf sets for each of the n positions by either using a different pmf type for each position or using different schemes for creating the pmf sets for different positions.

The profiles we create are such that no two pmfs present in the same position in different profiles have the same a-value with the highest probability. Let us assume that pmf 1 shown in figure 4.3 is assigned to the first position of profile 1. Further let pmf 2 and pmf 3 be assigned to the first position of profile 2 and profile 3 respectively. We know that for all vectors created using pmf 1 the probability that position 1 has a-value equal to 1 is 0.666 and the probability that position 1 has a-value equal to 2 is 0.222, because $\pi_{111} = 0.666$ and $\pi_{211} = 0.222$. Hence, the vectors created using profile 1 will be such that around 88% of them will have a-value either equal to 1 or equal to 2 in their first position. While close to 12% of these vectors will have a-values other than 1 and 2 at the first position. Similarly, a-values 3 and 4 will be most frequent at the first position in vectors created using profile 2 and values 5 and 6 will be most frequent in the first position of all vectors created using profile 3. In this manner vectors created from the same profile will be similar to each other while those created using different profiles will be dissimilar. The degree of dissimilarity will be affected by the difference in magnitute of the different probabilities i.e., the pmf type and the value of corresponding parameters for that pmf type.

4.4 Classification of the Problem Instances.

The artificially created problem instances can be classified by pmf type, by the parameters m, n, p, and the number of subsets (i.e., number of profiles used to create records) in the instance.

The instances that we create can be grouped in such a manner that all instances that use profiles having the same pmf type are together. We shall refer to these groups of instances as group G_l, group G_h, group G_s, group G_p and group G_g corresponding to the pmf types Linear, oneHuge, oneStep, partLinear and Geometric, respectively. The pmf type used affects the strength of the clusters in the instance and hence instances present in different groups differ significantly in the strength of the clusters.

The parameters m, n and p define the size of the problem instance. Hence when the instances need to be grouped by size we put all instances with the same values of m, n and p together. For the instances that we create the values of m, n and p that we use are given in table 4.2. The first column lists the 5 pmf types, the second column lists the different values of the parameters m, n and p used in creating the instances. Finally, the third column lists the number of subsets, i.e., the number of profile used in creating all the records belonging to that instance. Further, the parameter r takes value equal to 3 for all instances created using pmf types Linear, partlinear, oneHuge and Geometric. For all instances created using pmf type partLinear and oneStep the parameter k is assigned the same value as the number of subsets that we intend to create in that instance.

The artificial instances are named based on the type of pmf used and the param-

Table 4.2: Parameter values for creating artificial instances

pmf type	**Instance size**			**Num. of subsets** (q)
Linear	m	n	p	2
oneHuge	8	10	10	3
partLinear	10	20	20	5
oneStep	20	20	50	8
Geometric	20	20	100	

eters m, n, p and q. The first letter in the name identifies the pmf type, we use 'p' for partLinear, 's' for oneStep, 'g' for Geometric, 'h' for oneHuge and 'l' for Linear. The remaining part of the name comes from appending the parameters m, n, p and q and the number 1 in that order to the first letter, each separated by the character '-'.

The parameters and the scheme I used in the creation of these pmfs force the groups G_s and G_p to have very strong clusters, group G_g has instances with weaker clusters, group G_h has still weaker clusters and finally group G_l has the weakest clusters. The names of all instances created are displayed in table 4.3. Every column in this table corresponds to one group of instances (based on the pmf type) and each column contains all instances belonging to the corresponding group. The instances are arranged in the column in the increasing order of size.

Table 4.3: Names of all the artificial instances.

Group G_l	Group G_h	Group G_g	Group G_s	Group G_p
l-8-10-10-2-1	h-8-10-10-2-1	g-8-10-10-2-1	s-8-10-10-2-1	p-8-10-10-2-1
l-8-10-10-3-1	h-8-10-10-3-1	g-8-10-10-3-1	s-8-10-10-3-1	p-8-10-10-3-1
l-10-20-20-2-1	h-10-20-20-2-1	g-10-20-20-2-1	s-10-20-20-2-1	p-10-20-20-2-1
l-10-20-20-3-1	h-10-20-20-3-1	g-10-20-20-3-1	s-10-20-20-3-1	p-10-20-20-3-1
l-20-20-50-2-1	h-20-20-50-2-1	g-20-20-50-2-1	s-20-20-50-2-1	p-20-20-50-2-1
l-20-20-50-3-1	h-20-20-50-3-1	g-20-20-50-3-1	s-20-20-50-3-1	p-20-20-50-3-1
l-20-20-50-5-1	h-20-20-50-5-1	g-20-20-50-5-1	s-20-20-50-5-1	p-20-20-50-5-1
l-20-20-100-2-1	h-20-20-100-2-1	g-20-20-100-2-1	s-20-20-100-2-1	p-20-20-100-2-1
l-20-20-100-3-1	h-20-20-100-3-1	g-20-20-100-3-1	s-20-20-100-3-1	p-20-20-100-3-1
l-20-20-100-5-1	h-20-20-100-5-1	g-20-20-100-5-1	s-20-20-100-5-1	p-20-20-100-5-1
l-20-20-100-8-1	h-20-20-100-8-1	g-20-20-100-8-1	s-20-20-100-8-1	p-20-20-100-8-1

Chapter 5

Computational Experiments for Algorithms based on the MIP Models.

In chapter 3 we have discussed three (MIP) formulations for the q-mode problem. The LP relaxations of these MIP formulations provide a lower bound for the optimal objective value. Any one of these MIP formulations can be used as part of a branch and bound algorithm to solve the q-mode problem optimally and we conduct a computational experiment designed to identify the formulation that will be most appropriate for this purpose. As mentioned before, we refer to the LP relaxations of the MIP formulations (IPT), (IPR) and (IPA) as (LPT), (LPR) and (LPA) respectively, and the optimum objective values of any model (B) is referred to as $v(B)$, where (B) can take any of the following 6 values, (IPT), (IPR), (IPA), (LPT), (LPR) and (LPA). Once we identify an MIP formulation that is most appropriate for use in the context of a branch and bound algorithm, we use that formulation to solve a collection of instances of the q-mode problem optimally.

Apart from comparing the lower bounds obtained from the LP relaxations of the MIP formulations we also conduct a computational experiment to evaluate the performance of the algorithms based on the Benders' decomposition for formulation (IPR) that we discussed in section 3.2, i.e., algorithms B_{opt} and B_{Nopt}. These two algorithms iteratively solve the relaxed master problem (BM_r) and primarily differ in the strategy for obtaining a solution for (BM_r). Both algorithms, B_{opt} and B_{Nopt}, provide a lower bound for the optimal solution that we shall refer to as LB_M and the objective value of the best integer solution found during a run of the algorithm will be referred to as ν_M. The time taken is referred to as τ_M and the total number of iterations performed by the algorithm is referred to as r.

In this chapter, the first section introduces the performance measures that we use to analyze the quality of the lower bounds and the performance of the algorithms related to the Benders' Decomposition. The details and the discussion related to the computational experiments are presented in sections 5.2, 5.3 and 5.4.

5.1 Performance Measures.

We know from preliminary computational results that $v(LPT)$ is larger than both, $v(LPA)$ and $v(LPR)$. Hence we use $v(LPT)$ as the benchmark for evaluating the lower bounds obtained from the LP relaxations of the MIP formulations (IPA) and (IPR). These lower bounds are evaluated based on their deviation from the corresponding value of $v(LPT)$ for every instance in the computational experiment. The performance measure that we use to compare the quality of these lower bounds is the fractional deviation of the lower bound under consideration from $v(LPT)$. We refer to this performance measure as δ_A and δ_R corresponding to the formulations (IPA) and (IPR) respectively. This performance measure is defined as

$$\delta_A = \frac{v(LPT) - v(LPA)}{v(LPT)}, \quad \delta_R = \frac{v(LPT) - v(LPR)}{v(LPT)} \tag{5.1}$$

We also compare the execution time in seconds, τ_T, τ_A and τ_R, required to solve (LPT), (LPA) and (LPR) respectively. Another performance measure that we use is the fractional deviation of a lower bound obtained for each instance from the optimal objective value of that instance. We shall refer to this performance measure as Δ_T^*.

$$\Delta_T^* = \frac{\nu^* - v(LPT)}{\nu^*}, \tag{5.2}$$

where, ν^* is the optimal value for an instance of the q-mode problem.

5.2 Comparison of the Lower Bounds.

The current section and the two sections that follow correspond to the three computational experiments that we conduct. In the current section, we compare the lower bounds $v(LPT)$, $v(LPA)$ and $v(LPR)$. Next, in section 5.3, we discuss the results of a computational experiment where we use the MIP formulation (IPT) corresponding to each instance to optimally solve the randomly generated instances via the CPLEX branch and cut algorithm. Finally, in section 5.4 we compare the performance of the two algorithms B_{opt} and B_{Nopt}. All the computational experiments are conducted on a Sun Solaris Sun Blade 100 machine and the branch and cut solver provided as part of CPLEX version 8.0 is used for solving the LP relaxations, to find optimal integer solutions and also to solve the relaxed master problem in the algorithms related to the Benders' decomposition.

The comparison of the lower bounds obtained using the linear programming relaxations (LPT), (LPA) and (LPR) is presented in table 5.1. There are 50 different problem instances used in this comparison. Based on parameter p, i.e., the number of records in the instance and parameter q, i.e., the total number of clusters to be formed out of the p records we divide these instances into sets. Thus every instance belonging to a particular set has the same value of p and q. Furthermore, every set contains one instance from each of the five groups, G_p, G_s, G_g, G_h, and G_l described earlier in chapter 4. Hence, there are a total of 10 sets and each set has 5 instances. Every row in table 5.1 corresponds to one set and displays values that are averages taken over all the instances belonging to that set. For every instance in this computational experiment we partition the records into the same number of clusters as the number of profiles used to create the records in that instance. The first column in the table identifies the set and displays the corresponding parameters m, n, p and q. The second column gives the number of instances belonging to that set (this is 5 for every set). The third column labeled LPT contains two sub-columns. The sub-column labeled $\overline{v(LPT)}$ displays the average $v(LPT)$ taken over all 5 instances in each set. The sub-column labeled $\overline{\tau}_T$ displays the average time taken to calculate $v(LPT)$ taken over all instances in the set. The next two columns labeled LPA and LPR, correspond to formulation (LPA) and (LPR) respectively, and they contain the average values $\bar{\delta}_A$, $\overline{\tau}_A$, $\bar{\delta}_R$, and $\overline{\tau}_R$, as shown. Furthermore, we also display the values of δ_A and δ_R in the form of a graph which has the δ-values on the vertical axis and each of the 50 problem instances on the horizontal axis. This graph is displayed in figure 5.1.

Table 5.1: Comparison of the LP relaxation of the all the MIP formulations.

Problem size				Num	LPT		LPA		LPR	
m	n	p	q		$\overline{v(LPT)}$	$\overline{\tau}_T$	$\bar{\delta}_A$	$\overline{\tau}_A$	$\bar{\delta}_R$	$\overline{\tau}_R$
8	10	10	2	5	38.60	0.06	0.28	0.04	0.50	0.05
8	10	10	3	5	25.20	0.12	0.84	0.09	0.67	0.11
10	20	20	2	5	181.80	0.63	0.20	0.2	0.50	0.47
10	20	20	3	5	153	2.34	0.37	0.72	0.67	1.58
20	20	50	2	5	544.6	3.61	0.10	0.62	0.50	3.18
20	20	50	3	5	508.6	18.39	0.16	2.17	0.67	13.27
20	20	50	5	5	451.4	83.65	0.39	13.85	0.80	57.20
20	20	100	2	5	1118.8	16.45	0.06	0.91	0.50	15.92
20	20	100	3	5	1073.4	79.09	0.15	5.23	0.67	68.97
20	20	100	5	5	965.2	523.85	0.27	43.18	0.80	300.48

We summarise the observations that we make from table 5.1 as follows.

1. It is very clear from table 5.1 that on the average, the time required to obtain $v(LPT)$ increases with increase in the size of the problem instance, i.e., with increase in the value of parameter p. This is expected as a larger problem instance corresponds to larger number of variables and constraints in the (IPT) formulation for that instance.

2. All other parameters remaining the same, on the average, the time required to calculate $v(LPT)$ increases with increase in the parameter q. Increase in the value of parameter q leads to increase in the number of variables and the number of constraints in formulation (IPT).

3. This trend, where the time required increases with increase in parameters p and q, is observed for (LPA) and (LPR) too.

4. The lower bound $v(LPA)$ is worse than lower bound $v(LPT)$ for all 10 sets as indicated by the positive value of δ_A for all sets. Further, $\overline{\delta}_A$ ranges between 0.06 and 0.84.

5. The formulation (IPA) has the least number of constraints when compared to the other 2 formulations and as expected the solution time corresponding to (LPA) is the least among all the formulations for all groups of instances.

6. The formulation (IPR) shows the worst overall performance, the average value of $v(LPR)$ is worse than the average of $v(LPT)$ for all 10 sets and the average solution time for each set is comparable to that of formulation (IPT).

7. As a comparison between (LPT) and (LPA), the time required for obtaining $v(LPA)$, on the average, is considerably less than the time for $v(LPT)$. But $v(LPA)$ is much worse than $v(LPT)$ in many instances and hence a branch and bound algorithm based on (IPA) can potentially have a lot more nodes as compared to a branch and bound algorithm based on (IPT). And even though the sub-problem at each node can be solved faster in case of (IPA)

the algorithm could potentially take a large amount of time due to the large number of branch and bound nodes that need to be enumerated. In a limited computational experiment in which the same collection of instances are solved using their respective (IPA) and (IPT) models, we observed that solving the (IPA) model indeed requires significantly larger computation time.

From the above discussion it is clear that formulation (IPT) is the most appropriate, among the MIP formulations that we have discussed, to be used in the context of a branch and bound algorithm to optimally solve instances of the q-mode problem. This is re-enforced by the graph in figure 5.1 which clearly shows that δ_A and δ_R values are strictly positive for all 50 instances. In fact, for all 50 instances $\delta_R \geq 0.5$ and hence for all the instances $v(LPT)$ is greater than or equal to twice the value of $v(LPR)$.

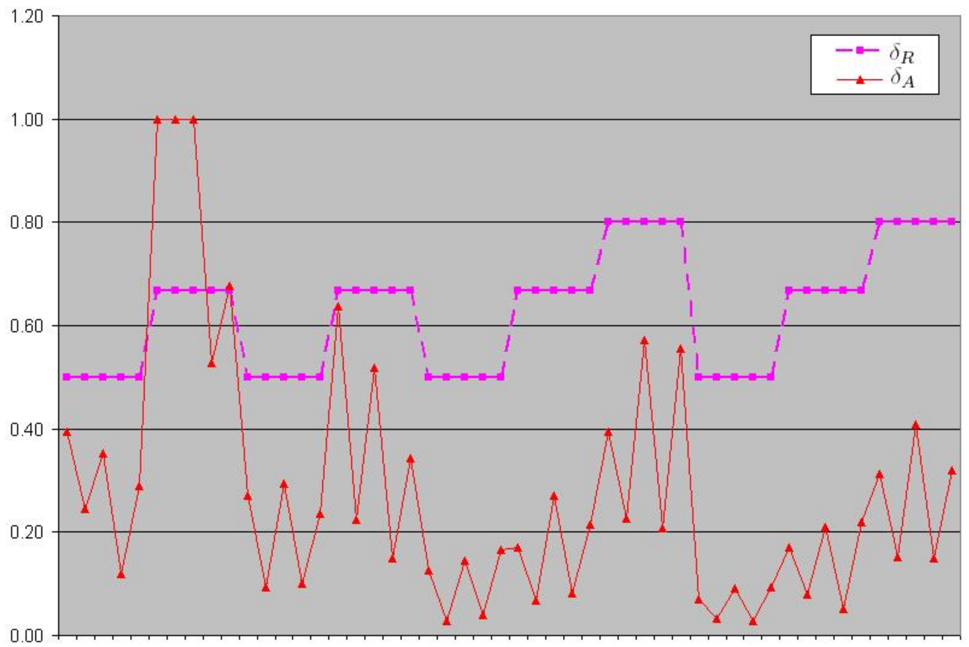

Figure 5.1: Comparison of the Linear Programming Relaxations.

5.3 Optimal Solutions using Formulation (IPT)

We use the MIP formulation (IPT) to solve a collection of randomly generated instances of the q-mode problem. The MIP formulation of each instance is submitted to the CPLEX 8.0 branch and cut solver. The results of this computational experiment are presented in table 5.2 and table 5.3. In table 5.2, we include the results corresponding to all problem instances in which we find the optimal solution within 3600 seconds of computer time and the results related to the remaining instances are presented in table 5.3. Table 5.2 contains 8 columns and each row in this table corresponds to one problem instance. The first column contains the name of the problem instance. The second and third columns display the optimal objective value of the LP relaxation, $v(LPT)$ and the execution time for the LP relaxation, τ_T, respectively, for the corresponding problem instance. The fourth and fifth columns display the optimal value ν^* for the problem instance and the solution time, τ^*, required by the CPLEX branch and cut solver, respectively. The total number of simplex iterations and total number of branch and bound nodes are displayed in columns six and seven which are labeled as "Simplex Iter." and "B&B nodes", respectively. Finally, the last column displays the fractional deviation of the lower bound $v(LPT)$ from the optimal solution ν^*, i.e., Δ_T^*. Moreover, the instances are arranged in table 5.2 in such a manner that all the instances belonging to the same group (by pmf type) are together.

All instances such that the CPLEX branch and cut algorithm did not terminate with an optimal solution within 3600 seconds are included in table 5.3. For these instances, the CPLEX branch and cut algorithm was run for exactly 3600 seconds. After such a premature termination, instead of the optimal soution, the CPLEX algorithm provides the objective value of the best integer feasible solution. We refer to this value as ν^b. The execution time is referred to as τ^b and it equals 3600 seconds for all instances. CPLEX also provides a lower bound for the optimum value that we refer to as $\widetilde{\nu}$. Further, we define α as $\frac{\nu^b-\widetilde{\nu}}{\nu^b}$ and note that since $\frac{\nu^b-\widetilde{\nu}}{\nu^b} \geq \frac{\nu^b-\nu^*}{\nu^b}$, α

gives an upper bound on the fractional deviation of the optimal solution from the best known solution ν^b.

The columns of table 5.3 are similar to those in table 5.2 with a few changes. The fourth column in 5.2 reports ν^b instead of ν^*. Further the column reporting Δ^* is dropped and two columns, one reporting $\widetilde{\nu}$ and the second reporting α, are added after the column that reports τ^b.

Based on tables 5.2 and 5.3 we make the following observations.

1. The last column in table 5.2 displaying Δ_T^* indicates that the value of Δ_T^* is affected by the strength of the clusters present in the problem instance. Instances belonging to group G_p and group G_s have the least value of Δ_T^*. The value of this performance measure increases further as we observe instances in group G_g which has slightly weaker clusters. Instances in group G_h have even higher values of Δ_T^* and finally instances in group G_l has the highest values. Thus the quality of the lower bound available from formulation (LPT) improves as the strength of the clusters in the instance becomes greater.

2. In some problem instances with strong natural clusters, the value of the lower bound, $v(LPT)$ equals the optimal value ν^*. In table 5.2, we report 13 instances, all of them with strong natural clusters, where the lower bound equals the optimal solution. In all these instances the CPLEX branch and cut algorithm terminates without having to create any branch and bound node, i.e., the optimal integer solution is available at the root node in the branch-and-cut tree.

3. Instances of the same size but with strong natural clusters have smaller solution times as compared to instances with weaker clusters. This observation is consistent with the above two observations. Since instances with stronger clusters have a better lower bound they require fewer branches (i.e., fewer nodes) in the branch and cut algorithm leading to shorter solution times. Instances of

the same size but with weaker clusters have weak lower bounds and hence have larger solution times.

4. The solution time is affected by both parameters p, i.e., total number of records in the instance and q, i.e., number of clusters required, and increases sharply with increase in the value of both these paramaters.

5. The CPLEX branch and cut algorithm was able to obtain the optimal solution via model (IPT) in 38 problem instances within a time limit of 3600 seconds. These include instances with strong natural clusters and some smaller instances having weak natural clusters. The remaining 12 instances for which the optimal solution could not be obtained in less than 3600 seconds are larger instances with weak clusters or very large instances (having large values of both p and q) with stronger natural clusters.

6. For the instances where the optimal solution is not available in 3600 seconds, CPLEX provides a lower bound $\widetilde{\nu}$ which is higher than the previously best known lower bound, $v(LPT)$, for all the 12 instances.

7. From table 5.3, we can see that some problem instances have a large value of α, 6 instances have $\alpha > 0.1$ with the largest value being 0.263. Thus for large sized instances with higher values of q and having weak natural clusters, the best integer solution obtained by CPLEX within the time limit of 3600 seconds can have objective value that is far away from the optimal value.

8. The approach of using the CPLEX branch and cut algorithm on the (IPT) formulation for optimally solving the q-mode problem is viable for large instances if the number of clusters required is relatively small and the data contains relatively strong natural clusters.

5.4 Comparison of the Algorithms based on Benders' Decomposition.

In order to explore the computational requirements of the algorithms based on Benders decomposition, i.e., B_{opt} and B_{Nopt}, we perform a limited computational experiment in which we solve a collection of randomly generated instances of the q-mode problem using the two algorithms. Since the solution time for both algorithms is large, we have included only small size instances in this experiment. Every instance in this collection is also solved using the corresponding (IPT) model via CPLEX and hence we know the corresponding optimal value.

The results are summarised in table 5.4. Each row in this table corresponds to a problem instance. The first column identifies the instance along with the optimal value, ν^*. Every instance is solved using both algorithms, B_{opt} and B_{Nopt}, described earlier. For this computational experiment we terminate the algorithm once the execution time exceeds 300 seconds. The time taken is counted at the end of each iteration and hence the algorithms run for slightly more than 300 seconds for each instance. In some instances the algorithm terminates within 300 seconds as it can be guaranteed that the best integer solution found is indeed optimal and hence the algorithm has running time which is less than 300 seconds for those instances. For each instance and for each algorithm, we report the following values in its corresponding row; the best solution obtained via the Benders' algorithm, ν_M, the lower bound obtained via the Benders' algorithm, LB_M, the fractional deviation of the lower bound from the optimal, Δ^*_M $(= \frac{\nu^* - LB_m}{\nu^*})$, the total execution time in seconds, τ_M, and the total number of iterations r. When we run B_{Nopt} the relaxed master problem BM_r is solved sub-optimally at each iteration and to obtain a valid lower bound we need to run BM_r optimally at the last iteration and the time required for the last iteration is referred to as t_{ip}. For the algorithm B_{Nopt} we also include the value of t_{ip} in table 5.4. The observations based on table 5.4 are presented below.

1. Both algorithms, B_{opt} and B_{Nopt}, terminate with the optimal solution for 4 instances within the time limit of 300 seconds. These four instances are p-8-10-10-2-1, s-8-10-10-2-1, g-8-10-10-2-1 and h-8-10-10-2-1. These four instances are among the 5 smallest instances that we consider in this experiment.

2. The algorithms, B_{opt} and B_{Nopt}, require a considerably large number of iterations for termination. In the above four instances there are 10 records that need to be partitioned into 2 clusters. Thus there are $\frac{2^{10}}{2} = 512$ possible distinct solutions. In three instances, s-8-10-10-2, g-8-10-10-2 and h-8-10-10-2, both algorithms require more than 240 iterations. In other words, nearly half the total number of solutions are enumerated for the three instances.

3. For these four instances algorithm B_{Nopt} has smaller execution time than algorithm B_{opt}, even though B_{Nopt} requires more iterations than B_{opt}.

4. In fact, for all instances in this computational experiment B_{Nopt} goes through more iterations than B_{opt} and finds a lower bound that is at least as good as the lower bound found by algorithm B_{opt}.

5. Clearly, as the problem size increases the relaxed master problem becomes more time consuming to solve optimally at each iteration and hence as the problem size increases the number of iterations that algorithm B_{opt} can perform in 300 seconds become considerably less than the number of iterations performed by B_{Nopt}.

6. The fact that the relaxed master problem takes a considerable time to solve can also be seen by observing that the time required (t_{ip}) to solve BM_r in the last iteration for algorithm B_{Nopt} increases with the size of the instance.

7. For all instances in this table we can observe that the objective value of the best solution found by B_{Nopt} is no worse than the objective value of the best solution found by B_{opt}. Since B_{Nopt} requires less time to solve the relaxed master

problem at each iteration it performs more iterations than B_{opt} within the time limit of 300 seconds. Thus it effectively enumerates more solutions ($y \in Y$) than B_{opt} and consequently the best value is no worse than B_{opt}.

8. In 9 out of the 11 instances reported in table 5.4, the algorithm B_{Nopt} finds the optimal solution within the 300 seconds time limit; since the corresponding lower bound is still less than the optimal value, the algorithm does not terminate.

9. Computationally, both algorithms based on Benders decomposition are ineffective in their present form. In order to improve the effectiveness of these algorithms a more efficient procedure to solve the relaxed Benders' master problem is required.

10. For the four instances where algorithms B_{Nopt} and B_{opt} terminate with a guaranteed optimal solution, the number of constraints in the relaxed master master problem in the final iteration for algorithm B_{Nopt} is more than that for B_{opt}. Thus there are a lot of redundant constraints in the case of B_{Nopt}. It is likely that B_{opt} also has some redundant constraints in the final relaxed master problem that it solves. To obtain good quality lower bounds in reasonable time, we need to design a procedure that efficiently identifies a few "good" solutions, i.e., solutions such that the relaxed master problem formed with the constraints generated from these solutions either provides a guaranteed optimal solution to the q-mode problem or at least provides a good quality lower bound.

Table 5.2: Optimal solutions using formulation (IPT)

type-m-n-p-q-1	$v(LPT)$	τ_T	ν^*	τ^*	Simplex Iter.	B&B nodes	Δ_T^*
l-8-10-10-2-1	45	0.07	55	0.59	1154	33	0.182
l-8-10-10-3-1	29	0.12	45	6.90	10664	353	0.356
l-10-20-20-2-1	246	0.68	273	36.72	48263	1509	0.099
h-8-10-10-2-1	34	0.06	41	0.35	477	6	0.171
h-8-10-10-3-1	25	0.13	37	4.30	5523	152	0.324
h-10-20-20-2-1	153	0.59	172	2.54	1726	14	0.110
h-10-20-20-3-1	150	2.28	195	330.58	166193	1320	0.231
h-20-20-50-2-1	463	2.58	494	24.85	8242	126	0.063
h-20-20-100-2-1	916	6.66	968	190.02	64837	1022	0.054
g-8-10-10-2-1	33	0.06	40	0.37	476	6	0.175
g-8-10-10-3-1	20	0.12	26	1.93	1299	11	0.231
g-10-20-20-2-1	126	0.55	127	0.66	851	0	0.008
g-10-20-20-3-1	112	2.02	121	15.66	3761	13	0.074
g-20-20-50-2-1	494	4.28	494	4.28	2346	0	0
g-20-20-50-3-1	471	18.93	475	69.36	7715	10	0.008
g-20-20-50-5-1	437	84.63	464	> 2400	329186	628	0.058
g-20-20-100-2-1	984	14.76	984	15.50	4018	0	0
g-20-20-100-3-1	982	100.78	991	250.44	16554	10	0.009
s-8-10-10-2-1	50	0.07	50	0.08	260	0	0
s-8-10-10-3-1	28	0.13	28	0.15	374	0	0
s-10-20-20-2-1	248	0.66	251	1.86	1249	2	0.012
s-10-20-20-3-1	188	2.48	193	13.85	3680	4	0.026
s-20-20-50-2-1	639	4.42	641	11.39	3064	2	0.003
s-20-20-50-3-1	616	18.83	622	86.57	7745	13	0.010
s-20-20-50-5-1	573	80.86	578	903.80	89881	209	0.009
s-20-20-100-2-1	1343	18.79	1344	14.57	5232	0	0.001
s-20-20-100-3-1	1303	96.17	1310	329.96	22976	34	0.005
s-20-20-100-5-1	1256	571.99	1260	> 2400	126188	170	0.003
p-8-10-10-2-1	31	0.06	31	0.08	224	0	0
p-8-10-10-3-1	24	0.12	28	1.69	919	7	0.143
p-10-20-20-2-1	136	0.67	136	0.71	993	0	0
p-10-20-20-3-1	125	2.12	125	2.42	1684	0	0
p-20-20-50-2-1	345	4.62	345	4.70	2038	0	0
p-20-20-50-3-1	341	19.24	341	22.65	5078	0	0
p-20-20-50-5-1	327	80.46	327	103.64	12006	0	0
p-20-20-100-2-1	730	21.52	730	21.87	4191	0	0
p-20-20-100-3-1	732	93.19	732	111.93	11321	0	0
p-20-20-100-5-1	713	527.22	713	528.72	25475	0	0

Table 5.3: Near Optimal solutions using formulation (IPT)

type-m-n-p-q-1	$v(LPT)$	τ_T	ν^b	τ^b	$\widetilde{\nu}$	α	Simplex Iter.	B&B nodes
l-10-20-20-3-1	190	2.79	251	3600	235.66	0.061	3056039	33596
l-20-20-50-2-1	782	2.13	820	3600	809	0.013	1883885	43919
l-20-20-50-3-1	686	19.86	790	3600	701.66	0.111	584417	3850
l-20-20-50-5-1	556	93.85	757	3600	557.66	0.263	245150	279
l-20-20-100-2-1	1621	12.15	1710	3600	1641.5	0.040	883583	21965
l-20-20-100-3-1	1471	28.03	1670	3600	1479.83	0.114	247039	1040
l-20-20-100-5-1	1215	559.85	1651	3600	1243.2	0.247	63281	21
h-20-20-50-3-1	429	15.09	491	3600	483.66	0.015	687403	3462
h-20-20-50-5-1	364	78.45	486	3600	384.6	0.209	558029	1165
h-20-20-100-3-1	879	77.3	984	3600	921.84	0.063	479426	2361
h-20-20-100-5-1	742	419.46	945	3600	747.73	0.209	138194	120
g-20-20-100-5-1	900	511.62	973	3600	910.60	0.064	117360	80

Table 5.4: Results for the Benders' algorithm

		B_{opt}					B_{Nopt}					
type-m-n-p-q-1	ν^*	ν_M	LB_M	Δ^*_M	τ_M	r	ν_M	LB_M	Δ^*_M	τ_M	r	t_{ip}
l-8-10-10-2-1	55	55	49	0.109	301.31	272	55	49	0.109	300.67	551	4.78
h-8-10-10-2-1	41	41	41	0	216.54	244	41	41	0	120.3	323	-
g-8-10-10-2-1	40	40	40	0	212.24	245	40	40	0	151.94	341	-
s-8-10-10-2-1	50	50	50	0	210.42	271	50	50	0	152.74	349	-
p-8-10-10-2-1	31	31	31	0	13.46	69	31	31	0	6.86	74	-
g-8-10-10-3-1	26	38	10	0.615	301.93	79	26	12	0.538	301.35	353	17.51
s-8-10-10-3-1	28	35	14	0.643	303.38	87	28	17	0.393	300.89	355	24.93
p-8-10-10-3-1	28	38	10	0.5	306.95	81	28	14	0.5	300.36	354	14.81
g-10-20-20-2-1	127	127	93	0.268	325.8	52	127	104	0.181	300.82	290	73.05
s-10-20-20-2-1	251	270	154	0.386	301.81	50	264	176	0.299	301.87	326	120.91
p-10-20-20-2-1	136	163	93	0.316	320.45	56	147	109	0.199	300.67	298	90.26

Chapter 6

A local improvement algorithm based on a very large scale neighborhood.

In this chapter, we describe a local improvement algorithm that we have designed for the q-mode problem. The first section describes the *solution representation*. This local improvement algorithm uses a very large scale neighborhood structure that we describe in the second section. The third section is devoted to the explanation of the move mechanism that we refer to as cyclic-exchanges. The large scale neighborhood is implicitly searched by creating an *improvement graph* associated with the current solution; this is discussed in section 4. Further in section 5, we discuss the procedure used to search this improvement graph for minimum cost cluster disjoint cycles. The basic steps of the local improvement algorithm are presented in section 6 and three alternative search strategies are explained in the last section.

6.1 Solution Representation.

The q-mode problem requires an assignment of records to clusters such that the sum of the total number of replacements corresponding to each cluster is minimized. Any assignment where every record is assigned to a unique cluster is a feasible solution to the problem. Therefore a feasible solution can be represented as an array C having p elements (i.e., the number of elements of the vector C equals the total number of records in the problem instance). The k^{th} element of the array, $C[k]$, represents the k^{th} record and identifies the cluster l to which the record k is assigned, i.e., $C[k] = l$. Let us assume for the purpose of demonstration that we have a problem instance containing $p = 6$ records that are to be partitioned into $q = 3$ clusters. Let us assign records k_1 and k_2 to cluster l_1, records k_3 and k_4 to cluster l_2, and records k_5 and k_6 to cluster l_3, as shown in figure 6.1. This represents a feasible solution which we shall refer to as S_1. The vector C for feasible solution S_1 will be $(1, 1, 2, 2, 3, 3)$.

For an instance of the q-mode problem, if we are given the vector C corresponding to a solution S, then we can identify all the records that belong to each cluster in solution S. Now, the total number of replacements corresponding to solution S can be obtained by finding the mode corresponding to each cluster. Thus an instance of the q-mode problem is transformed into a problem of finding q modes. Morgan et al. [17] describe an efficient algorithm, having computational requirement $o(np)$, to obtain the mode of a collection of records, where n is the number of positions in a record and p is the number of records. Thus, once the vector C is known a calculation having computational requirement $o(qnp)$ is needed to obtain the corresponding modes and the objective value for solution S.

6.2 Neighborhood Definition.

A neighborhood structure for a combinatorial optimization problem is defined by the *move mechanism*, i.e., the sequence of steps carried out to obtain a different feasible

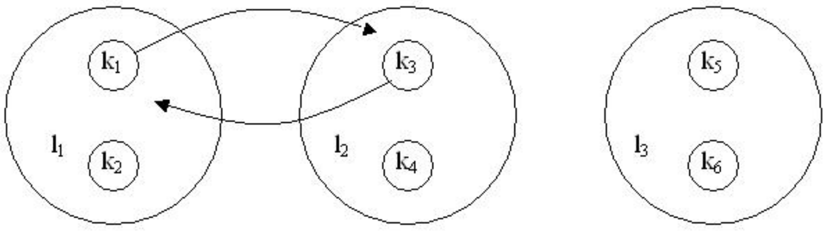

Figure 6.1: Example of a 2-exchange move in solution S.

solution from a given feasible solution. The move mechanism that we adopt for the q-mode problem involves exchange of records between different clusters. Let us consider that we are given a collection of p records, Φ, and a feasible solution S, i.e., an assignment of each of the p records to one of the q clusters. The removal of a record k_i from the cluster to which it is presently assigned and its reassignment to a different cluster l is represented as $(k_i \rightarrow \Phi_l)$. Further, if record k_i is moved from its present cluster and reassigned to the cluster that contains record k_j then this is represented as $(k_i \rightarrow k_j)$. The moves described above require the reassignment of just one record. We now describe moves that require reassignments of more than one record.

In solution S, let record k_i belong to cluster l_i and let record k_j belong to cluster l_j. We can obtain a new feasible solution S' from S by exchanging the cluster assignments of records k_i and k_j. This move would be referred to as a *2-exchange move* and is represented as $(k_i \rightarrow k_j \rightarrow)$. All the solutions obtained by all possible 2-exchanges would constitute a *2-exchange neighborhood*. This is a very popular neighborhood structure that has been used in the context of several partitioning problems. The 2-exchange move is demonstrated through an example in figure 6.1. The figure shows a feasible solution to a problem containing 6 records, $(k_1, \ldots, k_6)$, that need to be partitioned into 3 clusters, $(l_1, l_2$ and $l_3)$. The two arrows indicate a 2-exchange move that we represent as $(k_1 \rightarrow k_3 \rightarrow)$.

The 2-exchange idea can be extended by including more than 2 clusters in the exchange, to obtain what is known as a *cyclic-exchange*. The cyclic-exchange involves a transfer of records represented as $(k_1 \rightarrow k_2 \rightarrow k_3 \rightarrow \cdots \rightarrow k_r \rightarrow)$, i.e., record k_1 is assigned to the cluster that contained record k_2, record k_2 is assigned to the cluster that contained record k_3 and so on till the last record k_r is assigned to the cluster that contained the first record k_1. Each record in a cyclic-exchange belongs to a different cluster. Hence, no more than one record per cluster is involved in a single cyclic-exchange. Thus the total number of records (and clusters) in a cyclic-exchange is no more than q and no less than 2, i.e., $2 \leq r \leq q$. In figure 6.2, the three filled arrows together represent a cyclic-exchange of three records which will be represented as $(k_1 \rightarrow k_3 \rightarrow k_5 \rightarrow)$.

The cyclic-exchange can be modified slightly to obtain what is called a *path-exchange*. A path-exchange is an exchange such that, similar to the cyclic exchange, record k_1 is assigned to the cluster that contained record k_2, record k_2 is assigned to the cluster that contained record k_3 and so on till the record k_{r-1} is assigned to cluster containing record k_r, but no record moves from the last cluster (l_r) to the first cluster. Thus a path-exchange is basically a cyclic exchange where no record is re-assigned to the first cluster in the cycle and no record leaves the last cluster in the cycle. We use the notation $(k_1 \rightarrow k_2 \rightarrow k_3 \rightarrow \cdots \rightarrow k_r)$ to represent this path exchange. In every path-exchange move the size of the first cluster in the path will reduce by 1 and the size of the last cluster in the path will increase by 1. Here, the size of each cluster is given by the total number of records assigned to that cluster. Thus, path exchanges include exchanges that change the size of the clusters; on the other hand cyclic exchanges will maintain the cluster sizes as they were in the initial solution. The dashed arrows in figure 6.2 represent a path exchange that is represented as $(k_2 \rightarrow k_4 \rightarrow k_6)$. Here, cluster l_1 will reduce its size by 1 while cluster l_3 will increase its size by 1.

A cyclic-exchange or a path-exchange performed on solution S_1 will give us another feasible solution S_2 and the difference in the total number of replacements corresponding to solutions S_1 and S_2 (i.e., number of replacements in solution S_2 -

number of replacements for solution S_1) is defined as the *cost of the exchange*. Thus a neighbor S_2 obtained from S_1 through a negative cost exchange is a *better* neighbor, and the neighbor corresponding to the exchange having the least cost is the *best* neighbor of solution S_1.

The cyclic exchange neighborhood is extremely large and the size of the neighborhood is exponential in the number of clusters required in the problem instance. For a problem instance with p records and q clusters, if we consider a solution S in which all the clusters have the same number of records $\frac{p}{q}$, then the total number of cyclic-exchange moves associated with the solution is $\sum_{i=2}^{q} \binom{q}{i} i! \left(\frac{p}{q}\right)^q \geq \left(\frac{p}{q}\right)^q \sum_{i=2}^{q} \binom{q}{i} = \left(\frac{2p}{q}\right)^q$. Hence the total number of cyclic exchange neighbors will be $\Omega\left(\frac{2p}{q}\right)^q$ i.e., the size of the cyclic-exchange neighborhood grows exponentially with growth in q. Here, $\binom{q}{i}$ stands for q *choose* i, $i!$ stands for factorial of i, and $\Omega(g(x))$ means asymptotically lower bounded by function $g(x)$.

In general, the cyclic-exchange neighborhood is much larger than the two exchange neighborhood and also subsumes it and hence it is reasonable to expect that the solutions obtained from the cyclic-exchange neighborhood will be much better than the 2-exchange neighborhood. However, since the neighborhood size increases exponentially with the size of the problem we need an efficient method to find a cost decreasing neighbor. A search algorithm that explicitly enumerates and evaluates all neighbors in such a large neighborhood will be computationally ineffective. In practice, we overcome this difficulty by implicitly enumerating this neighborhood via a network optimization based method that efficiently identifies a good cyclic-exchange neighbor for any given solution. This concept is explained in detail in the next subsection.

6.3 Identification of a Cyclic Exchange Neighbor.

The first step in identifying a good cyclic-exchange neighbor for a given solution S is to create an associated graph $G(S)$. Following Ahuja et al. [2] we refer to $G(S)$ as

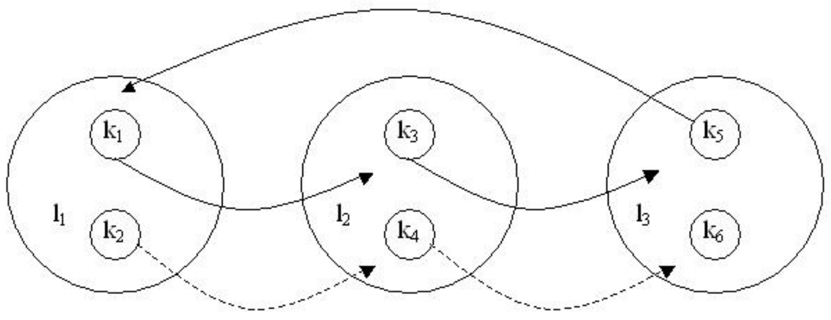

Figure 6.2: Example of cyclic and path exchange in solution S.

an *improvement graph* associated with solution S. The improvement graph $G(S)$ is a directed graph with p nodes, one node for each record k in the problem instance. Each directed edge of the graph $G(S)$ corresponds to a re-assignment of a record to a cluster different from the cluster that it is presently assigned to in the solution S. More specifically, the directed edge from node corresponding to record k_i (that is assigned to cluster l_i in S) to node k_j (that is assigned to cluster l_j in S) represents a re-assignment where record k_i leaves cluster l_i and is assigned to cluster l_j and record k_j leaves cluster l_j. We define the cost of the directed edge (k_i, k_j) to be the increase in the number of replacements in cluster l_j caused by the removal of record k_j from cluster l_j and the addition of record k_i to cluster l_j, regardless of its impact on other clusters. Clearly, the cost of an edge in the improvement graph can be negative. Moreover, a cycle in $G(S)$ that has no more than 1 record from each cluster represents a cyclic-exchange and the cost of this exchange is the sum of the cost of each edge in the cycle. Thus the cycle of lowest cost in the improvement graph $G(S)$ that has no more than one node from every cluster corresponds to the best cyclic neighbor of solution S.

Enhancements can be made to the improvement graph $G(S)$, by adding extra nodes and arcs so that every path-exchange also corresponds to a cycle in this augmented improvement graph. We shall refer to the set of nodes in the augmented improvement graph as $\mathcal{V}$ and the set of edges as $\mathcal{E}$. Given a solution S, an augmented improvement graph is constructed as follows. Consistent with the above discussion, each record k corresponds to a node in the augmented improvement graph. We refer to these nodes as *regular* nodes. Apart from the regular nodes every augmented improvement graph, $G(S)$, contains q *pseudo* nodes, one pseudo node corresponding to each of the q clusters, and exactly one *origin* node. Each regular node k is connected by an arc to every other regular node k' which does not belong to its own cluster. The cost calculation for all such arcs has been explained earlier. Further, each regular node k is connected to all pseudo nodes except for the pseudo node that corresponds to the cluster containing node k. The arc from regular node k to the pseudo node corresponding to cluster l represents the addition of record k to cluster l with no removal of records from cluster l. Thus the cost of any such arc is the increase in total replacements in cluster l caused by addition of record k to this cluster. Each pseudo node is connected to the origin node through an arc of zero cost. Finally, there is an arc from the origin node to every regular node. An arc from the origin node to regular node k represents the removal of record k from its cluster without any addition of records. Hence the cost of such an arc is the increase in total replacements in the corresponding cluster caused by removal of record k from this cluster (typically a negative value). Any cycle that contains the origin node will also include one of the pseudo nodes and it represents a path-exchange, while all cyclic-exchanges correspond to cycles in the augmented improvement graph which contain neither the origin nor any pseudo nodes.

6.4 Efficient Calculation of the Improvement Graph.

Given a solution S, where Φ_l is the set of records assigned to cluster l for $l = 1$ to q, every possible pair of records (k_1, k_2), where both records do not belong to the same cluster (i.e., $k_1 \notin \Phi_l$ and $k_2 \in \Phi_l$ for some $l = 1, \ldots, q$), corresponds to an arc in the improvement graph, $G(S)$. Thus calculation of $G(S)$ requires calculation of the cost of $O(p^2)$ arcs. A naive calculation of the improvement graph would involve calculation of a mode for calculating the cost of each arc. We can reduce the computational requirements by a two-fold strategy of increasing the memory requirements (i.e., maintaining an appropriate database) and using the fact that the arc costs are interrelated.

As defined earlier, let the number of records in cluster l having value i at position j be denoted by $F_l(i,j)$ and let $\max_i F_l(i,j)$ be denoted by $F_l^{max}(j)$. Let $inc_{jl}(k_1,k_2)$ be the increase in the replacements corresponding to position j in cluster l caused by the simultaneous addition of a record k_1 to and removal of record k_2 from this cluster. Thus, the cost of arc (k_1,k_2) is given by $\sum_j inc_{jl}(k_1,k_2)$. Also, let $inc_{jl}^{+}(k)$ and $inc_{jl}^{-}(k)$ be the increase in the number of replacements corresponding to position j caused by the addition of a record k to cluster l $(k \notin \Phi_l)$ and the increase in the number of replacements corresponding to position j caused by the removal of a record k from cluster l $(k \in \Phi_l)$, respectively.

The idea of creating a database to significantly reduce the computations required in the calculation of a mode of a cluster of records has been used by Morgan [16] in the context of the 2-model problem. The database that we create consists of three items. The first item in the database is an $m \times n \times q$ matrix, $\mathcal{F}$, which stores the frequency of every value i at position j in all records belonging to cluster l, i.e., $\mathcal{F}[i,j,l] = F_l(i,j)$. The second item in the database is a $q \times n$ matrix $\mathcal{M}$, which stores the frequency of the most frequently occurring value at position j among all records belonging to cluster l, i.e., $\mathcal{M}[l,j] = F_l^{\max}(j)$. The third item in the database is a $q \times n$ boolean matrix $\mathcal{U}$. The matrix $\mathcal{U}$ indicates whether there is a unique value i^* that occurs most frequently at position j in all records belonging to cluster l, i.e., $\mathcal{U}[l,j] = 1$ if $\arg\max_i F_l(i,j) = i^*$ is unique and 0 otherwise.

Now, let us assume that we are calculating the cost of arc (k_1,k_2), i.e., the cost of adding record k_1 and removing record k_2 from cluster l, and let us further assume that at a particular position j, record k_1 has value i_1 and record k_2 has value i_2.

In order to describe our procedure for calculating $inc_{jl}(k_1, k_2)$, let us first describe the corresponding values of $inc^+_{jl}(k_1)$ and $inc^-_{jl}(k_2)$. It can be verified easily that $inc^+_{jl}(k_1)$ equals 0 when the value i_1 is the most frequently occurring value at position j among all records belonging to cluster l, i.e., $F_l(i_1, j) = F_l^{\max}(j)$, and $inc^+_{jl}(k_1)$ equals 1 otherwise. Further, $inc^-_{jl}(k_2)$ equals 0 only when the frequency of value i_2 is strictly greater than the frequency of any other value at position j among all records belonging to cluster l, i.e., when $F_l(i_2, j) = F_l^{\max}(j)$, and $\arg\max_i F_l(i, j)$ is unique. In all other cases $inc^-_{jl}(k_2)$ equals -1. These two facts, along with the database that we described earlier, can be used to efficiently calculate the cost of the arc (k_1, k_2) in the following manner. First we calculate the cost of adding record k_1 to cluster l, i.e., $\sum_j inc^+_{jl}(k_1)$. Next we update the database to reflect the addition of record k_1 to cluster l. Then we calculate the cost of removing k_2 from cluster l, i.e., $\sum_j inc^-_{jl}(k_2)$. To calculate $G(S)$ we need to do this for every pair of regular nodes where both nodes do not belong to the same cluster. An equivalent procedure can be used to calculate the cost of arcs from every regular node to $(q-1)$ pseudo nodes and the cost of arcs from the source node to every regular node. We refer to this implementation of the improvement graph calculation as IGC_1.

Further reduction in computation comes out of the fact that in a vast majority of positions, the increase in replacements corresponding to position j caused by the simultaneous addition of record k_1 to cluster l and removal of record k_2 from cluster l is equal to the sum of two values, the first one being the increase in replacements corresponding to position j caused by the lone addition of record k_1 to cluster l and the second one is the increase in replacements corresponding to position j caused by the lone removal of record k_2 from cluster l. In other words, for most positions j

$$inc_{jl}(k_1, k_2) = inc^+_{jl}(k_1) + inc^-_{jl}(k_2) \tag{6.1}$$

To clarify this statement we enumerate all possible cases corresponding to a particular position j in table 6.1 and identify the cases where equation (6.1) is not satisfied.

Table 6.1: Different cases for reducing the computations for calculating the Improvement Graph arc costs.

Major case	Minor case	k_1 in			k_2 out			Both		
		$\|l\|$	*max*	*inc*	$\|l\|$	*max*	*inc*	$\|l\|$	*max*	*inc*
	$F_l(i_1,j) = F_l(i_2,j) = max$	1	1	0	-1	0	-1	0	1	-1
$F_l(i_1,j)$	$F_l(i_1,j) = F_l(i_2,j) = max$-1	1	0	1	-1	0	-1	0	0	0
$= F_l(i_2,j)$	$F_l(i_1,j) = F_l(i_2,j) < max$ -1	1	0	1	-1	0	-1	0	0	0
	$\mathbf{F_l(i_2,j) = umax}$, $\mathbf{F_l(i_1,j) = umax}$**-1**	**1**	**0**	**1**	**-1**	**-1**	**0**	**0**	**0**	**0**
	$F_l(i_2,j) = umax$, $F_l(i_1,j) < umax$-1	1	0	1	-1	-1	0	0	1	1
$F_l(i_1,j)$	$F_l(i_2,j) = max$, $F_l(i_1,j) = max$-1	1	0	1	-1	0	-1	0	0	0
$< F_l(i_2,j)$	$F_l(i_2,j) = max, F_l(i_1,j) < max$-1	1	0	1	-1	0	-1	0	0	0
	$F_l(i_2,j) = max$-1, $F_l(i_1,j) < max$-1	1	0	1	-1	0	-1	0	0	0
$F_l(i_1,j)$	$F_l(i_1,j) = max, F_l(i_2,j) < max$	1	1	0	-1	0	-1	0	1	-1
$> F_l(i_2,j)$	$F_l(i_1,j) < max$	1	0	1	-1	0	-1	0	0	0
$i_1 = i_2$=i	$F_l(i,j) = umax$	1	1	0	-1	-1	0	0	0	0
	$\mathbf{F_l(i,j) = max}$	**1**	**1**	**0**	**-1**	**0**	**-1**	**0**	**0**	**0**

For convenience of discussion, in the table 6.1, we shall represent $F_l^{max}(j)$ as *max*. Further, when there is a unique value i for which this maximum value, i.e., *max*, is attained, we refer to it as *umax*.

In table 6.1 we consider a scenario where record k_1 is to be added to cluster l and record k_2 is to be removed from it. Further record k_1 has value i_1 and record k_2 has value i_2 in position j. Based on the values $F_l(i_1,j)$ and $F_l(i_2,j)$, there can be four possible cases which we refer to as the *major* cases. The first column of table 6.1 identifies the four major cases, $F_l(i_1,j) = F_l(i_2,j)$, i.e., records with values i_1 and i_2 are equal in number, $F_l(i_1,j) < F_l(i_2,j)$, i.e., value i_1 is less frequent than i_2, $F_l(i_1,j) > F_l(i_2,j)$, i.e., value i_1 is more frequent than i_2 and the fourth major case where the record k_1 coming in and the record k_2 leaving cluster l have the same value $i_1 = i_2 = i$ at position j. The second column identifies the *minor* cases within each major case. The different minor cases occur depending upon whether $F_l(i_1,j)$ and/or $F_l(i_2,j)$ is equal to *max*, *umax*, (*max*-1) or less than (*max*-1). The third column represents the impact of addition of record k_1 to cluster l, the fourth column represents the impact of removal of record k_2 from cluster l, and the fifth column represents the impact of both operations performed simultaneously. The third, fourth and fifth columns are further divided into three sub-columns. The first sub-column labeled as $|l|$ gives the increase in the number of records in cluster l. The second sub-column labeled as *max* gives the increase in the value of $\max_i F_l(i,j)$ corresponding to position j for cluster l and the third sub-column labeled as *inc* gives the increase in the number of replacements once the corresponding operation is completed. Thus the first sub-column that we encounter in the table that has title *inc* displays the value of $inc^+_{jl}(k_1)$, the second column with the same title *inc* displays the value of $inc^-_{jl}(k_2)$ and the third one displays the value of $inc_{jl}(k_1,k_2)$.

From table 6.1, we can see that there are exactly two cases when equation (6.1) is not satisfied. The corresponding rows are highlighted for easy recognition. The first case is where $F_l(i_2,j) = umax$ and $F_l(i_1,j) = (umax\text{-}1)$, i.e., the number of records in cluster l that have value i_2 at position j is strictly greater than records with any other value at this position and the number of records in cluster l that have value i_1 at position j is exactly one less than this number. In this case,

$$inc_{jl}(k_1,k_2) = inc^+_{jl}(k_1) + inc^-_{jl}(k_2) - 1$$

The second case is where $i_1 = i_2 = i$ and $F_l(i,j) = max$, i.e, the record k_1 coming in and the record k_2 leaving cluster l have the same value $i_1 = i_2 = i$ at position j and for cluster l there is another value $i' \neq i$ that has the same frequency as i. In this case,

$$inc_{jl}(k_1,k_2) = inc^+_{jl}(k_1) + inc^-_{jl}(k_2) + 1$$

Using this relationship, we can now calculate the improvement graph as follows. The cost of removing a record k_2 from cluster l, i.e, $\sum_j inc^-_{jl}(k_2)$ is calculated only once. Moreover, when this cost is calculated we create two lists, the first list (list 1) contains all positions where $F_l(i_2,j) = umax$ and the second one (list 2) contains all positions where $F_l(i_2, j) = max$. Now the cost of arc (k_1, k_2) can be calculated by adding $\sum_j inc^-_{jl}(k_2)$ and $\sum_j inc^+_{jl}(k_1)$ and then adding one to this sum for each position in list 1 where $F_l(i_1, j) = (umax - 1)$ and subtracting one for every position in list 2 where $i_1 = i_2$.

We shall refer to this implementation of the improvement graph calculation as implementation IGC_2. As compared to IGC_1, IGC_2 avoids the update of the database between the calculation of $\sum_j inc^-_{jl}(k_2)$ and $\sum_j inc^+_{jl}(k_1)$. Further, it calculates the cost of the removal of a record k_2 from its cluster l exactly once for every pair of records (k_1, k_2) where $k_2 \in \Phi_l$. Thus, in implementation IGC_2, we have reduced the number of times the record removal costs are calculated and also reduced the number of positions for which addition or subtraction operations are performed in the calculation of every arc cost. We empirically demonstrate that IGC_2 is faster than IGC_1 through a computational experiment. The results of this experiment are presented in table 7.2 which is included along with some discussion later in chapter 7.

6.5 Identification of Valid Cycles in the Improvement Graph.

From the definition of the augmented improvement graph $G(S)$ corresponding to a solution S, presented in subsection 6.3, we know that finding the best cyclic-exchange or path-exchange neighbor of a solution S is equivalent to finding a cycle in $G(S)$ of minimum cost such that no two nodes in this cycle are present in the same cluster. Let us refer to such a cycle in $G(S)$ as a *cluster disjoint cycle*. Further, we refer to a cluster disjoint cycle that has a negative cost as a *valid cycle*. Thus every cyclic-exchange or path-exchange performed on the solution S corresponds to a cluster disjoint cycle in $G(S)$, and every cyclic-exchange or path-exchange that leads to an improving neighbor corresponds to a valid cycle in $G(S)$. If the neighborhood search requires us to find a strictly improving neighbor in an iteration then we need to have an algorithm to search for valid cycles in $G(S)$. On the other hand, if we require the best neighbor of S regardless of whether it has a lower cost or not (such as in a tabu search algorithm), then we need an algorithm that searches for the minimum cost cluster disjoint cycle in $G(S)$.

The problem of finding a valid cycle in a graph has been proved to be an NP-complete problem [19]. Hence any exact algorithm developed for finding valid cycles or cluster disjoint cycles in $G(S)$ will have computational requirements that grow exponentially with the size of the problem. Presently, exact algorithms developed for finding both the minimum cost cluster disjoint cycle and valid cycles are based on algorithms for the all pair shortest path problem in graphs. These algorithms are described in detail in [1] and [6]. Ahuja et al. [4] describe a "label-setting" algorithm for the minimum cost subset disjoint cycle problem. We propose to implement this algorithm for solving the minimum cost cluster disjoint cycle problem (MCCDCP) in the context of the q-mode problem. We describe in brief our implementation of this algorithm below.

Every ordered pair of nodes (i, j) in the improvement graph has at least one label associated with it. A label L_{ij}, consists of three parts. The first part is a variable sized vector, $\rho(i, j)$, that stores all the nodes in the cluster disjoint path from source node i to terminal node j in the correct sequence. In places where the context is clear we refer to this path simply as ρ. Note that a cluster disjoint path from node i to node j corresponds to a cluster disjoint cycle in $G(S)$ only if there is an arc from node j to node i in $G(S)$. The size of a cluster disjoint path, $\rho(i, j)$, is given by the total number of nodes present in it. The second part of the label L_{ij} is the cost of the cluster disjoint path from i to j, $\rho(i, j)$, i.e., $c(\rho(i, j))$. If there is no path from node i to node j , then $c(\rho(i, j)) = \infty$. The third part of the label L_{ij} is a 0-1 vector, $w_{ij}(l)$ of size q that identifies the clusters that are present in the path, i.e., if cluster l is present in the path then $w_{ij}(l) = 1$, and 0 otherwise for all $l = 1$ to q. Moreover, a label L_{ij} is said to *dominate* label L'_{ij}, if and only if, $c(\rho(i, j)) \leq c'(\rho(i, j))$ and

Figure 6.3: Pseudo code for Algorithm AS

```
1   MinCost = $\infty$, Bestpath = $\Phi$
2   for each $s \in V$ do
3     for all $j > s$
4       if $(s, j) \in E$ then
5         $\mathcal{L}^1_{sj} = \{(\rho = (s, j), c(\rho) = c(s, j), w_\rho(cl(s)) = w_\rho(cl(j)) = 1)\}$
6       else $\mathcal{L}^1(i, j) = \phi$
7     endfor
8     for each $l = 2, \ldots, Q$ do
9       for each $(i, s) \in E$, $i > s$ and each $(\rho, c(\rho), w_\rho) \in \mathcal{L}^{l-1}_{si}$ do
10          if MinCost $> c(\rho) + c(i, s)$ then
11            MinCost $\Leftarrow c(\rho) + c(i, s)$ and Bestpath $\Leftarrow \rho$
12          endif
13        endfor
14        for each $j \in V$ and $j > s$ do
15          for each $(i, j) \in E$, $i > s$ & each $(\rho, c(\rho), w_\rho) \in \mathcal{L}^{l-1}_{si}$ with $w_\rho(i) \neq w_\rho(j)$ do
16            Extend $\rho$ along $(i, j)$ to get $\rho'$
17            $c(\rho') \Leftarrow c(\rho) + c(i, j)$, $w_{\rho'} \Leftarrow w_\rho$ and $w_{\rho'}(cl(j)) \Leftarrow 1$
18            if $(\rho', c(\rho'), w_{\rho'})$ is not dominated by any label in $\mathcal{L}^l_{sj}$ then
19              Remove all labels in $\mathcal{L}^l_{sj}$ dominated by $(\rho', c(\rho'), w_{\rho'})$
20              $\mathcal{L}^l_{sj} \Leftarrow \mathcal{L}^l_{sj} \cup (\rho', c(\rho'), w_{\rho'})$
21            endif
22          endfor
23        endfor
24      endfor
25    endfor
```

$w_{ij}(l) \leq w'_{ij}(l)$ for all $l = 1$ to q. A cluster disjoint path between nodes i and j is said to be *efficient* if no other path between these two nodes dominates it. It has been shown by Ahuja et al. [4] that we need to maintain labels only on pairs of nodes (i, j) such that $i < j$. Given these ideas, we can now describe the algorithm for calculating minimum cost cluster disjoint cycles that we shall refer to as AS.

The algorithm proceeds by selecting one regular node from the augmented improvement graph as the source node. From the source node s the algorithm finds efficient cluster disjoint paths of size 2 to every other node i in $G(S)$. This can be accomplished easily because every arc in $G(S)$ connecting node s to a node i that belongs to a different cluster corresponds to a cluster disjoint path of size 2. Now we move on to finding cluster disjoint paths of size 3. This is accomplished as follows. From every node i, we look at every neighbor j and check if the cluster disjoint path from s to i can be extended to node j, and if yes, then we create a label corresponding to node j. Thus we now have efficient cluster disjoint paths of size at most 3 nodes from source s to all other nodes. We continue to obtain cluster disjoint paths of larger sizes untill cluster disjoint paths of size at most q are known from s to all other nodes in the improvement graph. This procedure is performed p times, every time with a different regular node in $G(S)$ as the source node. Note that it is possible that there are two nodes i_1 and i_2 that have j as a neighbor, and all three nodes i_1, i_2 and j belong to different clusters. Thus it is possible that there are two cluster disjoint paths from s to j, one through i_1 and the other through i_2. Hence every pair of nodes can have more than one label associated with it. We refer to the set of labels corresponding to cluster disjoint paths of size l with end points (i, j) as $\mathcal{L}^l_{ij}$. All labels belonging to $\mathcal{L}^l_{ij}$ will be refered to as L^l_{ij}.

The algorithm is given in figure 6.3. In line 2 we select one node from the improvement graph as the source node s. In the loop from lines 3 to 7, we initialize the set of labels for each pair of nodes i, j where the edge $(i, j) \in \mathcal{E}$. In the loop, from lines 9 to 13 we update the value of the minimum cost cycle based on the information that is currently known, i.e., the sets of labels $\mathcal{L}^l_{si}$ which store all cluster disjoint paths of size at most l from source s to all other nodes i. This calculation can be done because addition of edge (i, s), if it exists, to a cluster disjoint path from s to i corresponds to a cycle. The loop from lines 14 to 23 creates new labels on all pairs of nodes s, j. The algorithm looks at each node i such that the edge (i, j) exists in the improvement graph and the node j is such that $j \in \mathcal{V}, j > s$. The cluster disjoint path from s to i of size $(l - 1)$ is extended to node j to obtain a cluster disjoint path of size l from s to j, and in lines 16 and 17 the label L^l_{sj} corresponding to this path is created. In line 18, we ensure that label created is not dominated by any label in the set $\mathcal{L}^l_{sj}$. We then add label L_{sj}^l to set $\mathcal{L}_{sj}^l$ and remove all labels in $\mathcal{L}_{sj}^l$ that are dominated by the label L_{sj}^l. Thus, only those labels that correspond to efficient cluster disjoint paths are maintained. This procedure finds the minimum cost cluster disjoint cycle that contains source node s. This procedure is run p times, every time with a different regular node in $G(S)$ as source, to obtain the minimum cost cluster disjoint cycle in the improvement graph $G(S)$.

6.6 Local Improvement Algorithm.

We have designed a local improvement algorithm for the q-mode problem based on the very large scale neighborhood structure that we have just discussed. We refer to this algorithm as LI. The major steps in LI are given in figure 6.4. In line 1, we generate an initial solution S^1 by randomly assigning every record to a unique cluster. We then calculate the cost $c(S^1)$ of solution S^1 by calculating the modes of each of the q clusters corresponding to solution S_1. This ends the initialisation section of the algorithm. Now, in every iteration, we construct the augmented improvement graph corresponding to the current solution S^k as shown in line 4. Next, we use algorithm AS to identify the minimum cost cluster disjoint cycle, ζ^k, in $G(S^k)$. If the cycle has non-negative cost, i.e., $c(\zeta^k) \geq 0$, we stop the algorithm with S^k as the best solution. On the other hand, if the cycle has strictly negative cost, i.e., $c(\zeta^k) < 0$, then we can use ζ^k to obtain solution S^{k+1} from S^k. Here, we also update the values in the database that we build to efficiently calculate the cost of the arcs in the improvement graph $G(S^k)$. Further, the cost of solution S^{k+1} can be calculated with just a single operation as shown in line 8. We then go back to step 4.

Figure 6.4: Pseudo code for Algorithm LI

```
1    Generate initial solution, $S^1$
2    Calculate $c(S^1)$
3    $k \Leftarrow 1$
4    Construct $G(S^k)$
5    Run algorithm **AS** on $G(S^k)$ to obtain $\zeta^k$
6    **if** $c(\zeta^k) < 0$ **then**
7        use $\zeta^k$ to construct $S^{k+1}$ and update database
8        $c(S^{k+1}) \Leftarrow c(S^k) + c(\zeta^k)$
9        $k \Leftarrow k + 1$
10         goto 4
11    **else** STOP, $S^k$ is optimal
```

6.7 Search Strategy

The local improvement algorithm *LI* that we discussed above uses the procedure *AS* to identify the least cost neighbor S' of the current solution S in every iteration. The algorithm proceeds as long as the least cost neighbor S' is improving, i.e., cost of S' is strictly less than cost of current solution S. We refer to this strategy of searching the neighborhood as *best improvement strategy*.

In a preliminary computational experiment we noted that the procedure *AS* accounts for a very high percentage of the computations performed by algorithm *LI*. In order to reduce the overall computational requirements of the algorithm we modify our search strategy. To this end we implemented two modifications of procedure *AS*. The first modification involves a change in the stopping criterion used to terminate procedure *AS*. Instead of identifying the least cost cluster disjoint cycle in the improvement graph $G(S)$, we terminate *AS* as soon as it identifies a negative cost cluster disjoint cycle in $G(S)$. We refer to this modified procedure as AS^1. It must be noted that if $G(S)$ has exactly one or no negative cost cluster disjoint cycle then the cycle identified by both procedures, *AS* and AS^1, at termination, will be identical. When procedure AS^1 is used, in every iteration of the local improvement algorithm we move to the first neighbor that we find that has lower cost than the current solution. Thus use of AS^1 in the local improvement algorithm implies a *first improvement* search strategy. We refer to this algorithm as LI^1.

The second modification of procedure AS involves limiting the number of nodes in the improvement graph $G(S)$ that we use as source nodes. We refer to this procedure as AS_α. In AS_α only a fraction of the regular nodes, $\lfloor \frac{\alpha p}{100} \rfloor$, will be used as source nodes as compared to procedure AS where each of the p regular nodes is used as the source node. Based on some preliminary computational experiments, we decided to use $\alpha = 10$ so as to get a significant reduction in the solution time for the local improvement algorithm. Unfortunately, this also lead to a significant reduction in the solution quality. To counter this effect, we modified the steps of the local improvement algorithm. Whenever AS_α is unable to find a negative cost cluster disjoint cycle in $G(S)$ we revert back to procedure AS. We refer to this version of the local improvement algorithm as LI_α. Algorithm LI_α differs from algorithm LI at two steps. Firstly at step 5, we use procedure AS_α in algorithm LI_α instead of procedure AS. Secondly, if $c(\zeta^k) > 0$ then we do not stop. Instead, we revert to procedure AS to find the minimum cost cluster disjoint cycle, $\zeta^{k'}$. Now if $c(\zeta^{k'}) > 0$ then we stop, otherwise we identify a new neighbor S^{k+1} using S^k and $\zeta^{k'}$, make S^{k+1} the current solution, and resume algorithm LI_α using procedure AS_α.

Chapter 7

Computational experiments for the Local Improvement Algorithm.

We carried out a computational experiment to empirically evaluate the effectiveness of the algorithms, LI, LI_α and LI^1, that we discussed earlier, and to compare them to each other. The computational experiment is carried out by running these algorithms on randomly generated problem instances that we created as described in chapter 4. We used an exact algorithm to optimally solve these randomly generated instances and we refer to the objective value obtained using the exact algorithm as V^*. In some instances, the exact algorithm is unable to find the optimal solution in a reasonable amount of time. For these instances V^* refers to the objective value of the best known integer feasible solution. Details regarding the exact algorithm are discussed in chapter 3. Further, we refer to the objective value obtained by algorithms LI, LI_α and LI^1 as V_{LI}, V_{LI_α} and V_{LI^1} respectively. Each of the three algorithms start with the same initial solution for any given instance. This solution is obtained by a random assignment of every record to a unique cluster and the objective value of this solution is referred to as V_{ini}. We also discuss a second computational experiment conducted to compare the two implementations IGC_1 and IGC_2 for calculating the arc costs in an improvement graph.

In this chapter the first section describes the performance measures used in the analysis of the computational experiment and in the second section we present the results of the two computational experiments that we performed.

7.1 Performance Measures

For each algorithm B, where B refers to either LI, or LI^1 or LI_α, we define the following two performance measures for each problem instance that we solve.

$$\delta_B^{ini} = \frac{V_{ini} - V_B}{V_{ini}}, \quad \text{and} \quad \delta_B^* = \frac{V_B - V^*}{V^*} \tag{7.1}$$

where V_B is the objective value obtained via algorithm B, V^* is the a priori best known objective value, and V_{ini} is the objective value of the initial solution.

Further, the time required in seconds by each algorithm is used as the third performance measure, and it is referred to as t_B, where B can take values LI, LI^1 and LI_α, respectively.

7.2 Observations

In our first computational experiment we ran all the three algorithms on the 55 randomly generated instances that we created. The results of this computational experiment are presented in table 7.1. Each row in the table corresponds to one of the instances. The information in table 7.1 is displayed in such a manner that all instances belonging to the same group (by pmf type) are together. Further, within each group the instances are arranged in the increasing order of the size of the instance. The first column in the table displays the name of the instance while the second and third columns contain the corresponding values V^* and V_{ini}, respectively. All problem instances where the corresponding value V^* is not guaranteed to be optimal are marked with the symbol $^\bullet$. After the third column, table 7.1 contains 3 groups of columns each corresponding to one of the performance measures that we described earlier. Each group contains three columns corresponding to the three algorithms that we are comparing. Finally, the last column contains the best objective value (V_{fb}) that we have for each instance at the end of the computational experiment. In all instances where the optimal is known, we have $V_{fb} = V^*$. In some instances where the optimal is not known, one or more of the algorithms finds a solution which is better than the best solution that we previously knew for that instance. In these cases, $V_{fb} < V^*$, and these cases are displayed using bold characters.

We know that the groups G_p, G_s and G_g contain strongly clustered instances as compared to the other two groups. In the following discussion, we shall refer to instances in groups G_p, G_s and G_g as strongly clustered and instances in groups G_h and G_l as weakly clustered. We know the optimal value for 38 out of the 55 instances that are displayed in table 7.1 (29 out of 33 strongly clustered instances and 9 out of 22 weakly clustered instances).

Initially let us concentrate on the 38 instances where we can guarantee that V^* is the optimal value. The three algorithms, LI, LI_α and LI^1 obtain the optimal value in all 29 strongly clustered instances where V^* is known to be optimal. Among the 9 weakly clustered instances, LI and LI^1 obtain the optimal value in the same 5 instances. LI_α obtains the optimal value in 6 instances that include the 5 instances where LI and LI^1 obtain the optimal value. In the remaining three instances none of the algorithms obtain the optimal value, these instances are l-8-10-10-2-1, l-8-10-10-3-1 and h-8-10-10-3-1. Overall, within the 38 instances, the search algorithms obtain the optimal value in 35 instances and in the remaining three instances, the worst solution found by any of the three algorithms is within 8% of the corresponding optimal value.

Next we discuss the performance of the three algorithms on the 17 instances where we cannot guarantee that V^* is the optimal value. Of these 17 instances, 4 are strongly clustered and 13 are weakly clustered. Among the strongly clustered instances, LI

and LI^1 find solutions with objective values equal to V^* in 2 out of the 4 instances. In the third instance, LI obtains value equal to V^* while the value obtained by LI^1 is 7% above V^*. In the fourth instance, both LI and LI^1 obtain values that are 18% higher than the optimal value. Algorithm LI_α obtains values equal to V^* in all four instances. Among the weakly clustered instances, LI and LI_α find solutions with objective values that are equal to V^* in 5 instances. In 6 instances, LI and LI_α find solutions with objective values that are less than V^* but not always equal to each other. In the remaining two instances the values obtained by LI and LI_α are equal to each other and they are 4% and 12% higher than V^*. Algorithm LI^1 finds solutions with objective values that are equal to or lesser than V^* in 12 out of the 13 instances. In the last instance it finds the same value as found by LI and LI_α which is 12% higher than V^*.

On the average, for all the 33 strongly clustered instances that are displayed in table 7.1, LI obtains solutions with objective values that are 0.54% higher than V^* and LI^1 finds solutions that are 0.75% above the corresponding V^*. On the other hand, objective values obtained by LI_α for all 33 strongly clustered instances are equal to V^*. On the average, for the 22 weakly clustered instances, LI obtains solutions with objective values that are less than V^* by 0.03% , LI^1 obtains values that are less than V^* by 0.12% and LI_α obtains values that are less than V^* by 0.22%. In summary, we can conclude that in terms of solution quality LI_α marginally outperforms the two other algorithms LI and LI^1.

Since all three algorithms start from the same initial solution, the performance measure, δ^{ini}, indicates the same trend as shown by δ^*. On the average, over all the 55 instances that are part of the computational experiment, LI finds solutions that are 29.8% better than the corresponding initial solution V_{ini}, LI^1 finds solutions that are 29.74% better than the corresponding initial solution V_{ini} and LI_α finds solutions that that are 30.01% better than the corresponding initial solution V_{ini}. In general we observe that this performance measure is larger for the instances with stronger natural clusters. This is consistant with the intuitive argument that for a group of records that does not contain strong clusters any randomly generated assignment of records to clusters can lead to a solution which is not far away from the optimal assignment.

In terms of the time required, LI^1 considerably outperforms the other two algorithms. In all the 55 instances LI^1 is faster than LI_α and on the average the time required by LI^1 is 22.14% of the time required by LI_α. For instances that have 100 records and require 8 clusters the time required by LI is greater than 3600 seconds (the upper limit that we used in this experiment) and the exact time value is not available. Thus on the average for the 50 instances where the time values are available LI_α takes 51.9% of the time taken by LI.

Our second computational experiment was performed to compare the two implementations IGC_1 and IGC_2, that calculate the cost of all the arcs in an improvement graph. In this experiment we ran the algorithm LI_α on 45 problem instances and the cost of all arcs in the improvement graph at every iteration was calculated using both implementations. For each problem instance we kept track of the total time required by both implementations to calculate the improvement graph arc costs in all iterations that algorithm LI_α performed for that problem instance. The results are presented in table 7.2. For the analysis of this computational experiment we group together the problem instances that we have created based on parameter p, i.e., the number of records in the instance and parameter q, i.e., the total number of clusters to be formed out of the p records. Thus every instance belonging to one group has the same value of p and q. There are total of 9 groups in this experiment and each group has 5 instances. Every row in the table 7.2 displays the average time over all the instances belonging to that group. The first column in the table represents the problem size and displays the parameters m, n, p and q. The second column gives the number of instances belonging to that group (5 in every case). The third column, labeled Time, has two sub-columns which give the average time required by each implementation for all instances in that group in seconds.

The average time required by IGC_2 is significantly less than the time required by IGC_1 in all the groups of instances shown in table 7.2. Further, as the size of the instance increases the difference in the time required by IGC_1 and IGC_2 increases. This supports our earlier discussions that the additional measures we introduced in IGC_2 indeed result in significant reduction in computational effort in determining the arc costs in the improvement graph.

Using both tables 7.1 and 7.2 we also observe that the time taken by IGC_2 to calculate the arc costs is relatively insignificant as compared to the total solution time. Procedure AS, that finds the cluster disjoint cycle in the improvement graph requires the bulk of the overall computational effort of the algorithm. Hence further improvements in the overall efficiency of the algorithm depends on determining a more efficient algorithm for finding a cluster disjoint minimum cost cycle in the improvement graph as opposed to more efficient determination the arc costs.

Table 7.1: Computational results for LI, LI^1 and LI_α.

type-m-n-p-q-1	V^*	V_{ini}	δ^*_{LI}	$\delta^*_{LI^1}$	$\delta^*_{LI_\alpha}$	δ^{ini}_{LI}	$\delta^{ini}_{LI^1}$	$\delta^{ini}_{LI_\alpha}$	t_{LI}	t_{LI^1}	t_{LI_α}	V_{fb}
l-8-10-10-2-1	55	59	0.0182	0.0182	0.0182	0.05	0.05	0.05	0.01	0.00	0.00	55
l-8-10-10-3-1	45	53	0.022	0.022	0.022	0.13	0.13	0.13	0.00	0.01	0.01	45
l-10-20-20-2-1	273	291	0.022	0.022	0.0	0.04	0.04	0.06	0.01	0.01	0.01	273
l-10-20-20-3-1	251$^\bullet$	277	0.04	-0.04	0.012	0.09	0.10	0.08	0.04	0.03	0.02	**250**
l-20-20-50-2-1	820$^\bullet$	850	0.122	0.122	0.122	0.02	0.02	0.02	0.10	0.15	0.11	830
l-20-20-50-3-1	790$^\bullet$	825	-0.0063	-0.0114	-0.0190	0.05	0.05	0.06	0.57	0.29	0.67	**775**
l-20-20-50-5-1	757$^\bullet$	785	-0.0343	-0.0304	-0.041	0.07	0.06	0.08	7.22	1.45	6.31	**726**
l-20-20-100-2-1	1710$^\bullet$	1749	-0.0018	-0.0047	-0.0023	0.02	0.03	0.02	1.03	1.10	0.76	**1702**
l-20-20-100-3-1	1670$^\bullet$	1734	-0.015	-0.0114	-0.0102	0.05	0.05	0.05	7.27	1.34	2.36	**1645**
l-20-20-100-5-1	1651$^\bullet$	1672	-0.046	-0.097	-0.0509	0.06	0.06	0.06	81.20	10.76	85.70	**1567**
l-20-20-100-8-1	1584$^\bullet$	1613	-0.06	-0.07	-0.07	0.08	0.09	0.09	-	177.16	1489.66	**1472**
h-8-10-10-2-1	41	59	0.00	0.00	0.00	0.31	0.31	0.31	0.00	0.00	0.00	41
h-8-10-10-3-1	37	47	0.08	0.08	0.08	0.15	0.15	0.15	0.01	0.01	0.01	37
h-10-20-20-2-1	172	243	0.00	0.00	0.00	0.29	0.29	0.29	0.02	0.02	0.01	172
h-10-20-20-3-1	195	253	0.00	0.00	0.00	0.23	0.23	0.23	0.04	0.04	0.04	195
h-20-20-50-2-1	494	690	0.00	0.00	0.00	0.28	0.28	0.28	0.18	0.25	0.18	494
h-20-20-50-3-1	491$^\bullet$	710	0.00	0.00	0.00	0.31	0.31	0.31	0.77	0.48	0.33	491
h-20-20-50-5-1	486$^\bullet$	737	0.00	0.00	0.00	0.34	0.34	0.34	9.68	2.12	4.08	486
h-20-20-100-2-1	968	1384	0.00	0.00	0.00	0.30	0.30	0.30	1.72	1.74	1.23	968
h-20-20-100-3-1	984$^\bullet$	1502	0.00	0.00	0.00	0.34	0.34	0.34	8.85	3.29	5.46	984
h-20-20-100-5-1	945$^\bullet$	1566	0.00	0.00	0.00	0.40	0.40	0.40	143.13	22.79	74.86	945
h-20-20-100-8-1	943$^\bullet$	1577	0.00	0.00	0.00	0.40	0.40	0.40	-	334.09	1747.57	943
g-8-10-10-2-1	40	56	0.00	0.00	0.00	0.29	0.29	0.29	0.01	0.01	0.00	40
g-8-10-10-3-1	26	53	0.00	0.00	0.00	0.51	0.51	0.51	0.01	0.01	0.00	26
g-10-20-20-2-1	127	242	0.00	0.00	0.00	0.48	0.48	0.48	0.01	0.01	0.02	127
g-10-20-20-3-1	121	253	0.00	0.00	0.00	0.52	0.52	0.52	0.04	0.02	0.03	121
g-20-20-50-2-1	494	690	0.00	0.00	0.00	0.28	0.28	0.28	0.16	0.23	0.16	494
g-20-20-50-3-1	475	755	0.00	0.00	0.00	0.37	0.37	0.37	0.78	0.30	0.31	475
g-20-20-50-5-1	464	765	0.00	0.00	0.00	0.39	0.39	0.39	11.56	1.26	3.73	464
g-20-20-100-2-1	984	1440	0.00	0.00	0.00	0.32	0.32	0.32	1.78	1.91	1.31	984
g-20-20-100-3-1	991	1541	0.00	0.00	0.00	0.36	0.36	0.36	8.55	2.55	3.33	991
g-20-20-100-5-1	973$^\bullet$	1613	0.00	0.00	0.00	0.40	0.40	0.40	150.44	38.14	79.12	973
g-20-20-100-8-1	964$^\bullet$	1566	0.00	0.00	0.00	0.38	0.38	0.38	-	366.62	1124.11	964
s-8-10-10-2-1	50	68	0.00	0.00	0.00	0.26	0.26	0.26	0.01	0.00	0.01	50
s-8-10-10-3-1	28	54	0.00	0.00	0.00	0.48	0.48	0.48	0.01	0.01	0.01	28
s-10-20-20-2-1	251	298	0.00	0.00	0.00	0.16	0.16	0.16	0.01	0.02	0.01	251
s-10-20-20-3-1	193	288	0.00	0.00	0.00	0.33	0.33	0.33	0.04	0.04	0.04	193
s-20-20-50-2-1	641	787	0.00	0.00	0.00	0.19	0.19	0.19	0.19	0.18	0.15	641
s-20-20-50-3-1	622	797	0.00	0.00	0.00	0.22	0.22	0.22	0.69	0.44	0.36	622
s-20-20-50-5-1	578	799	0.00	0.00	0.00	0.28	0.28	0.28	13.32	2.57	5.15	578
s-20-20-100-2-1	1344	1620	0.00	0.00	0.00	0.17	0.17	0.17	1.84	1.03	1.10	1344
s-20-20-100-3-1	1310	1642	0.00	0.00	0.00	0.20	0.20	0.20	8.48	3.31	3.44	1310
s-20-20-100-5-1	1260	1697	0.00	0.00	0.00	0.26	0.26	0.26	153.67	20.92	59.35	1260
s-20-20-100-8-1	1053$^\bullet$	1574	0.00	0.07	0.00	0.33	0.28	0.33	-	235.99	1585.92	1053
p-8-10-10-2-1	31	58	0.00	0.00	0.00	0.47	0.47	0.47	0.01	0.00	0.00	31
p-8-10-10-3-1	28	53	0.00	0.00	0.00	0.47	0.47	0.47	0.01	0.00	0.01	28
p-10-20-20-2-1	136	182	0.00	0.00	0.00	0.25	0.25	0.25	0.01	0.02	0.01	136
p-10-20-20-3-1	125	244	0.00	0.00	0.00	0.49	0.49	0.49	0.04	0.03	0.03	125
p-20-20-50-2-1	345	635	0.00	0.00	0.00	0.46	0.46	0.46	0.19	0.11	0.15	345
p-20-20-50-3-1	341	715	0.00	0.00	0.00	0.52	0.52	0.52	0.73	0.34	0.40	341
p-20-20-50-5-1	327	734	0.00	0.00	0.00	0.55	0.55	0.55	8.89	3.55	5.58	327
p-20-20-100-2-1	730	1309	0.00	0.00	0.00	0.44	0.44	0.44	1.82	2.29	1.38	730
p-20-20-100-3-1	732	1449	0.00	0.00	0.00	0.49	0.49	0.49	8.66	2.84	3.97	732
p-20-20-100-5-1	713	1587	0.00	0.00	0.00	0.55	0.55	0.55	148.87	31.71	55.22	713
p-20-20-100-8-1	690$^\bullet$	1608	0.18	0.18	0.00	0.49	0.49	0.57	-	430.96	1344.88	690

Table 7.2: Comparison of implementations IGC_1 and IGC_2 for calculation of the improvement graph costs.

Problem size				Num	Time	
m	n	p	q		IGC_1	IGC_2
8	10	10	2	5	0.004	0.002
8	10	10	3	5	0.008	0.002
10	20	20	2	5	0.06	0.004
10	20	20	3	5	0.072	0.004
20	20	50	3	5	1.56	0.044
20	20	50	5	5	1.868	0.066
20	20	100	2	5	8.818	0.168
20	20	100	3	5	10.646	0.22
20	20	100	5	5	12.93	0.306

Chapter 8

Conclusion and Future Work.

8.1 Summary of Results

In this work we design algorithms for the q-mode problem using two different approaches. The first approach is based on MIP formulations for the q-mode problem which leads to exact algorithms, i.e., algorithms that obtain an optimal solution to the q-mode problem. The second approach includes the design of a very large scale neighborhood for the q-mode problem which can be used to develop a heuristic search algorithm for this problem.

As part of the first approach, we first demonstrate that the binary constraints on v_{ijl} variables in the formulation (IP) can be relaxed to non-negativity restrictions. We show that the resulting formulation (IPR) can be used to obtain an optimal solution for the q-mode problem. Next we propose a Benders decomposition for the formulation (IPR) which can be used to develop an exact algorithm for the q-mode problem. Further we propose an alternate MIP formulation and also modify the formulation (IPR) such that the corresponding LP relaxation of the resulting formulation (IPT) has a potentially larger optimal value.

As part of the second approach, we design a very large scale neighborhood for the q-mode problem using the concept of *cyclic-exchange* moves. Further, we use this neighborhood to design three closely related local improvement algorithms for the q-mode problem.

A computational experiment that we conduct reveals that the LP relaxation of formulation (IPT) has a higher optimal objective value as compared to the LP relaxations of (IPR) and (IPA). Based on this observation we select formulation (IPT) to optimally solve instances of the q-mode problem using the CPLEX branch and cut algorithm. The ability to solve large problem instances using MIP formulations of the q-mode problem via CPLEX is directly related to the quality of the lower bound obtained from the LP relaxation of the model. The computational experiment that we perform reveals that the quality of the LP relaxation of the formulation (IPT) depends on the strength of the clusters present in the problem instance. The optimal value of the LP relaxation has a high value in the case of instances with strong clusters while the optimal value of the LP relaxation is relatively low for instances with weak clusters. Using formulation (IPT), the CPLEX branch and cut algorithm is able to optimally solve reasonably large size problem instances that have strong natural clusters. On the other hand, the CPLEX branch and cut algorithm is unable to obtain the optimal solution in smaller problem instances that have weak clusters.

The Benders decomposition for the formulation (IPR) presents an alternative strategy that can be used to solve the q-mode problem optimally. The computational experiments reveal that this strategy can be used to obtain an optimal solution for small size problems. However the algorithms based on Benders decomposition, either B_{opt} or B_{Nopt}, are ineffective in their present form to solve medium or large size problem instances. This is primarily due to the fact that we presently solve the relaxed master problem at each iteration as a general integer programming problem. We employ the CPLEX branch and cut algorithm for this problem and it is simply too time consuming. A more computationally effective approach for solving the the relaxed master problem can lead to significant improvement in the algorithms based on Benders decomposition.

Our computational experiments with the local improvement algorithm that we propose demonstrate that it can obtain very good quality local optima. The local improvement algorithm finds optimal or near optimal solutions for problem instances which have strong clusters. For problem instances with weak clusters, on the average, the algorithm improves on the best known solutions using significantly less computer time than the time for which the exact algorithm was run. Since this algorithm takes less computer time it can be used to solve considerably larger instances, though we cannot comment on the solution quality for these large instances at this moment.

8.2 Future Research.

In the context of the work presented in this dissertation, we propose the following subjects for further research.

1. MIP models for the q-mode problem.

   To increase the size of the problem instances for which optimal solutions can be obtained we have to investigate methods by which the size of the branch and cut tree built by CPLEX can be reduced. This can be achieved by strengthening the formulation (IPT) by the addition of valid inequalities so that the optimal value of the LP relaxation is further improved.

2. Algorithms based on Benders decomposition.

   (a) To increase the effectiveness of the algorithms based on Benders decomposition we need to investigate alternative methods for the solution of the relaxed master problem at every iteration. The special structure of the relaxed master problem and its close relationship with the clustering concepts discussed throughout this thesis might be employed to devise a more effective algorithm for solving it. This approach is of course directed at the possiblity of reducing the computational requirements at each iteration.

(b) Another approach is to reduce the total number of iterations required by the algorithms. The Benders master problem needs to be investigated to be able to efficiently identify a subset of the constraints of the master problem (BM). This subset should be such that when we optimally solve the corresponding relaxed master problem (BM_r) (consisting of only these constraints) then we obtain a lower bound which equals the optimal value.

3. Efficient algorithms for the minimum cost cluster disjoint cycle problem in graphs.

   A significant portion of the computations in the local improvement algorithm is accounted for by the procedure that finds cluster disjoint cycles in the improvement graphs. Thus the effectiveness of the local improvement algorithm can be improved by employing a more efficient heuristic for finding negative cost cluster disjoint cycles in graphs instead of using the procedure AS or its variations.

References

[1] Ahuja, R., K., Magnanti, T., L., Orlin, J., B., "Network Flows: Theory Algorithms and Applications.", Prentice Hall, NJ, USA, 1993.

[2] Ahuja, R., K., Orlin, J., B., Sharma, D., "Very Large-Scale Neighborhood Search", *Intl. Trans in Oper. Res.*, Vol. 7, pp 301-317, 2000.

[3] Ahuja, R., K., Orlin, J., B., Sharma, D., "Multi-exchange Neighborhood Structures for the Capacitated Minimum Spanning Problem.", *Math. Prog.*, pp 71-97, 2001.

[4] Ahuja, R. K., Boland, N., Dumitrescu, I., "Exact and heuristic algorithms for the subset disjoint minimum cost cycle problem", http://www.ise.ufl.edu/ahuja/vlsn/Papers/Pap-ABD.pdf, 2001.

[5] Ahuja, R., K., Ergun, O., Orlin J., B., Punnen, A., P., "A survey of very large-scale neighborhood search techniques", *Disc. App. Math.*, 123 pp 75-102, 2002.

[6] Desrosiers, J., Dumas, Y., Solomon, M. M., Soumis, F., "Time Constrained Routing and Scheduling", Network Routing, HandBooks in Operations Research and Management Science 8, North Holland, Amsterdam, pp 35-139, 1995.

[7] Ganti, V., Gehrke, J., Ramakrishnan, R., "CACTUS-Clustering Categorical Data Using Summaries", Proceedings of the fifth ACM SIGKDD international conference on knowledge discovery and data mining, San Diego, California, pp 73-83, 1999.

[8] Glover, F., "Ejection chains, Reference structures, and alternating path algorithms for the traveling salesman problem", *Disc. App. Math.*, 65, pp 223-253, 1996.

[9] Guha, S., Rastogi, R., Shim, K., "ROCK: A Robust Clustering Algorithm for Categorical Attributes", *Info. Sys.*, Vol 25, pp 345-366, 2000.

[10] Han, J., Kamber, M., "Data Mining: Concepts and Techniques", Morgan Kaufmann, San Francisco, USA, 2001.

[11] Huang, Z., "A Fast Clustering Algorithm to Cluster Very Large Categorical Data Sets in Data Mining.", SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (SIGMOD-DMKD'97), Tucson, Arizona, May 1997.

[12] Jain, A., K., Murty, M., N., Flynn, P., J., "Data Clustering: A Review ", ACM Computing Surveys, Vol 31(3), 264-323, 1999.

[13] Kernighan, B., W., Lin, S., "An Efficient Heuristic Procedure for Partitioning Graphs", *The Bell Systems Technical Journal*, Vol 49(2), 291-307, 1970.

[14] Lin, S., Kernighan, B., W., "An Efficient Heuristic Algorithm for the Travelling Salesman Problem", *Operations Research*, Vol 21, 498-516, 1970.

[15] MacQueen, J., B., "Some Methods of Classification and Analysis of Multivariate Observations", Proceedings of the 5^{th} Berkeley Symposium on Mathematical Statistics and Probability, pp 281-297, 1967.

[16] Morgan, S., "Cluster Analysis in Electronic Manufacturing", PhD Dissertation, Operations Research Program, NC State University, Raleigh, NC, 2001.

[17] Morgan, S., D., Fathi, Y., Taheri, J., "Algorithms for the Model Configuration Problem.", *IIE Transactions*, Vol 36, pp 169-180, 2004.

[18] Parker, R. G., Rardin, R., L., "Discrete Optimization" Academic Press Inc, 1988.

[19] Thomson, P. M., Orlin, J., B., “The Theory of Cyclic Transfers”, Working paper No. OR 200-89, Operations Research Center, MIT, Cambridge, MA, 1989.

Appendix A

Lower Bounds for the q-mode Problem.

In this appendix, we present a lower bound for the q-mode problem which we shall refer to as LB_F. This lower bound was first introduced by Morgan et al. [16] in the context of the 2-model problem. In the first section we introduce some notation and demonstrate the validity of the lower bound. We also theoretically compare LB_F with the lower bound obtained from the LP relaxation of the IP model (IPT). In particular, we show that $v(LPT) \leq LB_F$. In the next section we discuss a class of q-mode problems for which $v(LPT)$ is 0. Finally, after section 2, we present the results of a computational experiment carried out to compare LB_F and $v(LPT)$ on an empirical basis. We use the randomly generated problem instances described in chapter 4 for this experiment. We observe that for each of the 50 problem instances in this experiment $v(LPT) = LB_F$.

A.1 An Alternate Lower Bound for the q-mode Problem.

As defined before, let $F_l(i,j)$ be the number of records in the cluster l that have value i in position j, and let $\max_i F_l(i,j) = F_l^{max}(j)$. Similarly, $F(i,j)$ denotes the number of records in the entire collection Φ that have value i in position j and $\max_i F(i,j) = F^{max}(j)$. Thus $F_l(i,j)$ is the frequency of occurance of value i at position j in all records assigned to cluster l and $F(i,j)$ is the frequency of occurance of value i at position j among all records in the collection. Further, we denote the frequency of the e^{th} most frequent value at position j among all records in the entire collection by $F^{max_e}(j)$, for all $e = 1$ to m, i.e., the frequency of the most frequently occurring value in Φ is denoted by $F^{max_1}(j)$, the frequency of the second most frequently occurring value in Φ is denoted by $F^{max_2}(j)$, and so on. In the event that two values have the same frequency we break the tie arbitrarily. It follows from the definition of $F^{max_e}(j)$ that,

$$\sum_{e=1}^{m} F^{max_e}(j) = \sum_{i=1}^{m} F(i,j) = p \tag{A.1}$$

A solution to the q-mode problem is a partitioning of the given collection of records, Φ into q clusters $\Phi_1, \ldots, \Phi_q$. Assuming $q \leq m$ it follows that, the sum of the frequencies of the most frequently occurring values at position j in each of the q clusters has to be less than or equal to the sum of the q largest frequencies of values at position j in the entire colection Φ, i.e.,

$$\sum_{l} F_l^{max}(j) \leq \sum_{e=1}^{q} F^{max_e}(j) \tag{A.2}$$

Let us assume that for the given collection of records Φ, $y^* \in Y$ is an optimal assignment of records to clusters, i.e., $\nu^* = \min_{y \in Y} v(IPR_y) = v(IPR_{y^*})$. Here ν^* is the optimal objective value, while (IPR_y) is the IP model obtained by fixing the values of the y_{kl} variables in model (IPR). Let the cluster assignment given by y^* partition the given collection of records into q clusters, $\Phi_1^*, \ldots, \Phi_q^*$, i.e., $\Phi_1^* \cup \ldots \cup \Phi_q^* = \Phi$. Further, let $|\Phi_l^*|$ represent the number of records present in cluster l. By definition, the value at position j in the mode vector corresponding to cluster l is $\arg\max_i F_l(i,j)$ and the associated number of replacements corresponding to the position j of cluster l equals $|\Phi_l^*| - F_l^{max}(j)$. Hence, the optimal objective value, $\nu^* = v(IPR_{y^*})$, is given by $\sum_j \sum_l (|\Phi_l^*| - F_l^{max}(j)) = np - \sum_j \sum_l F_l^{max}(j)$.

From equation A.2 we can see that

$$\nu^* = np - \sum_j \sum_l F_l^{max}(j) \geq np - \sum_j \sum_{e=1}^{q} F^{max_e}(j) \tag{A.3}$$

i.e., $LB_F = np - \sum_j \sum_{e=1}^{q} F^{max_e}(j)$ is a lower bound for ν^*. We now compare this lower bound with the lower bound $v(LPT)$ that we discussed earlier.

Lemma A.1. *The optimal value of the LP relaxation of formulation (IPT) is no greater than the lower bound LB_F, i.e., $v(LPT) \leq LB_F$.*

Proof : We prove this statement assuming that $q < m$. If $q \geq m$ this lemma follows trivially from our discussion in Appendix A.2. The linear programming relaxation of the formulation (LPT) is given as

$$(LPT) \left(\begin{array}{cc} \min \sum_j \sum_k \sum_l t_{jkl} & \\ \text{s.t } \; a_{ijk} y_{kl} - v_{ijl} \leq t_{jkl} & \forall \; i,j,k,l \\ \sum_i v_{ij1} = 1 & \forall \; j,l \\ \sum_l y_{kl} = 1 & \forall \; k \\ t_{jkl}, v_{ijl}, y_{kl} \geq 0 & \forall \; i,k \end{array} \right)$$

If we fix each of the y_{kl} variables to be equal to $\frac{1}{q}$, i.e, $y_{kl} = \frac{1}{q} \; \forall \; k,l$ then we get a linear programming model that we refer to as $(LPT^{\bullet})$. Since $(LPT^{\bullet})$ is obtained by fixing the values of the y_{kl} variables in model (LPT), we have

$$v(LPT) \leq v(LPT^{\bullet}) \tag{A.4}$$

We know that for each value of j and k we have $a_{ijk} = 1$ for exactly one value i and $a_{ijk} = 0$ otherwise. Let this value assigned to position j in record k be denoted by i_{jk}. Thus, $a_{i_{jk}jk} = 1$ and $a_{ijk} = 0 \ \forall \ i \neq i_{jk}$. Further the constraint $\frac{a_{ijk}}{q} - v_{ijl} \leq t_{jkl}$ is redundant when $a_{ijk} = 0$. Thus we can rewrite $LPT^{\bullet}$ as

$$(LPT^{\bullet}) \qquad \left(\begin{array}{cc} \min \sum_j \sum_k \sum_l t_{jkl} & \\ \text{s.t } \frac{1}{q} - v_{i_{jk}jl} \leq t_{jkl} & \forall j,k,l \\ \sum_i v_{ijl} = 1 & \forall \ j,l \\ t_{jkl}, v_{ijl} \geq 0 & \forall \ i,j,k,l \end{array} \right)$$

This problem is separable in index variable j and we obtain a subproblem $(LPT^{\bullet}_j)$ for each value of j.

$$(LPT^{\bullet}_j) \qquad \left(\begin{array}{cc} \min \sum_k \sum_l t_{jkl} & \\ \text{s.t } \frac{1}{q} - v_{i_{jk}jl} \leq t_{jkl} & \forall k,l \\ \sum_i v_{ij1} = 1 & l \\ t_{jkl}, v_{ijl} \geq 0 & \forall \ i,k,l \end{array} \right)$$

Every subproblem $(LPT^{\bullet}_j)$ is further separable in index variable l and hence the problem $(LPT^{\bullet})$ is separable into nq subproblems, one for each value of $j = 1$ to n and $l = 1$ to q. Each subproblem denoted by $(LPT^{\bullet}_{jl})$ can be written as

$$(LPT^{\bullet}_{jl}) \qquad \left(\begin{array}{ll} \quad \min \sum_k t_{jkl} & \\ \text{s.t } \frac{1}{q} - v_{i_{jk}jl} \leq t_{jkl} & \forall k \\ \quad \sum_i v_{ijl} = 1 & \\ \quad t_{jkl}, v_{ijl} \geq 0 & \forall \ i, k \end{array} \right)$$

Thus in its fully separated form $(LPT^{\bullet})$ can be written as

$$(LPT^{\bullet}) \qquad \sum_j \sum_l \left(\begin{array}{ll} \quad \min \sum_k t_{jkl} & \\ \text{s.t } \frac{1}{q} - v_{i_{jk}jl} \leq t_{jkl} & \forall \ k \\ \quad \sum_i v_{ij1} = 1 & \\ \quad t_{jkl}, v_{ijl} \geq 0 & \forall \ i, k \end{array} \right)$$

and, $v(LPT^{\bullet}) = \sum_j \sum_l v(LPT^{\bullet}_{jl})$.

We can show that all the "$\leq$" constraints in the subproblem $(LPT^{\bullet}_{jl})$ can be replaced with "$=$" constraints without affecting the optimal solution. Let $\{t^*_{jkl}, v^*_{ijl}\}$ be an optimal solution for problem $(LPT^{\bullet}_{jl})$. Also, for some k', suppose $t^*_{jk'l} > \frac{1}{q} - v^*_{i_{jk'}jl}$. Let $t'_{jk'l} = \frac{1}{q} - v^*_{i_{jk'}jl}$. It follows that

$$\sum_k t^*_{jkl} > \sum_{k \neq k'} t^*_{jkl} + t'_{jk'l}$$

It is also true that $t'_{jk'l} \geq 0$. Hence, $\{t^*_{jk'l}, v^*_{ijl}\}$ is not an optimal solution to $(LPT^{\bullet}_{jl})$. Hence we can write $(LPT^{\bullet}_{jl})$ as follows

$$(LPT^{\bullet}_{jl}) \qquad \left(\begin{array}{ll} \quad \min \sum_k t_{jkl} & \\ \text{s.t } \frac{1}{q} - v_{i_{jk}jl} = t_{jkl} & \forall \ k \\ \quad \sum_i v_{ij1} = 1 & \\ \quad t_{jkl}, v_{ijl} \geq 0 & \forall \ i, k \end{array} \right)$$

We can further deduce that $\frac{1}{q} - v_{i_jkjl} = t_{jkl}$ and $t_{jkl} \geq 0$ together imply that $\min \sum_k t_{jkl} = \min \left[\sum_k \left(\frac{1}{q} - v_{i_jkjl}\right)\right]$ and $v_{ijl} \leq \frac{1}{q}$.

Hence, problem $(LPT_j^{\bullet})$ can be further re-written as

$$(LPT_{jl}^{\bullet}) \qquad \left(\begin{array}{c} \min \left[\sum_k \left(\frac{1}{q} - v_{i_jkj1}\right)\right] \\ \text{s.t. } \sum_i v_{ijl} = 1 \\ 0 \leq v_{ijl} \leq \frac{1}{q} \qquad \forall i \end{array} \right)$$

Now, $\min \sum_k \left(\frac{1}{q} - v_{i_jkjl}\right) = \frac{p}{q} - \max \sum_k v_{i_jkjl}$. Hence $(LPT_{jl}^{\bullet})$ can be written as

$$(LPT_{jl}^{\bullet}) \qquad \left(\begin{array}{c} \frac{p}{q} - \max \left[\sum_k v_{i_jkjl}\right] \\ \text{s.t. } \sum_i v_{ijl} = 1 \\ 0 \leq v_{ij1} \leq \frac{1}{q} \qquad \forall i \end{array} \right)$$

We observe that at every feasible solution to problem $(LPT_{jl}^{\bullet})$ we have $\sum_k v_{i_jkjl} = \sum_i F(i,j) v_{ijl}$. Thus after replacing the objective function the problem $(LPT_{jl}^{\bullet})$ becomes

$$(LPT_{jl}^{\bullet}) \qquad \left(\begin{array}{c} \frac{p}{q} - \max \left[\sum_i F(i,j) v_{ijl}\right] \\ \text{s.t. } \sum_i v_{ijl} = 1 \\ 0 \leq v_{ij1} \leq \frac{1}{q} \qquad \forall i \end{array} \right)$$

$$\therefore v(LPT_{jl}^{\bullet}) = \left(\begin{array}{c} \frac{p}{q} - \max\left(\sum_i F(i,j) v_{ijl}\right) \\ \text{s.t. } \sum_i v_{ijl} = 1 \\ 0 \leq v_{ij1} \leq \frac{1}{q} \qquad \forall i \end{array} \right) = \frac{p}{q} - \sum_{e=1}^{q} \frac{1}{q} F^{max_e}(i,j) \qquad \text{(A.5)}$$

From, equation (A.5) we can conclude that

$$\begin{aligned}
v(LPT^\bullet) = \sum_j \sum_l v(LPT^\bullet_{jl}) &= \sum_j \sum_l \left[\frac{p}{q} - \sum_{e=1}^{q} \frac{1}{q} F^{max_e}(j)\right] \\
&= \sum_j q \left[\frac{p}{q} - \sum_{e=1}^{q} \frac{1}{q} F^{max_e}(j)\right] \\
&= \sum_j \left[p - \sum_{e=1}^{q} F^{max_e}(j)\right] \\
&= np - \sum_j \sum_{e=1}^{q} F^{max_e}(j) = LB_F
\end{aligned} \tag{A.6}$$

Hence from equations (A.4) and (A.6), $v(LPT) \leq v(LPT^\bullet) = LB_F$. ■

A.2 Special class of Instances of the q-mode Problem.

In this section we show that for the q-mode problem, if $q \geq m$ we have $v(LPT) = 0$. We include the formulation (LPT) below for ease of reference.

$$(LPT) \left(\begin{array}{cc}
\min \sum_j \sum_k \sum_l t_{jkl} & \\
\text{s.t } \; a_{ijk} y_{kl} - v_{ijl} \leq t_{jkl} & \forall \; i,j,k,l \\
\sum_i v_{ij1} = 1 & \forall \; j,l \\
\sum_l y_{kl} = 1 & \forall \; k \\
t_{jkl}, v_{ijl}, y_{kl} \geq 0 & \forall \; i,k
\end{array} \right)$$

Assuming $q \geq m$ let us consider a solution to (LPT) that is given as $y_{kl} = \frac{1}{q}, v_{ijl} = \frac{1}{m}, t_{jkl} = 0 \;\forall\; i,j,k,l$. We refer to this solution as S. It is easy to verify that S is a feasible solution to (LPT) since

$$\sum_l y_{kl} = q \cdot \frac{1}{q} = 1 \ \forall \ k \tag{A.7}$$

$$\sum_i v_{ijl} = m \cdot \frac{1}{m} = 1 \ \forall \ j, l \tag{A.8}$$

and,

$$\begin{aligned} \text{since } m \leq q \text{ it follows that } \frac{1}{q} \leq \frac{1}{m} \therefore y_{kl} &\leq v_{ijl} \ \forall \ i, j, k, l \\ \therefore a_{ijk} y_{kl} &\leq v_{ijl} \ \forall \ i, j, k, l \\ \therefore a_{ijk} y_{kl} - v_{ijl} &\leq 0 \ \forall \ i, j, k, l \end{aligned} \tag{A.9}$$

The objective value of solution S is 0. Hence we can conclude that the optimal objective value of the LP relaxation of (IPT) equals 0, i.e., $v(LPT) = 0$, for all problem instances where $q \geq m$.

Table A.1: Comparison of LB_F and $v(LPT)$: Instances with weak clusters.

type-m-n-p-q-1	$v(LBT)$	LB_F
l-8-10-10-2-1	45	45
l-8-10-10-3-1	29	29
l-10-20-20-2-1	246	246
l-10-20-20-3-1	190	190
l-20-20-50-2-1	782	782
l-20-20-50-3-1	686	686
l-20-20-50-5-1	556	556
l-20-20-100-2-1	1621	1621
l-20-20-100-3-1	1471	1471
l-20-20-100-5-1	1215	1215
h-8-10-10-2-1	34	34
h-8-10-10-3-1	25	25
h-10-20-20-2-1	153	153
h-10-20-20-3-1	150	150
h-20-20-50-2-1	463	463
h-20-20-50-3-1	429	429
h-20-20-50-5-1	364	364
h-20-20-100-2-1	916	916
h-20-20-100-3-1	879	879
h-20-20-100-5-1	742	742

Table A.2: Comparison of LB_F and $v(LPT)$: Instances with strong clusters.

type-m-n-p-q-1	$v(LBT)$	LB_F
g-8-10-10-2-1	33	33
g-8-10-10-3-1	20	20
g-10-20-20-2-1	126	126
g-10-20-20-3-1	112	112
g-20-20-50-2-1	494	494
g-20-20-50-3-1	471	471
g-20-20-50-5-1	437	437
g-20-20-100-2-1	984	984
g-20-20-100-3-1	982	982
g-20-20-100-5-1	900	900
s-8-10-10-2-1	50	50
s-8-10-10-3-1	28	28
s-10-20-20-2-1	248	248
s-10-20-20-3-1	188	188
s-20-20-50-2-1	639	639
s-20-20-50-3-1	616	616
s-20-20-50-5-1	573	573
s-20-20-100-2-1	1343	1343
s-20-20-100-3-1	1303	1303
s-20-20-100-5-1	1256	1256
p-8-10-10-2-1	31	31
p-8-10-10-3-1	24	24
p-10-20-20-2-1	136	136
p-10-20-20-3-1	125	125
p-20-20-50-2-1	345	345
p-20-20-50-3-1	341	341
p-20-20-50-5-1	327	327
p-20-20-100-2-1	730	730
p-20-20-100-3-1	732	732
p-20-20-100-5-1	713	713

Appendix B

Detailed Computational Results.

Table B.1: Comparison of $v(LPR)$, $v(LPT)$ and $v(LPA)$: Instances with weak clusters.

type-m-n-p-q-1	$v(LPR)$	τ_R	$v(LPT)$	τ_T	$v(LPA)$	τ_A
l-8-10-10-2-1	22.5	0.06	45	0.07	34	0.05
l-8-10-10-3-1	9.67	0.1	29	0.12	0	0.1
l-10-20-20-2-1	123	0.5	246	0.68	223.2	0.33
l-10-20-20-3-1	63.33	1.88	190	2.79	147.3	0.96
l-20-20-50-2-1	391	3.42	782	2.13	761	0.86
l-20-20-50-3-1	228.67	15.7	686	19.86	639	3.65
l-20-20-50-5-1	111.2	61.45	556	93.85	430.25	24.54
l-20-20-100-2-1	810.5	20.94	1621	12.15	1568	1.78
l-20-20-100-3-1	490.33	85.06	1471	28.03	1355	11.32
l-20-20-100-5-1	243	347.64	1215	568.67	1031.25	61.12
h-8-10-10-2-1	17	0.05	34	0.06	22	0.03
h-8-10-10-3-1	8.33	0.11	25	0.13	0	0.09
h-10-20-20-2-1	76.5	0.44	153	0.59	108	0.18
h-10-20-20-3-1	50	1.62	150	2.28	72.1	0.77
h-20-20-50-2-1	231.5	1.95	463	2.58	396	0.62
h-20-20-50-3-1	143	11.1	429	15.09	313	2.79
h-20-20-50-5-1	72.8	49.7	364	78.45	155.5	11.12
h-20-20-100-2-1	458	6.73	916	6.66	833	1.13
h-20-20-100-3-1	293	49.09	879	77.3	695	6.63
h-20-20-100-5-1	148.4	252.99	742	419.46	439	40.77

Table B.2: Comparison of $v(LPR)$, $v(LPT)$ and $v(LPA)$: Instances with strong clusters.

type-m-n-p-q-1	$v(LPR)$	τ_R	$v(LPT)$	τ_T	$v(LPA)$	τ_A
g-8-10-10-2-1	16.5	0.05	33	0.06	20	0.03
g-8-10-10-3-1	6.67	0.1	20	0.12	0	0.09
g-10-20-20-2-1	63	0.44	126	0.55	92	0.16
g-10-20-20-3-1	37.33	1.34	112	2.02	40.7	0.53
g-20-20-50-2-1	247	3.67	494	4.28	432	0.64
g-20-20-50-3-1	157	13.68	471	18.93	391	2.16
g-20-20-50-5-1	87.4	56.14	437	84.63	265	10.1
g-20-20-100-2-1	492	15.85	984	23.12	916	0.71
g-20-20-100-3-1	327.33	67.38	982	100.78	815	3.47
g-20-20-100-5-1	180	292.09	900	511.62	618	18.69
s-8-10-10-2-1	25	0.06	50	0.07	44	0.05
s-8-10-10-3-1	9.33	0.11	28	0.13	13.25	0.11
s-10-20-20-2-1	124	0.54	248	0.66	223	0.24
s-10-20-20-3-1	62.67	1.75	188	2.48	160	0.73
s-20-20-50-2-1	319.5	3.71	639	4.42	614	0.57
s-20-20-50-3-1	205.33	14.3	616	18.83	565	0.96
s-20-20-50-5-1	114.60	68.98	573	80.86	454	19.03
s-20-20-100-2-1	671.5	20.7	1343	18.79	1304	0.5
s-20-20-100-3-1	434.33	79.1	1303	96.17	1235	2.54
s-20-20-100-5-1	251.2	348.31	1256	571.99	1068.5	88.88
p-8-10-10-2-1	15.5	0.04	31	0.06	22	0.04
p-8-10-10-3-1	8	0.12	24	0.12	7.75	0.08
p-10-20-20-2-1	68	0.42	136	0.67	104	0.11
p-10-20-20-3-1	41.67	1.33	125	2.12	82.2	0.61
p-20-20-50-2-1	172.5	3.14	345	4.62	288	0.39
p-20-20-50-3-1	113.67	11.56	341	19.24	268	1.29
p-20-20-50-5-1	65.4	49.74	327	80.46	145	4.46
p-20-20-100-2-1	365	15.40	730	21.52	662	0.41
p-20-20-100-3-1	244	64.22	732	93.19	572	2.20
p-20-20-100-5-1	142.6	261.36	713	547.52	485	6.44

Table B.3: Detailed computational results for the local improvement algorithm : weakly clustered instances.

type-m-n-p-q-1	V^*	V_{ini}	V_{LI}	τ_{LI}	V_{LI^1}	τ_{LI^1}	V_{LI_α}	τ_{LI_α}
l-8-10-10-2-1	55	59	56	0.01	56	0	56	0
l-8-10-10-3-1	45	53	46	0	46	0.01	46	0.01
l-10-20-20-2-1	273	291	279	0.01	279	0.01	273	0.01
l-10-20-20-3-1	251	277	252	0.04	250	0.03	254	0.02
l-20-20-50-2-1	820	850	830	0.1	830	0.15	830	0.11
l-20-20-50-3-1	790	825	785	0.57	781	0.29	775	0.67
l-20-20-50-5-1	757	785	731	7.22	734	1.45	726	6.31
l-20-20-100-2-1	1710	1749	1707	1.03	1702	1.1	1706	0.76
l-20-20-100-3-1	1670	1734	1645	7.27	1651	1.34	1653	2.36
l-20-20-100-5-1	1651	1672	1574	81.2	1569	10.76	1567	85.7
l-20-20-100-8-1	1584	1613	1484	-	1473	177.16	1472	1489.66
h-8-10-10-2-1	41	59	41	0	41	0	41	0
h-8-10-10-3-1	37	47	40	0.01	40	0.01	40	0.01
h-10-20-20-2-1	172	243	172	0.02	172	0.02	172	0.01
h-10-20-20-3-1	195	253	195	0.04	195	0.04	195	0.04
h-20-20-50-2-1	494	690	494	0.18	494	0.25	494	0.18
h-20-20-50-3-1	491	710	491	0.77	491	0.48	491	0.33
h-20-20-50-5-1	486	737	486	9.68	486	2.12	486	4.08
h-20-20-100-2-1	968	1384	968	1.72	968	1.74	968	1.23
h-20-20-100-3-1	984	1502	984	8.85	984	3.29	984	5.46
h-20-20-100-5-1	945	1566	945	143.13	945	22.79	945	74.86
h-20-20-100-8-1	943	1577	943	-	943	334.09	943	1747.57

Table B.4: Detailed computational results for the local improvement algorithm : strongly clustered instances.

type-m-n-p-q-1	V^*	V_{ini}	V_{LI}	τ_{LI}	V_{LI^1}	τ_{LI^1}	V_{LI_α}	τ_{LI_α}
g-8-10-10-2-1	40	56	40	0.01	40	0.01	40	0
g-8-10-10-3-1	26	53	26	0.01	26	0.01	26	0
g-10-20-20-2-1	127	242	127	0.01	127	0.01	127	0.02
g-10-20-20-3-1	121	253	121	0.04	121	0.02	121	0.03
g-20-20-50-2-1	494	690	494	0.16	494	0.23	494	0.16
g-20-20-50-3-1	475	755	475	0.78	475	0.3	475	0.31
g-20-20-50-5-1	464	765	464	11.56	464	1.26	464	3.73
g-20-20-100-2-1	984	1440	984	1.78	984	1.91	984	1.31
g-20-20-100-3-1	991	1541	991	8.55	991	2.55	991	3.33
g-20-20-100-5-1	973	1613	973	150.44	973	38.14	973	79.12
g-20-20-100-8-1	964	1566	964	-	964	366.62	964	1124.11
s-8-10-10-2-1	50	68	50	0.01	50	0	50	0.01
s-8-10-10-3-1	28	54	28	0.01	28	0.01	28	0.01
s-10-20-20-2-1	251	298	251	0.01	251	0.02	251	0.01
s-10-20-20-3-1	193	288	193	0.04	193	0.04	193	0.04
s-20-20-50-2-1	641	787	641	0.19	641	0.18	641	0.15
s-20-20-50-3-1	622	797	622	0.69	622	0.44	622	0.36
s-20-20-50-5-1	578	799	578	13.32	578	2.57	578	5.15
s-20-20-100-2-1	1344	1620	1344	1.84	1344	1.03	1344	1.1
s-20-20-100-3-1	1310	1642	1310	8.48	1310	3.31	1310	3.44
s-20-20-100-5-1	1260	1697	1260	153.67	1260	20.92	1260	59.35
s-20-20-100-8-1	1053	1574	1053	-	1127	235.99	1053	1585.92
p-8-10-10-2-1	31	58	31	0.01	31	0	31	0
p-8-10-10-3-1	28	53	28	0.01	28	0	28	0.01
p-10-20-20-2-1	136	182	136	0.01	136	0.02	136	0.01
p-10-20-20-3-1	125	244	125	0.04	125	0.03	125	0.03
p-20-20-50-2-1	345	635	345	0.19	345	0.11	345	0.15
p-20-20-50-3-1	341	715	341	0.73	341	0.34	341	0.4
p-20-20-50-5-1	327	734	327	8.89	327	3.55	327	5.58
p-20-20-100-2-1	730	1309	730	1.82	730	2.29	730	1.38
p-20-20-100-3-1	732	1449	732	8.66	732	2.84	732	3.97
p-20-20-100-5-1	713	1587	713	148.87	713	31.71	713	55.22
p-20-20-100-8-1	690	1608	813	-	813	430.96	690	1344.88

Appendix C

File Format: Instance of the q-mode problem.

Line 1: paramters	The first line contains the values of the paramaters m, n, p and q in this sequence. Each parameter value is separated by a single space.
Lines 2 to (p**+1)**: The records	Each line contains one record, i.e., each line contains n numbers each number separated by a single space. The j^{th} number in the $(k+1)^{th}$ line is the value assigned to position j of record k.
Line $(p+2)$ contains either 0 or -1.	
Line $(p+2)$ contains 0	The next q lines contain the q mode vectors corresponding to a solution, i.e., there are q more lines and each line contains n numbers separated by a single space.
Line $(p+2)$ contains -1	The next p lines contain a cluster assignment of records to clusters, i.e., there are p more lines, each line containing two numbers separated by a single space. The first number identifies the record and the second number identifies the cluster to which that record is assigned.

Appendix D

C++ program: Random Generation of q-mode Instances.

```
/* This program generates data for the q-model problem
   The paramaters that need t be supplied are
   1. p -> number of records.
   2. q -> number of clusters.
   3. n -> number of attributes for each record.
   4. m -> number of values for each attribute.
   5. type -> type of pmf to be used.
   6. ns -> number of pmfs in the pmf_set.

*/

#define debug 1

#include "genData.h"

// begin main program
int main(int argc, char* argv[])
{

  int n,m,p,q,ns,type; //parameter variables
  int i,j,k,l;
  int temp;

  /*
    Check for the command line input if correct number
    of parameters are entered then store them in the appropriate
    variables else prompt user for input.
  */

  if (argc != 7)
  {
    cout<<"Incorrect number of parameters. USAGE is: \n";
    cout<<">> "<<argv[0]<<" m n p q ns type \n";
    return 0;
  }
  else
  {
    m = atoi(argv[1]);
    n = atoi(argv[2]);
    p = atoi(argv[3]);
    q = atoi(argv[4]);
    ns = atoi(argv[5]);
    type = atoi(argv[6]);
    //n = atoi(argv[1]);
    #if debug ==1
      cout<<m<<" "<<n<<" "<<p<<" "<<q<<" "<<ns<<" "<<type;
    #endif
  }


  /*
    Allocate memory to all variables
    1. permutation(ns,m)
    2. pmf_set(ns,m)
    3. cpf_set(ns,m)
    4. record_set(p,n)
    5. prob_values(m)
    6. profile(q,n)
  */

  int **permutation;
  int **record_set;
  int **final_record;
  float *prob_values;
  float **pmf_set;
  float **cpf_set;
  int **profile;
  int *clustAssign;
  int *finalClustAssign;

  char *pmfname;
```

```
  permutation = new int*[ns+1];
  for(l=1;l<=ns;l++)
  {
    permutation[l] = new int[m+1];
    for(i=1;i<=m;i++)
    {
      permutation[l][i] =0;
    }
  }

  pmf_set = new float*[ns+1];
  for(l=1;l<=ns;l++)
  {
    pmf_set[l] = new float[m+1];
    for(i=1;i<=m;i++)
    {
      pmf_set[l][i] =0;
    }
  }

  cpf_set = new float*[ns+1];
  for(l=1;l<=ns;l++)
  {
    cpf_set[l] = new float[m+1];
    for(i=1;i<=m;i++)
    {
      cpf_set[l][i] =0;
    }
  }

  record_set = new int*[p+1];
  for(k=1;k<=p;k++)
  {
    record_set[k] = new int[n+1];
    for(j=1;j<=n;j++)
    {
      record_set[k][j]=0;
    }
  }

  final_record = new int*[p+1];
  for(k=1;k<=p;k++)
  {
    final_record[k] = new int[n+1];
    for(j=1;j<=n;j++)
    {
      final_record[k][j]=0;
    }
  }

  prob_values = new float[m+1];
  for(i=1;i<=m;i++)
  {
    prob_values[i] = 0;
  }

  profile = new int*[q+1];
  for(k=1;k<=q;k++)
  {
    profile[k] = new int[n+1];
    for(j=1;j<=n;j++)
    {
      profile[k][j]=0;
    }
  }

  clustAssign = new int[p+1];
  for(k=1;k<=p;k++)
  {
    clustAssign[k] = 0;
  }
```

```
finalClustAssign = new int[p+1];
for(k=1;k<=p;k++)
{
  finalClustAssign[k] = 0;
}


/*  Assign the probability values based on the type selected
  type  name      function
  1 -> uniform
  2 -> linear     setLinearProb(m,r, prob_values)
  3 -> manyStep
  4 -> oneStep
  5 -> partLinear
  6 -> oneLarge
  7 -> oneHuge
  8 -> onlyOne
  9 -> geometric
*/

int r;

switch(type)
{
  case 1 :
    cout<<"case Uniform\n";
    pmfname = "Uniform";
    setUniformProb(m,prob_values);
    createLinearCpfset(m, ns, pmf_set, cpf_set, prob_values);
    break;
  case 2 :
    cout<<"case Linear\n";
    pmfname = "Linear";
    cout<<"Input r\n";
    cin>>r;
    setLinearProb(m,r, prob_values);
    createLinearCpfset(m, ns, pmf_set, cpf_set, prob_values);
    break;
  case 3:
    cout<<"case manyStep\n";
    pmfname = "manyStep";
    cout<<"Input r\n";
    cin>>r;
    //setmanyStepProb(m,r, prob_values);
    createLinearCpfset(m, ns, pmf_set, cpf_set, prob_values);
    break;
  case 4 :
    cout<<"case oneStep\n";
    pmfname = "oneStep";
    //cout<<"Input r\n";
    //cin>>r;
    k = 3;
    setoneStepProb(m,k, prob_values);
    createLinearCpfset(m, ns, pmf_set, cpf_set, prob_values);
    break;
  case 5 :
    cout<<"case partLinear\n";
    pmfname = "partLinear";
    cout<<"Input r\n";
    cin>>r;
    setpartLinearProb(m,3, r, prob_values);
    createLinearCpfset(m, ns, pmf_set, cpf_set, prob_values);
    break;
  case 6 :
    cout<<"case oneLarge\n";
    pmfname = "oneLarge";
    cout<<"Input r\n";
    cin>>r;
    //setoneLargeProb(m,r, prob_values);
    createLinearCpfset(m, ns, pmf_set, cpf_set, prob_values);
    break;
```

131

```
  case 7 :
    cout<<"case oneHuge\n";
    pmfname = "oneHuge";
    cout<<"Input r\n";
    cin>>r;
    setoneHugeProb(m,r, prob_values);
    createLinearCpfset(m, ns, pmf_set, cpf_set, prob_values);
    break;
  case 8 :
    cout<<"case onlyOne\n";
    pmfname = "onlyOne";
    cout<<"Input r\n";
    cin>>r;
    //setonlyOneProb(m,r, prob_values);
    createLinearCpfset(m, ns, pmf_set, cpf_set, prob_values);
    break;

  case 9 :
    cout<<"case Geometric\n";
    pmfname = "Geometric";
    cout<<"Input r\n";
    cin>>r;
    setGeoProb(m,r, prob_values);
    createLinearCpfset(m, ns, pmf_set, cpf_set, prob_values);
}

createRecordset(n,p,m,q, ns, record_set, cpf_set,profile);

#if debug == 1
  cout<<"\nPrinting out the pmf to be used \n[";
  for(i=1;i<=m;i++)
  {
    cout<<prob_values[i]<<" ";
  }
  cout<<"]\n";

  cout<<"Printing out the cpf set to be used to create the "<<ns<<" clusters";
  for(l=1;l<=ns;l++) // for each pmf
  {
    cout<<endl;
    for(i=1;i<=m;i++)
    {
      cout<<cpf_set[l][i]<<" ";
    }
  }
  cout<<endl;

#endif

  /* print out the profiles
  cout<<"The profiles are \n"<<endl;
  for(l=1;l<=q;l++)
  {
    for(j=1;j<=n;j++)
    {
      cout<<profile[l][j]<<" ";
    }
    cout<<endl;
  }

  // print out the most likely largest occurring value
  cout<<endl;
  cout<<"The most likely models are\n";
  for(l=1;l<=q;l++)
  {
    for(j=1;j<=n;j++)
    {
      temp = profile[l][j];
      cout<<(temp-1)*floor(m/ns) +1<<" ";

    }
    cout<<endl;
    }*/
    //predicted optimal solution
    int rpc = int(p/q);
    for(l=1;l<=q;l++)
    {
      for(temp=1;temp<=rpc;temp++)
      {
        clustAssign[((l−1)*rpc+temp)]=l;
      }
    }
    for(temp=q*rpc+1;temp<=p;temp++)
    {
      clustAssign[temp]=q;
    }

    //reordering the record set
    int *recordflag = new int[p+1];
    for(k=0;k<=p;k++)
    {
      recordflag[k] =1;
    }

    int i_in;
    int temprnd=0;
    int k1,k2;
    for(k=p;k>=1;k−−)
    {
      //select a random number between 1 and k
      srand48(k*type);
      temprnd = int(k*drand48())+1;
      cout<<k<<" "<<(temprnd = int(k*drand48()) +1)<<endl;
      k1=0;
      k2=1;
      //cin>>i_in;
      while(k2 <= p)
      {
        if(recordflag[k2]==1)
        {
          k1++;
          if(k1==temprnd)
          {
            break;
          }
        }
        k2++;
      }
      //k2=k2−1;
      recordflag[k2]=0;

      //assign record number k to k2 position
      for(j=1;j<=n;j++)
      {
        final_record[k2][j] = record_set[k][j];
      }
      finalClustAssign[k2] = clustAssign[k];
    }

    //cin>>i_in;

    // create filename
    char* file = new char[100];

    sprintf(file,"%s%s%i%s%i%s%i%s%i%s%i%s",pmfname,"_",m,"_",n,"_",p,"_"
             ,q,"_",1,".data");

    //open file
    ofstream out(file);
    if(!out)
    {
      cout<<"Error writing to file"<<file<<endl;
      return 1;
    }
```

```

    #if debug == 1
      cout<<m<<" "<<n<<" "<<p<<" "<<q<<endl;
      for(k=1;k<=p;k++)
      {
        for(j=1;j<=n;j++)
        {
          cout<<final_record[k][j]<<" ";
        }
        cout<<endl;
      }
      cout<<"−1\n";
      for(k=1;k<=p;k++)
      {
        cout<<k<<" "<<finalClustAssign[k]<<endl;
      }
    #endif

    out<<m<<" "<<n<<" "<<p<<" "<<q<<endl;
    for(k=1;k<=p;k++)
    {
      for(j=1;j<=n;j++)
      {
        out<<final_record[k][j]<<" ";
      }
      out<<endl;
    }
    out<<"−1\n";
    for(k=1;k<p;k++)
    {
      out<<k<<" "<<finalClustAssign[k]<<endl;
    }
    out<<k<<" "<<finalClustAssign[k];
    out.close();



  //calcModelProb(m,n,p,q,pmf_set);

/* Calculate permutation based on type */

/* Calculate pmf_set based on permutation and prob_values */

/* Calculate cpf_set based on pmf_set */

/* Create records based on cpf_set */
}


//This functions creates the standard form of the Linear pmf

void setUniformProb(int m, float *prob)
{
  int i;
  float em = m;
  float b = 1/em;
  for(i=1;i<=m;i++)
  {
    prob[i] = b;

  }
}

//This functions creates the standard form of the Linear pmf

void setLinearProb(int m, int r, float *prob)
{
  float a,b;

  b= float(m*(r+1));
  b = 2/b;
  #if debug == 1
    cout<<"m ="<<m<<"r ="<<r<<"b = "<<b<<endl;
  #endif

  int i;
  for(i=m-1;i>=0;i--)
  {
    prob[m-i] = b + (float)(i*(r-1)*b)/(m-1);

  }
}


//This functions creates the standard form of the Geometric pmf
void setGeoProb(int m, int r, float *prob)
{
  int i;
  double rm=1;
  float a,b;

  for(i=1;i<=m;i++)
  {
    rm = rm*r;
    #if debug == 1
      cout<<"rm ="<<rm<<" i="<<i<<endl;
    #endif
  }

  #if debug == 1
    cout<<"rm ="<<rm<<" r="<<r<<endl;
  #endif

  b =float(r-1)/float(rm-1);

  #if debug == 1
    cout<<"m ="<<m<<"r ="<<r<<"b = "<<b<<endl;
  #endif



  prob[m]=b;
  for(i=m-1;i>=1;i--)
  {
    #if debug == 1
      cout<<prob[i+1];
    #endif

    prob[i] = r*prob[i+1];

  }

  #if debug == 1
    cout<<prob[i+1];
  #endif
}

//This functions creates the standard form of the partLinear pmf
void setpartLinearProb(int m, int k, int r, float *prob)
{
  float a,b;

  b = float(k*(r+1));
  b = 2/b;

  #if debug == 1
    cout<<"k ="<<k<<"r ="<<r<<"b = "<<b<<endl;
  #endif

  int i;

  for(i=1;i<=m;i++)
```

133

```
  {
    prob[i]=0;
  }

  for(i=1;i<=k;i++)
  {
    prob[i] = b + float(((k-i)*(r-1)*b)/(k-1));
  }

  /*for(i=1;i<=m;i++)
  {
    cout<<prob[i]<<" ";
  }
  cout<<endl;*/

}


//This functions creates the standard form of the oneStep pmf
void setoneStepProb(int m, int k, float *prob)
{
  int i;
  float kay=k;
  #if debug == 1
    cout<<"m ="<<m<<"k ="<<k<<endl;
  #endif

  for(i=1;i<=m;i++)
  {
    prob[i] =0;
  }

  for(i=1;i<=k;i++)
  {
    prob[i] = 1/kay;

  }
}

//This functions creates the standard form of the oneHuge pmf
void setoneHugeProb(int m, int r, float *prob)
{
  int i;
  float em=m;
  float er=r;

  #if debug == 1
    cout<<"m ="<<m<<"r ="<<r<<endl;
  #endif

  prob[1]=er/(er+em-1);
  cout<<prob[1]<<endl;
  for(i=2;i<=m;i++)
  {
    prob[i] = 1/(er+em-1);
    cout<<prob[i];
  }

}

/* This function creates "ns" different pmfs.
This scheme for creating pmfs can be used only if ns<=m.
All the pmfs are of the same pmf type.

*/
void createLinearCpfset(int m, int ns, float **pmfset, float** cpfset, float  *p
rob)
 {
  int l,i;
  int start =0;
  int f = m/ns;
  // create ns different pmfs
  for(l=1;l<=ns;l++) // for each pmf
  {
    start = (l-1)*f +1;
    for(i=1;i<=m;i++)
    {
      if(start != m+1)
      {
        pmfset[l][start] = prob[i];
        start++;
      }
      else
      {
        pmfset[l][1] = prob[i];
        start =2;
      }

    }
  }

  // use the pmfs to create "ns" different cpfs
  for(l=1;l<=ns;l++) // for each pmf
  {
    cpfset[l][1] = pmfset[l][1];
    for(i=2;i<=m;i++)
    {
      cpfset[l][i] = pmfset[l][i] + cpfset[l][i-1];
    }
  }


 }

void createRecordset(int n,int p,int m,int q, int ns, int **recordset, float **c
pfset, int **profile)
{
  int *pmf_no;
  int randInt;
  int temp, i,j,l,l2;
  int rpc= (int) (p/q); /*records per cluster. The last cluster has an
              additional p%q records */
  int temprpc;
  int pmfSelected;
  int attrib;

  float randFloat;

  #if debug ==1
    cout<<"rpc = "<<rpc;
  #endif

  pmf_no = new int[m+1];
  for(i=1;i<=m;i++)
  {
    pmf_no[i]=1;
  }

  for(j=1;j<=n;j++) // for each attibute j
  {
    for(l2=1;l2<=ns;l2++)
    {
      pmf_no[l2] =1;
    }

    for(l=1;l<=q;l++)
    {
      #if debug ==1
        cout<<"\nl = "<<l<<"j= "<<j<<endl;
      #endif
      // randomly selecting one pmf from the pmf set
      randInt = (int) ((ns-l+1)*drand48()) +1;
```

```
      #if debug ==1
        cout<<"randInt = "<<randInt;
      #endif
      temp=0;
      pmfSelected=0;

      /* identify the pmf that has number "randInt"
         in the pmf_set that remains */
      for(l2=1;l2<=ns;l2++)
      {
        if(pmf_no[l2] ==1)
        {
          temp++;
          if(temp == randInt)
          {
            pmf_no[l2]=0;
            pmfSelected= l2;
            #if debug ==1
              cout<<" pmfSelected ="<<pmfSelected<<endl;
            #endif
            profile[l][j] = pmfSelected;
            break;
          }
        }

      } //end for l2

      /* assign values to the jth attibute of
         records in the lth cluster using pmf pmfSelected */

      for(temprpc=1;temprpc<=rpc;temprpc++)
      {
        randFloat = drand48();
        #if debug ==1
          //cout<<"randFloat="<<randFloat<<endl;
        #endif
        for(i=1;i<=m;i++)
        {
          if(randFloat<cpfset[pmfSelected][i])
          {
            attrib = ((l-1)*rpc+temprpc);
            #if debug ==1
              cout<<"attrib = "<<attrib<<" l= "<<l<<" rpc= "<<rpc<<" temprpc = "<<temprpc<
<endl;
            #endif
            recordset[attrib][j] = i;
            #if debug ==1
              cout<<i<<" ";
            #endif
            break;
          }
        }

      } // end for temprpc


    } //end for l

    for(temprpc=rpc*q+1;temprpc<=p;temprpc++)
    {
      randFloat = drand48();
      for(i=1;i<=m;i++)
      {
        if(randFloat<cpfset[pmfSelected][i])
        {
          recordset[temprpc][j] = i;
          break;
        }
      }

    } //end for temprpc
```

```
  } //end for j


} // end function


long NchooseR(int n, int r)
{
  int i;
  long num=1;
  long denom=1;

  if(r<int(n/2))
  {
    r = n-r;
  }

  for(i=1;i<=n-r;i++)
  {
    num = (num*long(n-i+1))/long(i);
  }

  //num = num/denom;

  return num;
}

double power(double r, int n)
{
  double pow=1;
  int i;

  for(i=1;i<=n;i++)
  {
    pow = pow*r;
  }
  return pow;
}

int power(int r, int n)
{
  int pow=1;
  int i;

  for(i=1;i<=n;i++)
  {
    pow = pow*r;
  }
  return pow;
}
```

Appendix E

C++ Program: Reading a q-mode Instance from File.

```
void dataset::readinData(char *filename)
{

  int i,j,k,l;

  // variables related to reading data file
  char *line;
  char *tempch;
  char *file;
  int tempi;
  int tempr;
  int i_in;     // temp variable for read
  int iLineNo;  // number of the line being read −1;
  int iPos;     // position being read for an order
  int ifirstline; // check if first line has been read
  int recordNo;

  #if debug == 1
    cout<<"*** Inside function dataset::readinData ***";
  #endif

  line = new char[2000];
  tempch = new char[4];
  file = new char[1000];
  file = filename;
  // obtain data from file

  /* Data file has format
     M N P Q <− first line
     one order per line   */

  #if debug ==1
    cout<<"Opening file "<<file<<endl;
  #endif

  ifstream filestr(file); //input file stream

  if (!filestr.is_open())
  {
    cout<<" Could not open file :"<<endl;
    exit;
  }

    // Read parameters M N P Q from first line
  if(!filestr.getline(line,2000).eof())
  {

    #if Debug == 1
      cout<<"Line is : "<<line<<endl;
    #endif

    //reading M
    tempch = strtok(line," ");
    M = atoi(tempch);
    #if Debug == 1
      cout<<M<<endl;
    #endif

    //reading N
    tempch = strtok(NULL," ");
    N = atoi(tempch);
    #if Debug == 1
      cout<<N<<endl;
    #endif

    //reading P
    tempch = strtok(NULL," ");
    P = atoi(tempch);
    #if Debug == 1
      cout<<P<<endl;
    #endif
```

```
    //reading Q
    tempch = strtok(NULL," ");
    Q = atoi(tempch);
    #if Debug == 1
      cout<<Q<<endl;
    #endif

  }
  else
  {
    cout<<"ERROR in the input file\n";
    exit;
  }

    //allocating memory to the variables

  try
  {

    record = new int*[P+1];
    for(k=1;k<=P;k++)
    {
      record[k] = new int[N+1];
      for(j=1;j<=N;j++)
      {
        record[k][j]=0;
      }
    }

    model = new int*[Q+1];
    for (l=1;l<=Q;l++)
    {
      model[l] = new int [N+1];
      for(j=1;j<=N;j++)
      {
        model[l][j]=0;
      }
    }

    clustAssign = new int[P+1];
    for(k=1;k<=P;k++)
    {
      clustAssign[k]=0;
    }


    #if debug == 1
      cout <<"Memory allocation completed\n";
    #endif

  }
  catch(bad_alloc mem_exp)
  {
    cout<<"Memory Allocation  failure \n";
    exit;
  }

  // Read the data file for order data

  iLineNo = 1;
  int modelData =0;
  int clustAssFlag =0;
  int tempint=0;
  while(!filestr.getline(line,2000).eof())
  {
    #if Debug == 1
      cout<<"Line is : "<<line<<endl;
    #endif

    tempch = strtok(line," ");
    if(atoi(tempch) ==0)
    {
      modelData=1;
      break;
    }
    else if(atoi(tempch) <0)
    {
      clustAssFlag=1;
      break;
    }
    else
    {
      record[iLineNo][1] = atoi(tempch);

      #if Debug == 1
        cout<<endl<<record[iLineNo][1]<<" ";
      #endif

      iPos = 2;
      while(iPos<=N)
      {
        tempch = strtok(NULL," ");
        record[iLineNo][iPos] = atoi(tempch);
        #if Debug == 1
          cout<<record[iLineNo][iPos]<<" ";
        #endif
        iPos++;
      }

      iLineNo++;
    }
  }

  // This reads in model data
  if(modelData==1)
  {
    iLineNo=1;
    while(!filestr.getline(line,2000).eof())
    {
      #if Debug == 1
        cout<<"Line is : "<<line<<endl;
      #endif

      tempch = strtok(line," ");
      model[iLineNo][1] = atoi(tempch);

      #if Debug == 1
        cout<<endl<<model[iLineNo][1]<<" ";
      #endif

      iPos = 2;
      while(iPos<=N)
      {
        tempch = strtok(NULL," ");
        model[iLineNo][iPos] = atoi(tempch);
        #if Debug == 1
          cout<<model[iLineNo][iPos]<<" ";
        #endif
        iPos++;
      }
      iLineNo++;

    }

  }

  //This reads cluster assignment data
  if(clustAssFlag==1)
  {
    iLineNo=1;
    while(!filestr.getline(line,2000).eof())
    {
      #if Debug == 1
        cout<<"Line is : "<<line<<endl;
```

```
      #endif

      tempch = strtok(line," ");
      recordNo = atoi(tempch);
      #if Debug == 1
        cout<<endl<<recordNo<<" ";
      #endif


      tempch = strtok(NULL," ");
      clustAssign[recordNo] = atoi(tempch);
      #if Debug == 1
        cout<<clustAssign[recordNo];
      #endif


      iLineNo++;

    }

    #if Debug == 1
      cout<<",Initial solution is "<<this−>create_model(record,clustAssign,
     M,N,P,Q)<<endl;
    #endif
  }


  filestr.close();

  #if debug == 1
    cout<<"file "<<file<<" closed\n *** End of function dataset::readinData ***\n";
  #endif

//  delete[] line;
//  delete[] tempch;
//  delete[] file;

}
```