8-2007

Algorithms for Multi-Sample Cluster Analysis

Fahad Almutairi
University of Tennessee - Knoxville

To the Graduate Council:

I am submitting herewith a dissertation written by Fahad Almutairi entitled "Algorithms for Multi-Sample Cluster Analysis." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Management Science.

Kenneth C. Gilbert, Major Professor

We have read this dissertation and recommend its acceptance:

Hamparsum Bozdogan, Kenneth B. Kahn, Charles E. Noon

Accepted for the Council:
Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a dissertation written by Fahad Almutairi entitled "Algorithms for Multi-Sample Cluster Analysis." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Management Science.

Kenneth C. Gilbert
Major Professor

We have read this dissertation
and recommend its acceptance:

Hamparsum Bozdogan

Kenneth B. Kahn

Charles E. Noon

Accepted for the Council:

Carolyn R. Hodges
Vice Provost and Dean
of the Graduate School

(Original signatures are on file with official student records.)

Algorithms for Multi-Sample Cluster Analysis

A Dissertation Presented for the

Doctor of Philosophy Degree

The University of Tennessee, Knoxville

Fahad Almutairi

August 2007

Dedication

I am honored to dedicate my dissertation to my beloved country Kuwait.

Acknowlegment

I am thankful for Dr. H. Bozdogan for introducing me to the MSCA problem and for his comments and suggestions. I would like to thank Dr. C. Noon and Dr. K. Kahn for sharing their ideas and insights with me. I can not thank enough my advisor Dr. K. Gilbert. This work would not have been possible without his help and support.

Finally, I would like to thank the faculty, staff, and graduate students of the SOMS department for their kindness and professionalism.

Abstract

In this study, we develop algorithms to solve the Multi-Sample Cluster Analysis (MSCA) problem. This problem arises when we have multiple samples and we need to find the statistical model that best fits the cluster structure of these samples. One important area among others in which our algorithms can be used is international market segmentation. In this area, samples about customers' preferences and characteristics are collected from different regions in the market. The goal in this case is to join the regions with similar customers' characteristics in clusters (segments).

We develop branch and bound algorithms and a genetic algorithm. In these algorithms, any of the available information criteria (AIC, CAIC, SBC, and ICOMP) can be used as the objective function to be optimized. Our algorithms use the Clique Partitioning Problem (CPP) formulation. They are the first algorithms to use information criteria with the CPP formulation.

When the branch and bound algorithms are allowed to run to completion, they converge to the optimal MSCA alternative. These methods also proved to find good solutions when they were stopped short of convergence. In particular, we develop a branching strategy which uses a "look-ahead" technique. We refer to this strategy as the complete adaptive branching strategy. This strategy makes the branch and bound algorithm quickly search for the optimal solution in multiple branches of the enumeration tree before using a depth-first branching strategy. In computational tests, this method's performance was superior to other branching methods as well as to the genetic algorithm.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Cluster analysis is the assignment of a number of objects into homogenous and mutually exclusive subsets, known as clusters, such that both the degree of similarity of the objects *within* each subset and the degree of dissimilarity *between* all subsets is at the maximum level possible. Because cluster analysis methods have been used in a wide variety of areas, including biology, psychology, medicine, artificial intelligence, pattern recognition, computer science, and market segmentation, it is almost impossible to survey all that has been written on this subject. However, here we are considering the Multi-Sample Cluster Analysis (MSCA) problem, which was originally introduced by Bozdogan (1981, 1986). This problem is described in detail in Section 1.1. Many other sources of general information on clustering individuals are also available, see, e.g., Everitt (1993) and others.

Cluster analysis requires having a specific mathematical objective function, which needs to be maximized or minimized in order to determine the best clustering alternative. Many kinds of objective functions have been used.

The type of objective function used depends on the area of study it is being applied in and the purpose of the study. The objective functions used include the Within Groups Sum of Squares (WGSS), the maximum cluster diameter, sum of binary relations, classical hypothesis testing, and functions of the determinant and/or trace of the within and/or between sum of squares and products matrices (Rao 1971; Marriott 1982; Grotschel and Wakabayashi 1989). In this study, we consider all of the available general statistical information criteria. These information criteria are described in Section 1.2.

As previously noted, the MSCA problem arises in many areas of applications, including biology and remote sensing. However, the area of application we are mainly interested in for this study is the area of international market segmentation. International market segmentation has gained growing attention recently due to the increasing pressures of globalization and competition (Steenkamp and Hofstede 2002). An overview of this area of application is presented in Section 1.3.

The clustering of individuals problem has a combinatorial nature, and many combinatorial algorithms have been developed to solve it. The clustering problem has been formulated as an Uncapacitated Facility Location problem or as a Set Partitioning problem, depending on the specific problem under consideration. Because these problems are well-known NP-hard problems (Wolsey 1998), an "efficient" algorithm is not likely to be found. Efficient here means that the time required to solve the problem is a polynomial function of the size of the problem (Wolsey 1998). Most of the algorithms used for the clustering of individuals problem have been of the heuristic kind, like hierarchical and k-means clustering algorithms, which does not guarantee

finding the optimal clustering alternative. However, many other approaches proved practical for specific problems. Details of this topic are surveyed by Mulvey and Crowder (1979). In this research, we look at the MSCA problem from a mathematical programming point of view, consider three available formulations, and choose one of these formulations–the Clique Partitioning Problem formulation (CPP)– in Chapter 2. We also provide a variation of the CPP formulation. Branch and bound algorithms, which use the chosen formulation, and their performance results in many experiments are presented in Chapter 3. In Chapter 4, an adaptive clustering genetic algorithm (GA) with re-initialization, which also uses the CPP formulation as its encoding scheme, is presented. The GA's performance is discussed and compared to the performance of the branch and bound algorithms.

1.1 Multi-Sample Cluster Analysis (MSCA)

This subsection is an overview of Bozdogan's original work (1981, 1986). Bozdogan uses Model-Selection criteria to introduce Multi-Sample Cluster Analysis (MSCA), the act of clustering samples, as an alternative to Multiple Comparison Procedures (MCPs) in multi-sample data analysis.

In classical statistics, the Analysis of Variance (ANOVA) is used for comparing two or more univariate samples, and the Multivariate Analysis of Variance (MANOVA) is used for comparing multivariate samples. MCPs are based on these analyses. However, Bozdogan argued that the ANOVA and MANOVA analyses are not informative. Hence, he introduced MSCA as a useful procedure to compare all possible clustering alternatives using efficient combinato-

rial algorithms rather than making an arbitrary choice among the clustering alternatives.

One of the most controversial aspects of the classical statistics approach is to arbitrarily fix the level of significance α at $1\%, 5\%$, or 10% prior to the test. Another problem in classical statistics is performing many pair-wise tests, which increases the probability of rejecting at least one null hypothesis when it is actually true. Many scholars tried to solve this multiplicity problem by using various methods to adjust the significance level. However, it is still unclear which of these methods works best. To determine the required number of pair-wise tests, let K be the number of samples. In MCPs, $\binom{K}{2} = \frac{K(K-1)}{2}$ tests are required to perform all pair-wise comparisons among the K groups. However, MCPs can not handle hypothesis testing for more than two samples and must be modified accordingly. Despite all of these problems, MCPs are the second most frequently applied type of statistical methods (Mead and Pike 1975). These and other problems with MCPs caused Hsu to write, "If they rank second in frequency of use, they rank perhaps first in frequency of abuse" (Hsu 1996).

The problem that MCPs try to solve can be looked at as clustering means, groups, samples, or treatments. Plackett, in his discussion of the review paper by O'Neill and Wetherill, was the first to suggest the use of cluster analysis in place of an MCP (O'Neill and Wetherill 1971). Later, others attempted to use cluster analysis in similar applications. However, MSCA, which is also called K-Group Classification or K-Sample Cluster Analysis, is new and different radical approach. In this new approach model selection criteria is used to choose the best clustering alternative. The model selection criteria do not

include any arbitrary choice. Instead, they achieve parameter parsimony by adapting themselves to the number of parameters estimated in the model and adjusting the level of significance accordingly. These criteria are described in detail in Section 1.2.

In MSCA a collection of groups, samples, profiles, or treatments are clustered into homogeneous subsets. This problem is more complicated than clustering individuals or objects in single-sample cases.

Following Bozdogan (1981, 1986), suppose each object or observation has p response measures (dependent variables) in all K groups, samples, or factor levels. Let

$$D(n \times p) = \begin{bmatrix} D_1 \\ D_2 \\ : \\ D_K \end{bmatrix} \begin{matrix} (n_1 \times p) \\ (n_2 \times p) \\ : \\ (n_K \times p) \end{matrix} \tag{1.1}$$

be a data matrix of K groups or samples, where $D_g(n_g \times p)$ is the matrix of the gth group's observations, $g = 1, 2, .., K$, and $n = \sum_{g=1}^{K} n_g$. In MSCA, we try to partition the K samples into k homogeneous clusters where $k \leq K$ is to be determined. Researchers try to choose the smallest possible k that is consistent with the data because this choice will give a robust test statistic, will achieve the desired parsimonious grouping of samples, and will reduce the dimensionality of the multi-sample data set.

The first step in MSCA is to generate all possible clustering alternatives using efficient combinatorial algorithms. Next, an information criterion is computed for all different groupings without making an arbitrary choice among

the clustering alternatives. Then, the clustering alternative with the minimum information criterion value is chosen.

Duran and Odell (1974) found the following formula to determine the number of ways to cluster K samples into k clusters where $k \leq K$ such that none of the k clusters is empty. This is given by

$$\sum_{g=0}^{k} (-1)^g \begin{bmatrix} k \\ g \end{bmatrix} (k-g)^K. \tag{1.2}$$

However, the order of the k clusters is irrelevant. Hence, the total number of clustering alternatives of K samples into k clusters, which is called the Stirling Number of the Second Kind, is

$$w = S(K,k) = \frac{1}{k!} \sum_{g=0}^{k} (-1)^g \begin{bmatrix} k \\ g \end{bmatrix} (k-g)^K. \tag{1.3}$$

On the other hand, if the number of clusters k is unknown, then the total number of clustering alternatives is

$$\sum_{k=1}^{K} S(K,k). \tag{1.4}$$

To show the difficulty of the MSCA problem, we include Table 1.1 which shows the total number of clustering alternatives for a number of K values along with 2^K (exponential expression of K). This table shows that for a large number of samples, it is prohibitive to try to enumerate all possible solutions. In Chapter 2, we show how some researchers have tried to use practical algorithms to find the best solution without enumerating all possible solutions. In Chapters 3 and 4, we present algorithms specifically for MSCA

Table 1.1: The size of the MSCA problem.

K	Exponential	Stirling
1	2	1
2	4	2
3	8	5
4	16	15
5	32	52
6	64	203
7	128	877
8	256	4140
9	512	21147
10	1024	115980
11	2048	678570
12	4096	4213600
13	8192	27644000
14	16384	1.91×10^8
15	32768	1.38×10^9
16	65536	1.05×10^{10}
17	131072	8.29×10^{10}
18	262144	6.82×10^{11}
19	524288	5.83×10^{12}
20	1048576	5.17×10^{13}
40	1.09951×10^{12}	1.5745×10^{35}
60	1.15292×10^{18}	9.77×10^{59}

using information criteria and the CPP formulation.

1.2 Information Criteria

The information criteria optimization approach for statistical model selection was first introduced by Akaike (1973). This pioneering work joined likelihood theory and information theory to produce a significant and straightforward approach for statistical model selection. Later, many other information criteria were introduced, including CAIC, SBC, and ICOMP. Each information

criterion has its own advocates. However, comparing or recommending any of these information criteria is beyond the scope of this study. Nevertheless, we will provide the users of these information criteria with algorithms to solve the MSCA problem efficiently.

1.2.1 General Structure

The general structure of any information criterion consists of two parts as follows:

$$\text{Information Criterion (Model)} = \text{Lack of Fit term} + \text{Penalty term}. \quad (1.5)$$

The input to the information criterion is the statistical model considered. The lack of fit term's value shows how well the model fits the data. This term is negative two times the log of the maximized likelihood function, as follows:

$$\text{Lack of Fit term} \;=\; -2 \times \log(\text{Maximized Likelihood Function}).$$

This expression must be derived for each model. The expressions for the MSCA problem were derived by Bozdogan (1981, 1986, 2004) and are presented in Subsections 1.2.6 and 1.2.7.

All information criteria have the same lack of fit term. The penalty term expression depends on the selected information criterion and is presented in Subsections 1.2.2-1.2.5. Complex models, which have an unnecessarily large

number of parameters, will have large values for the penalty term and hence are less likely to be selected by the information criterion.

1.2.2 AIC

The Akaike Information Criterion, or simply AIC, is the first information criterion (Akaike 1973). AIC estimates the entropy or the Kullback-Liebler information asymptotically. This criterion adds negative two times the log of the maximized likelihood function and positive two times the dimension of the model considered. It is defined as

$$AIC = -2\log(\text{Maximized Likelihood}) + 2m. \tag{1.6}$$

Hence, AIC penalizes only the number of the parameters in the model under consideration. However, AIC does not consider other complexity factors like the collinearity between the parameters in the model such as in the regression case.

1.2.3 CAIC

The Consistent Akaike Information Criterion, or $CAIC$, improves upon AIC by including the effect of the sample size (Bozdogan 1987). $CAIC$ is derived in the same way as AIC. However, it was extended to make the criterion consistent (i.e., to asymptotically choose the correct model as the sample size $n \to \infty$). Therefore, $CAIC$ multiplies the number of parameters by the log of

the sample size plus one as follows:

$$CAIC = -2\log(\text{Maximized Likelihood}) + (\log(n) + 1)m. \tag{1.7}$$

Because $CAIC$ has a generally larger penalty term, it is expected to choose lower dimensional models than AIC does.

1.2.4 SBC

Surprisingly, if we use a different approach, we can derive a criterion that is very similar to $CAIC$. Schwarz (1978) followed the Bayesian approach to statistical analysis to derive the Schwarz Bayesian Criterion, or SBC. However, this approach leads to the same lack of fit term as in AIC and $CAIC$ but to a slightly different penalty term than in $CAIC$, as follows:

$$SBC = -2\log(\text{Maximized Likelihood}) + \log(n)m. \tag{1.8}$$

This criterion leads asymptotically to simpler models than AIC. However, $CAIC$ has a greater penalty term and is expected to choose simpler models than SBC does.

1.2.5 ICOMP

Bozdogan (1988, 1990, 1994, 2004) introduced the Information Complexity information criterion (ICOMP). This criterion introduces the information complexity theory (van Emden 1971) into the statistical model selection criterion. ICOMP takes into consideration many important factors in statistical model

selection, which have not been considered by previous information criteria, including the important interaction (collinearity) between the model parameters. The general structure of $ICOMP$ is as follows:

$ICOMP(Model)$ = Lack of Fit+Lack of Parsimony+Profusion of Complexity.

The first two terms in this structure are the same terms as in the previous criteria. Lack of Parsimony is the same penalty term as in the previously described criteria. It penalizes the number of parameters in the model. Profusion of Complexity term is the new addition to the field of information criteria. It extends the penalty term beyond just the number of parameters to also consider the correlations between the parameters estimates. This extension gives more value to the penalty term. The general expression of $ICOMP$ is as follows.

$$ICOMP = -2\log(\text{Maximized Likelihood}) + 2C_1(\widehat{F}^{-1}), \tag{1.9}$$

where

$$C_1(\widehat{F}^{-1}) = \frac{s}{2}\log(\frac{tr(\widehat{F}^{-1})}{s}) - \frac{1}{2}\log\left|\widehat{F}^{-1}\right|$$

is the entropic complexity of the estimated inverse Fisher information matrix $\widehat{F}^{-1}$ (IFIM) (also known as the Cramer-Rao lower bound matrix), and where s is the rank of IFIM, and $tr(\cdot)$ and $|\cdot|$ mean the trace and the determinant of IFIM, respectively.

For a detailed description of this criterion, see Bozdogan (1988, 1990, 1994, 2004).

1.2.6 MANOVA Model

Two models can be used in MSCA: the varying means and common covariances model, also called the MANOVA model, and the varying means and varying covariances model, thereafter called the varying model. In this study, we compute the information criterion values for these two models and choose the one with the minimum value in our algorithms as the best choice.

Here, we present the expressions of AIC and $ICOMP$ for the MANOVA model. The AIC (from Bozdogan (1986)) and ICOMP (from Bozdogan (2004)) formulas for this model are as follows:

$$AIC(\{\mu_g, \Sigma\}) = np\log(2\pi) + n\log\left|n^{-1}W\right| + np + 2\left[kp + \frac{p(p+1)}{2}\right], \quad (1.10)$$

where W is the within-groups sum of squares and products (SSP) matrix, and p is the number of variables, and where

$$W = \sum_{g=1}^{k} A_g,$$

$$A_g = \sum_{i=1}^{n_g} (d_{gi} - \overline{d}_g)(d_{gi} - \overline{d}_g)'.$$

$$ICOMP(\{\mu_g, \Sigma\}) = np\log(2\pi) + n\log\left|n^{-1}W\right| + np + 2C_1(\widehat{F}^{-1}), \quad (1.11)$$

where

$$\begin{aligned} C_1(\widehat{F}^{-1}) &= \frac{kp + \frac{p(p+1)}{2}}{2}\log\left[\frac{tr(Q^{-1})tr(\widehat{\Sigma}) + \frac{1}{2}tr(\widehat{\Sigma}^2) + \frac{1}{2}(tr(\widehat{\Sigma}))^2 + \sum_j \widehat{\sigma}_{jj}^2}{kp + \frac{p(p+1)}{2}}\right] \\ &\quad -\frac{p}{2}\log\left|Q^{-1}\right| - (\frac{k+p+1}{2})\log\left|\widehat{\Sigma}\right| - \frac{p(p-1)}{4}\log(2) \end{aligned}$$

, and

$$Q = \begin{bmatrix} \frac{n_1}{n} & 0 & .. & 0 \\ 0 & \frac{n_2}{n} & & : \\ : & : & & 0 \\ 0 & 0 & .. & \frac{n_k}{n} \end{bmatrix},$$

$$\widehat{\Sigma} = \frac{1}{n}W.$$

However, the simplified lack of fit term, after dropping all constants, as in Bozdogan (1986) is as follows:

$$n\log|W|.$$

1.2.7 Varying Means and Varying Covariances Model

Here, we present the AIC and $ICOMP$ expressions for the varying means and varying covariances model (Bozdogan 1986 and 2004).

$$AIC(\{\mu_g, \Sigma_g\}) = np\log(2\pi) + \sum_{g=1}^{k} n_g \log\left|n_g^{-1}A_g\right| + np + 2\left[kp + \frac{kp(p+1)}{2}\right].$$

$$ICOMP(\{\mu_g, \Sigma_g\}) = np\log(2\pi) + \sum_{g=1}^{k} n_g \log\left|n_g^{-1}A_g\right| + np + 2C_1(\widehat{F}^{-1}),$$

where

$$\begin{aligned} C_1(\widehat{F}^{-1}) &= \frac{kp + \frac{kp(p+1)}{2}}{2} \times \\ &\log\left[\frac{\sum_{g=1}^{k}\left(tr(Q^{-1})tr(\widehat{\Sigma}_g) + \frac{1}{2}tr(\widehat{\Sigma_g}^2) + \frac{1}{2}(tr(\widehat{\Sigma_g}))^2 + \sum_j \widehat{\sigma_g}_{jj}^2\right)}{kp + \frac{Kp(p+1)}{2}}\right] \\ &-\frac{p}{2}\log\left|Q^{-1}\right| - (\frac{k+p+1}{2})\sum_{g=1}^{k}\log\left|\widehat{\Sigma_g}\right| - \frac{p(p-1)}{4}\log(2), \end{aligned}$$

and where

$$\widehat{\Sigma_g} = \frac{1}{n_g}A_g.$$

However, the simplified lack of fit term as in Bozdogan (1986) is:

$$\sum_{g=1}^{k} n_g \log |A_g| \,.$$

1.2.8 The Monotonic Conditions

An important property of the information criteria is that they consist of two parts: the lack of fit part and the penalty part. These two parts have opposite monotonic proportion with the number of parameters in the statistical model (Bao, Bozdogan, Chatpattananan, and Gilbert 2005). As the number of parameters is increased the lack of fit part's value decreases and the penalty part's value increases. In the context of the MSCA problem, this means that when the number of clusters increases the lack of fit part's value decreases and the penalty part's value increases. This important property is exploited in our branch and bound algorithms as will be shown in Chapter 3.

The lack of fit part's value for the MSCA problem by definition decreases with the increase in the number of clusters. The penalty part's value of AIC, CAIC, and SBC increases linearly with the increase in the number of clusters.

The monotonic condition of the penalty part of ICOMP is shown in Bozdogan and Haughton (1998). Even if the monotonic condition does not hold for the penalty part of an information criterion for the MSCA problem we can assume that the monotonic condition holds and use the branch and bound algorithms as heuristics that do not guarantee finding the optimal solution. We also develop a genetic algorithm that can solve the MSCA problem without requiring any restriction, like the monotonic conditions, on the information criteria as will be discussed in Chapter 4.

1.3 International Market Segmentation

The concept of market segmentation has been described by many researchers, including Lilien and Rangaswamy (1998). Consumers in any market differ in their needs, preferences, and many other characteristics. In light of this fact, it is ineffective for businesses to use the same marketing strategies to appeal to all consumers. On the other hand, developing a specific marketing strategies for every consumer is inefficient. A more reasonable option is to do market segmentation. In market segmentation studies, the market is divided into a finite number of groups or segments of consumers. The consumers within each segment should be as homogeneous (alike) as possible with respect to the considered characteristics. In contrast, the segments themselves should be as heterogenous (different) as possible.

Market segmentation is generally followed by the targeting step. In this step, the segment or segments that are more likely to be more profitable to the business firm are identified and called the target market. All marketing efforts are then designed to appeal to this target market. This approach should be the most efficient for any business firm. For a review of this area, see Beane and Ennis (1987). Cluster analysis has been widely used to perform market segmentation (Punj and Stewart 1983).

One of the most important applications of Multi-Sample Cluster Analysis is the area of international (or global) market segmentation. In this area, marketing researchers help international firms try to divide the international market into meaningful segments to help them understand the market, design the best marketing efforts, target promising segments, and position their

products and/or services accordingly.

A detailed review of this area of research and the related articles was done by Steenkamp and Hofstede (2002). Recently, this area of research has received more attention. The developments that led to the phenomenon of globalization have forced companies to need international market segmentation. Though segmentation can be done on the individual consumer level, this strategy is not cost effective (Steenkamp and Hofstede 2002). Another extreme option is to do segmentation at the country-level. Although this approach is widely used, it ignores differences that are present within each country. A possible reason for using this approach is the ease of getting published macroeconomic data. However, macroeconomic data includes only the averages of individuals' data and ignores the variances. A better approach is to use samples taken from regions within the countries as the objects to be segmented (Steenkamp and Hofstede 2002).

Madsen and Askegaard (1998) provide a direct example of international market segmentation. In their article, 79 regions in 15 European countries are classified into 12 clusters based on a sample of $20,000$ observations on 138 questions related to food culture. These questions were collapsed into 41 variables. The goal was to develop a map of the European food culture. Unfortunately, the authors used only the averages of the 79 samples in the SPSS hierarchical cluster analysis methods. They admit that using only the averages is an important limitation. MSCA would have considered the whole samples. Unfortunately, there are currently no algorithms that can solve this problem using the MSCA. This could be one of the reasons that led the researchers to use their approach.

A recent example of international market segmentation studies is the work of Bijmolt et al. (2004). These authors used a new approach to simultaneously find the consumer-level and the country-level segments for an international financial services market. They had a sample of $1,000$ observations on the ownership of 8 financial products from each of 15 European countries. Their approach involved a sophisticated mixture model and an adaptive version of the EM algorithm which is a heuristic and does not guarantee finding the optimal solution. They had to repeat the algorithm for 10 different random initialization to find the best possible solution. They repeated their approach for all possible numbers of clusters and scored $CAIC$ for the final results. Another possible approach would be to apply the MSCA directly to the 15 samples. This approach would use $CAIC$ directly to find the optimal clustering alternative. Unfortunately, there are currently no algorithms that use this approach, which could be one of the reasons for using the EM algorithm by these researchers.

A similar problem to the MSCA problem is the Multi-Sample Cluster-Wise Regression problem. In the Multi-Sample Cluster-Wise Regression problem, the samples that give similar regression coefficients to the variables are joined together. In this problem, all of the variables are used, and no subset selection is done. An example of this type of research is the study by Hofstede et al. (2002). In this work, the authors used a Bayesian approach [Markov Chain Monte Carlo (MCMC) and Gibbs sampling] to segment samples from 120 regions in 7 European countries. They compared four models, including the spatial independent model, which is exactly the same as Multi-Sample Cluster-Wise Regression model. They had to repeat the same procedure for

each possible number of clusters and choose the model with the maximum Bayes factor. We can use MSCA with the information criteria expression for the regression case to do the same task. Again, no algorithms are currently available to solve this problem by using the MSCA. This could also be one of the reasons that led the researchers to use the Bayesian approach.

There are a lot of other examples of opportunities to apply the MSCA in international market segmentation (Ronen and Kraut 1977; Hofstede 1976; Sirota and Greenwood 1971; Cui and Liu 2000; Kahle 1986). Some examples of country-level segmentation on averages of individual data are by Ronen and Shenkar (1985), Kale (1995), Kumar et al. (1994), Steenkamp (2001), and Vandermerwe and L'Huillier (1989). Other Multi-Sample Cluster-Wise Regression examples include Hofstede, Frenkel Ter et al. (1999) and Mittal et al. (2004). Some surveys of cluster analysis in marketing are by Green, Paul E. et al. (1967) and Punj and Stewart (1983).

Chapter 2

MSCA Formulations and Approaches

The general approach to solve any problem involves building a model that represents this problem. All three models that have been developed for the cluster analysis problems are mathematical models. All of the research we read on mathematically modeling cluster analysis problems was on clustering single observations. Here, we adapt these mathematical models for the MSCA problem and discuss the related work on branch and bound algorithms. Here, we review only the branch and bound algorithms because this is the approach we used to solve the MSCA problem in Chapter 3. We discuss the Clique Partitioning Problem, CPP, formulation for the MSCA problem. For detailed surveys of all formulations and algorithms, see Rao (1971), Hansen and Jaumard (1997), and Xu and Wunsch (2005).

2.1 Uncapacitated Facility Location Formulation

In the Uncapacitated Facility Location (UFL) problem, we have a number of potential facility locations and a number of customer regions. We need to decide which locations to open and which customer regions to assign to each open location. There is no capacity limit on the number of customer regions that we can assign to each location. The original UFL problem had a linear objective function and is considered an NP-hard problem (Wolsey 1998). This problem has been solved by many approaches including the cutting planes approach (Wolsey 1998). Here, we present the same formulation for the MSCA problem using the information criteria as the objective function.

2.1.1 Formulation

Let

$$x_{ij} = \left\{ \begin{array}{l} 1 \text{ if the } i\text{th sample is assigned to the } j\text{th cluster.} \\ 0 \text{ otherwise.} \end{array} \right\} 1 \leq i, j \leq K,$$

$$y_j = \left\{ \begin{array}{l} 1 \text{ if the } j\text{th cluster is created.} \\ 0 \text{ otherwise.} \end{array} \right\} 1 \leq j \leq K.$$

The following is the formula for the matrix of total sums of squares and products for a specific clustering alternative:

$$W = \sum_{j=1}^{K} \sum_{i=1}^{K} x_{ij} \sum_{l=1}^{n_i} (d_{il} - \overline{d}_j)(d_{il} - \overline{d}_j)' = \sum_{j=1}^{K} A_j. \tag{2.1}$$

The following is the formula for the mean of each cluster, assuming it is zero for empty clusters:

$$\overline{d}_j = \frac{\sum_{i=1}^{K} x_{ij} \sum_{l=1}^{n_i} d_{il}}{\sum_{i=1}^{K} x_{ij} n_i}. \tag{2.2}$$

Now we present the formulation for the MSCA problem, under the MANOVA model, using AIC as the objective function but excluding the constants.

$\min \quad n \log |W| + 2pk$

$s.t.$

1. $\sum_{j=1}^{K} x_{ij} = 1, \forall i = 1, .., K$
2. $\sum_{i=1}^{K} x_{ij} \leq K y_j, \forall j = 1, .., K$
3. $\sum_{i=1}^{K} y_j - k = 0$

where K is the number of samples, k is the number of clusters of samples or groups, p is the dimension of the observations, and n is the total number of observations. This formulation has $K^2 + K + 1$ decision variables and $2K + 1$ constraints.

This is the basic formulation, and researchers have developed several variations of it to deal with many different issues. The most important issue in the UFL formulation is the redundancy problem, meaning that the same clustering alternative has more than one representation. The reason for this problem

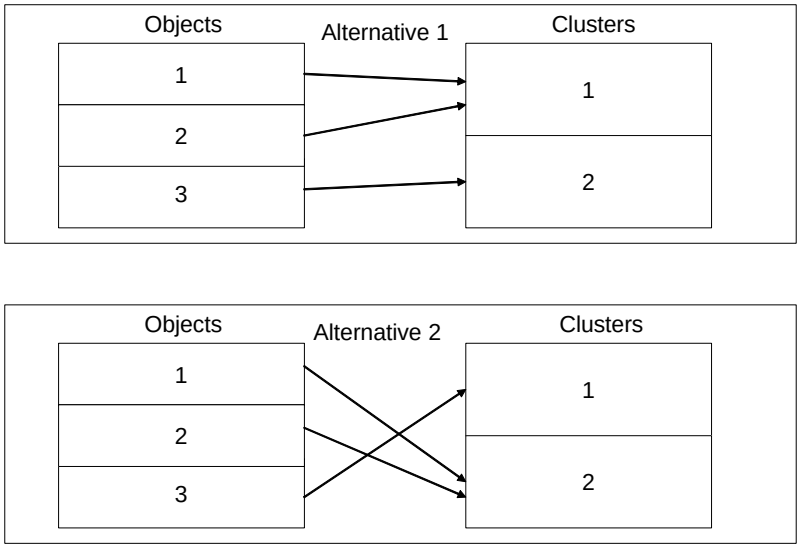

Figure 2.1: Two redundant alternatives that join the same objects together.

is that we can interchange the labels of the clusters (j) and still get the same clustering alternative and the same value for the objective function. A simple example that illustrates this problem is in Figure 2.1.

2.1.2 Algorithms

The UFL formulation was first suggested for cluster analysis by Vinod (1969). Since then this formulation has been used in many algorithms. All of the algorithms that use this formulation fix the number of clusters and use an objective function that allows this restriction.

The first branch and bound algorithm to solve the cluster analysis problem was developed by Koontz et al. (1975). These authors used basic branching by the best assignment of a free object to a cluster and basic bounding using the properties of the $WGSS$, which include the recursion property. However, their

algorithm does not deal with the redundancy problem. Diehr (1985) improves on this algorithm by starting with a heuristic and avoiding the redundancy problem by assigning objects to the empty cluster with the lowest index.

Klein and Aronson (1991) used the linear sum of dissimilarity measures between objects as the objective function. This objective function will always lead to the maximum allowed number of clusters as the optimal solution. Therefore, they used the UFL formulation to fix the number of clusters. However, the CPP variables were used to linearize the objective function. Unfortunately, their way of handling the redundancy problem were incomplete. The branch and bound algorithm was preceded by a heuristic to get a good initial upper bound. The branching strategies were sequential. The bounding strategies depended on the linear objective function.

The most sophisticated approach for solving the cluster analysis problem was by du Merle et al. (2000). These authors were the first to show that the Lagrangian relaxation of the UFL formulation is equivalent to the linear relaxation of the Set Partitioning formulation. They used the branch and bound approach within a collection of different techniques. These techniques included the Lagrangian relaxation, cutting planes, the neighborhood search heuristic, and quadratic programming. These techniques were needed because the authors used the $WGSS$ objective function, which is fractional and quadratic. The authors claim that their work was the first to find the optimal solution for the IRIS benchmark data set (Fisher 1936). Unfortunately, their approach had the limitation of fixing the number of clusters, and hence, it must be repeated for every possible number of clusters.

A heuristic-based branch and bound algorithm was developed by Brusco

(2003). This algorithm used Klein's (1991) formulation and the same linear sum of pair-wise distances as the objective function. Improved bounds were used, but they still depended on the linear objective function. He used the exchange heuristic, which is a neighborhood search heuristic in which objects are moved or exchanged between clusters. He also developed another branch and bound algorithm that uses the $WGSS$ as the objective function (Brusco 2006). This algorithm is applied to a subset of the objects and then repeated each time a new object is added. In addition, the algorithm begins with a heuristic that reorders the objects by placing the closest neighbors at the opposite ends of the list. This will ensure that the algorithm will run smoothly when new objects are added and that it will not need to change the original solution significantly. The bounds used in this algorithm are extensions of the work of Koontz et al. (1975).

2.2 Set Partitioning Formulation

The Set Partitioning problem involves dividing a set of objects into mutually exclusive subsets. This problem is a special case of the set covering problem, which is a well known NP-hard problem (Wolsey 1998). The Set Partitioning formulation was first applied to the cluster analysis problems by Rao (1971). This author suggested solving the cluster analysis problems using the general approaches used to solve the Set Partitioning problem, as described by Garfinkel, R. S. and Nemhauser, G. L. (1969).

2.2.1 Formulation

The number of possible assignments to a single cluster is $N = \sum_{i=0}^{K} \binom{K}{i}$. y_j is a binary variable that equals 1 if a cluster is assigned the jth possible assignment alternative. A_j is the matrix of the sums of squares and products of the jth possible assignment alternative for a cluster. The following is our formulation:

$\min\ AIC = n \log(\left|\sum_{j=1}^{N} A_j y_j = A_1 y_1 + A_2 y_2 + .. + A_N y_N\right|) + 2pk$

s.t.

1. $\sum_{j=1}^{N} y_j - k = 0$, number of clusters constraint.

2. $By = 1$, partitioning constraints.

B is a $K \times N$ matrix where each column B_j of B represents a possible assignment to a cluster (i.e., $b_{ij} = 1$ if sample i is in the assignment j and $b_{ij} = 0$ otherwise).

Example: $K = 3$

$\min n \log(|A_1 y_1 + A_2 y_2 + A_3 y_3 + A_4 y_4 + A_5 y_5 + A_6 y_6 + A_7 y_7 + A_8 y_8|) + 2pk$

s.t.

$y_1 + y_2 + y_3 + y_4 + y_5 + y_6 + y_7 + y_8 = k$

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

This formulation has many more variables, $\sum_{i=0}^{K} \binom{K}{i} + 1$, but less constraints, $K + 1$, than the UFL formulation. For example, for $K = 10$, we will have 1025 decision variables while using the UFL formulation we will have only 112 decision variables.

2.2.2 Algorithms

The work on the Set Partitioning formulation involved using the decomposition and column generation approaches (Johnson, Mehrotra, and Nemhauser 1993; Mehrotra and Trick 1998). No branch and bound algorithms used this formulation for the cluster analysis problem.

2.3 The Clique Partitioning Problem (CPP) Formulation

The Clique Partitioning Problem formulation was introduced by Wakabayashi (1986). This problem is to partition a complete graph of nodes, where all

nodes are connected to each other by edges, into subgraphs with the minimum linear sum of the edges' weights within each subgraph. This problem is exactly the same as the cluster analysis problem, with the nodes being the objects to be clustered and the edges being the pair-wise dissimilarity measures.

2.3.1 Formulation

The problem is formulated as follows.

Let

$$x_{ij} = \left\{ \begin{array}{l} 1 \text{ if nodes } i \text{ and } j \text{ are in the same partition.} \\ 0 \text{ otherwise.} \end{array} \right\},$$

$$1 \leq i < j \leq K,$$

where K is the number of nodes in the graph. The number of decision variables is

$$\binom{K}{2} = \frac{K(K-1)}{2}.$$

These decision variables can be represented by a half a matrix as

$$x = \begin{bmatrix} x_{12} & x_{13} & .. & x_{1K} \\ & x_{23} & .. & x_{2K} \\ & & .. & : \\ & & & x_{(K-1)K} \end{bmatrix}.$$

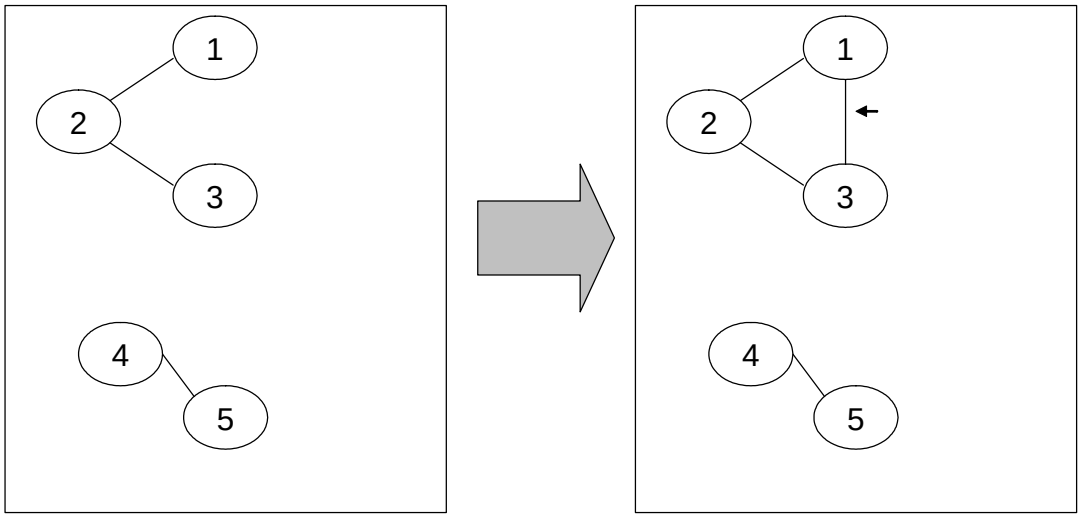

Figure 2.2: Triangle Constraint: Node 2 is connected to nodes 1 and 3, which forces nodes 1 and 3 to be connected to each other.

The only constraints are the triangle constraints for all $1 \leq i < j < h \leq K$:

$$\begin{aligned} x_{ij} + x_{jh} &\leq 1 + x_{ih} \\ x_{ij} + x_{ih} &\leq 1 + x_{jh} \\ x_{ih} + x_{jh} &\leq 1 + x_{ij}. \end{aligned}$$

These constraints require that if a node is connected by edges to two different nodes then these two nodes must be connected by an edge (belong to the same subgraph). A simple example that shows how these constraints work is presented in Figure 2.2.

The number of constraints is:

$$3\binom{K}{3} = 3\frac{K!}{3!(K-3)!} = \frac{1}{2}K(K-1)(K-2).$$

Hence, the CPP formulation has fewer variables but more constraints than the Uncapacitated Facility Location and the Set Partitioning formulations.

The objective function to be minimized is $\sum_{i=1}^{K-1}\sum_{j=i+1}^{K}\omega_{ij}x_{ij}$. If the edges' weights are all positive or all negative, finding the optimal solution is trivial. That is, depending on whether the the weights are all positive or all negative, respectively, each node is either in its own cluster (all x's are zeros) or all nodes are in one cluster (all x's are ones). However, if the weights include positive and negative values then the problem is NP-hard (Wakabayashi 1986). Wakabayashi proved this fact by transforming the well-known NP-hard Maximum Cut problem to the CPP problem.

We chose to use the CPP formulation to solve the MSCA problem because it does not have the redundancy problem, unlike the UFL formulation, and it has a few number of variables, unlike the Set Partitioning formulation. The large number of constraints in the CPP formulation can be handled smoothly in the branch and bound algorithms as we will show in Chapter 3. In the following, we present the MSCA objective function using the CPP variables.

The Mean of a Cluster

We need the mean of each cluster for the computation of the W matrix. The formula of the mean of each cluster in terms of the binary decision variables

is as follows:

$$\overline{d}_i = \frac{\left(\sum_{l=1}^{n_i} d_{il} + \sum_{j=i+1}^{K} x_{ij} \sum_{l=1}^{n_j} d_{jl}\right) \prod_{p=1}^{i-1}(1 - x_{pi})}{n_i + \sum_{j=i+1}^{K} x_{ij} n_j}, i = 1, .., K. \qquad (2.3)$$

This expression is nonzero for the clusters that are not empty. The clusters are ordered by the smallest sample index they contain. We use x_{ij} to determine if two samples are in the same cluster. If a sample is in the same cluster as a sample with a smaller index, $\prod_{p=1}^{i-1}(1 - x_{pi})$ is used to skip this sample row in the decision variables' half matrix . Now we use $\overline{d}_i$ to compute the matrix W.

The Matrix of Sum of Squares and Products

Using the binary decision variables and the mean of each cluster, we can compute the matrix W as follows:

$$W = \sum_{i=1}^{K} \prod_{p=1}^{i-1}(1-x_{pi}) \left(\sum_{l=1}^{n_i} (d_{il} - \overline{d}_i)(d_{il} - \overline{d}_i)' + \sum_{j=i+1}^{K} x_{ij} \sum_{l=1}^{n_j} (d_{jl} - \overline{d}_i)(d_{jl} - \overline{d}_i)' \right). \qquad (2.4)$$

This expression reuses x_{ij} and $\prod_{p=1}^{i-1}(1 - x_{pi})$ as previously described in this subsection for the mean expression. The only extra term needed to evaluate AIC is the number of clusters in a clustering alternative.

Number of Clusters

To find the number of clusters k in a clustering alternative, we use the decision variables' values as:

$$k = K - \sum_{i=1}^{K-1} \prod_{p=1}^{i-1} (1 - x_{pi}) \sum_{j=i+1}^{K} x_{ij}. \tag{2.5}$$

This expression is equal to the maximum possible number of clusters K minus the number of times we join a sample to an already formed cluster. We use $\prod_{p=1}^{i-1}(1-x_{pi})$ to skip the decision variables that have been set to 1 by a triangle constraint with a lower index sample. This is a non-linear formulation. We present a linear formulation in Subsection 2.3.3.

2.3.2 Algorithms

The CPP problem was solved mostly by cutting plane algorithms (Wakabayashi 1986; Grotschel and Wakabayashi 1989). These approaches solve the Linear Programming (LP) relaxation of the problem, where the x_{ij} variables can assume continuous values between 0 and 1, without the triangle constraints. If the LP relaxation solution has x_{ij} values that are not 0 or 1 (not integral), they add a subset of the triangle constraints and other facet-defining inequalities.

Since this work, there have been only two attempts to use the branch and bound approach to solve the CPP with the linear objective function. Dorndorf and Pesch (1994) developed a heuristic-based branch and bound algorithm. In their work, the problem is first solved by the Ejection Chain heuristic, which is an advanced Tabu search algorithm that saves the best sequences of moves to reuse them to get out of any local optimum. Then, the branch and bound

algorithm starts by branching on the best x_{ij} variables among all free x_{ij} variables according to the solution from the Ejection Chain algorithms. The bounds are based on the linear objective function and the negative and positive dissimilarity coefficients. We need to note that searching among all free x_{ij} variables, which is already a lot of work, includes the x_{ij} variables that join the free samples to the same clusters, which is redundant.

The second work was by Palubeckis (1997), who developed another heuristic-based branch and bound algorithm. However, this algorithm uses some polyhedral results. This approach starts by using agglomerative clustering and then iterative clustering (like K-means) heuristic methods. However, it allows for deleting or creating clusters. The bounding strategy is based on a transformation of the heuristics solution. The branching strategy chooses a node rather than a variable x_{ij} to branch on. This branching strategy is based on the linear objective function, facet computations, and the heuristics solution. However, in only two of the problems that were tested with this approach, the heuristics methods were not enough to find the optimal solution and use of the branch and bound algorithm was required.

2.3.3 Linear Formulation for the Number of Clusters

Many researchers have used the Uncapacitated Facility Location formulation that fixes the number of clusters. However, this formulation includes the misleading notion of labeling the clusters and, hence, admits redundancy unless more constraints and/or methodological techniques are considered as discussed in Section 2.1. A number of researchers studied variations of the CPP problem

by introducing upper and lower limits on the number of objects within each cluster. These limits indirectly control the number of clusters.

Here, we introduce a linear formulation to fix or find the number of clusters in a clustering alternative using the CPP variables. The number of clusters in a clustering alternative is required to compute AIC, $CAIC$, and SBC. As far as we know, no one has used the CPP x_{ij} variables to fix the number of clusters. This restriction fixes the number of clusters to a given number k. Only one previous work introduced a lower limit to the number of clusters. However, it used the spanning tree definition and not the CPP variables (Chopra and Rao 1993). Our approach does not use the definition of the spanning tree and involves the CPP variables, plus additional variables and constraints, as follows:

Let

$$t_{ijh} = \left\{ \begin{array}{l} 1 \text{ if the three nodes } i, j, \text{ and } h \text{ are in the same cluster} \\ 0 \text{ otherwise} \end{array} \right\}$$

$$1 \leq i < j < \ h \leq K$$

be the triangle variables and

$$y_{jh} = \left\{ \begin{array}{l} 1 \text{ if } x_{jh} \text{ is set to 1 by a triangle constraint with} \\ \quad \text{a lower index node } i \text{ where } i < j < h \\ \\ 0 \text{ otherwise} \end{array} \right\}$$

be the redundancy variables.

Then, the CPP constraints will be as follows:

A) For all $i < j < h$

1. $x_{ij} + x_{jh} \leq 1 + x_{ih}$
2. $x_{ij} + x_{ih} \leq 1 + x_{jh}$
3. $x_{ij} + x_{jh} \leq 1 + x_{ih}$
4. $t_{ijh} \leq x_{ij}$
5. $t_{ijh} \leq x_{ih}$
6. $x_{ij} + x_{ih} \leq 1 + t_{ijh}$
7. $t_{ijh} \leq y_{jh}$

B) $y_{jh} \leq \sum_{i=1}^{j-1} t_{ijh}$, for $\forall 1 < j < h$

C) $k = K - \sum_{i=1}^{K-1} \sum_{j=i+1}^{K} x_{ij} + \sum_{j=2}^{K-1} \sum_{h=j+1}^{K} y_{jh}$.

In Group A, there are seven constraints. Constraints A1-A3 are the same triangle constraints as in the original CPP formulation.

Constraint A6 forces the triangle variable t_{ijh} to be set to 1 if the two variables x_{ij} and x_{ih} are set to 1. Constraints A4 and A5 force this triangle variable to be set to 0 if any of these two variables is set to 0. Therefore, t_{ijh} is set to 1 when there is an active triangle relationship between i, j, and h (i.e., $x_{ij} = x_{ih} = x_{jh} = 1$). This is realized when only two of them is set to 1 because the third one will be set to 1 by the triangle constraints.

Constraint A7 forces the redundancy variable y_{jh} to be 1 if the triangle variable t_{ijh} is set to 1. Therefore, we will count only the ones in the row of the decision variables half matrix of the lowest index i in the triangle relationships. Hence, we will not count any variable that was set to 1 by a triangle relationship with a lower index variable.

Constraint B forces the redundancy variable y_{jh} to be set to 0 when all the triangle variables t_{ijh} are set to 0 (i.e., the variable x_{jh} value is not forced to be set to 1 by a triangle constraint with a lower index variable).

Constraint C is the expression for the number of clusters. This expression subtracts the sum of the x_{ij} variables from the maximum number of clusters K and then adds the sum of the redundancy variables. The sum of the x_{ij} variables counts the number of times two samples are joined together. However, this sum includes the x_{ij} variables that were set to 1 by a triangle constraint with a lower index node. Therefore, these are added back by the sum of the redundancy variables.

A simple example that shows how this model works is in the following half matrix of decision variables:

$$x = \begin{bmatrix} [1] & 1 & 1 & 1 & 0 & 0 \\ & [2] & \underline{1} & \underline{1} & 0 & 0 \\ & & [3] & \underline{1} & 0 & 0 \\ & & & [4] & 0 & 0 \\ & & & & [5] & 1 \\ & & & & & [6] \end{bmatrix}.$$

The numbers in brackets shows the node number in the diagonal of the

matrix. This matrix shows two clusters. The first cluster includes nodes 1, 2, 3, and 4. The second cluster includes nodes 5 and 6. The underlined ones are not counted in the model because they are set by a triangle relationship with node 1. There are 4 active triangle relationships:

$$t_{1,2,3} = t_{1,2,4} = t_{1,3,4} = t_{2,3,4} = 1.$$

Hence, there are 3 redundant ones:

$$y_{2,3} = y_{2,4} = y_{3,4} = 1.$$

Therefore, the model will calculate the number of clusters as follows:

$$\begin{aligned} k &= K - \sum_{i=1}^{K-1} \sum_{j=i+1}^{K} x_{ij} + \sum_{j=2}^{K-1} \sum_{h=j+1}^{K} y_{jh} \\ &= 6 - 7 + 3 = 2 \text{ clusters.} \end{aligned}$$

This variation of the CPP problem involved more variables and constraints. We may expect the linear case of the problem to be difficult. However, Grotschel and Wakabayashi (1989) solved the original linear CPP mostly by adding only a subset of the violated triangle constraints. We expect that the same approach could be used for this variation of the problem. However, because the MSCA problem involves a nonlinear objective function, this direction of research is beyond the scope of our study.

Chapter 3

MSCA Branch and Bound Algorithms Using the CPP Formulation

3.1 Introduction

Branch and bound algorithms enumerate all possible solutions to a problem either implicitly or explicitly and at the end find the optimal solution. The explicit enumeration of possible solutions in these algorithms is done through the branching strategies by actually evaluating the objective function values of these solutions. The implicit enumeration of possible solutions is done by bounding strategies on partial solutions that lead to these possible solutions. If a lower bound on the objective function of the completions of a partial solution exceeds the objective function value of some known solution, then it is unnecessary to explicitly enumerate any of the solutions that are completions

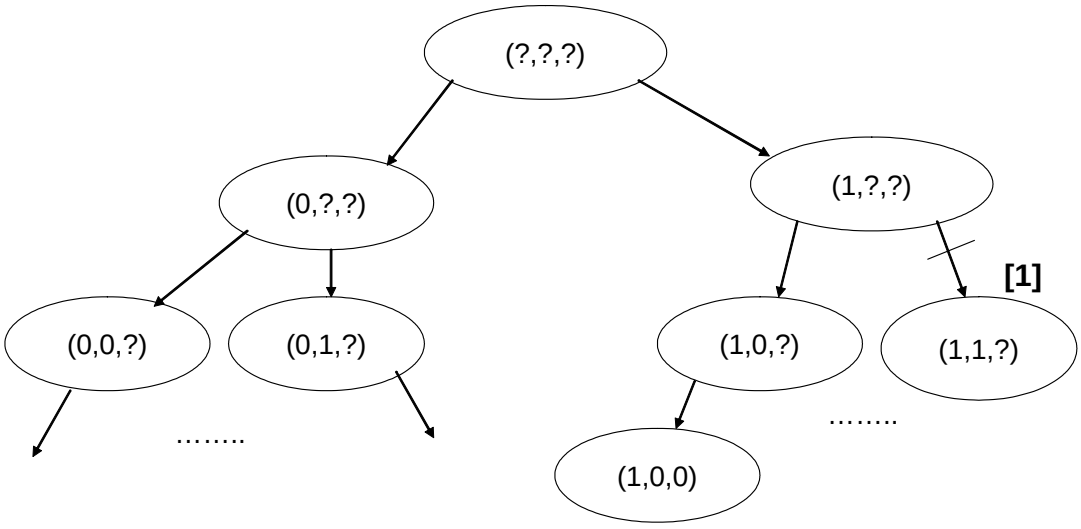

Figure 3.1: A simple branch and bound example.

of this partial solution. Hence these complete solutions should not be enumerated. The first branch and bound algorithm was developed by Trotter and Shetty (1974).

A simple hypothetical example of a branch and bound enumeration tree is presented in Figure 3.1. In this example, we enumerate 3 binary variables x_1, x_2, and x_3, which take the value 0 or 1. Node 1 have been pruned because it meets the bounding conditions. For a detailed description of general branch and bound algorithms, please see Wolsey (1998).

The MSCA problem uses the information criteria as the objective function. Although the information criteria is a nonlinear objective function, they have special properties that enable the development of branching and bounding strategies (Bao, Bozdogan, Chatpattananan, and Gilbert 2005). This is one of the reasons we chose the branch and bound approach. In this chapter, we first present a description of all the considered branching and bounding

strategies. These strategies include a heuristic version of some bounds and a saving technique that aims to save computation time. Then, we develop a complete enumeration algorithm for the MSCA problem. Later, we present a sequence of branch and bound algorithms. Each branch and bound algorithm is an improvement on the previous one by including more branching and/or bounding strategies. Next, we present the performance results of these algorithms and strategies and recommend the use of the best ones. Finally, we discuss some computational remarks, conclusions, and future research.

3.2 Branching Strategies

The branching strategies state how the algorithm progresses through a series of partial solutions to reach complete solutions. We present the following four branching strategies that will be used in our algorithms:

3.2.1 Sequential Branching

The sequential branching strategy changes the values of the x_{ij} variables between 0 and 1 in sequence. It starts by assigning 0 or 1 to all x_{ij} variables and then interchanging the values sequentially to check all possible combinations. It also assigns values implicitly by the triangle constraints. Although this strategy looks naive, it is useful in understanding the MSCA problem.

3.2.2 Adaptive Branching

In any branching strategy, we can choose the decision variable x_{ij} and the value to branch on (0 or 1) by two intelligent steps rather than by sequential

order. The first step is to select a sample from the set of all free samples, which have not been clustered yet. This first step will be explained in the next branching strategy, reordering. The second step is either to choose an already formed partial cluster that the selected free sample will join or to let this free sample start a new partial cluster.

In the adaptive branching strategy, the algorithm branches on the samples sequentially through the indexes of the x_{ij} variables. In the beginning, the first sample always starts a partial cluster. Then, the second sample is either joined to the first sample partial cluster or left to start a new partial cluster. The next samples are then either joined to one of the already formed partial clusters or left to start their own partial clusters. The decision to join a sample to a formed partial cluster or to start a new partial cluster is taken to achieve the minimum information criterion value (i.e., a greedy branching). Whenever we reach a complete solution or prune a partial solution, we backtrack to the immediate parent partial solution (parent node). Figure 3.2 shows the adaptive branching complete tree, where (i, j) means that samples i and j are in the same cluster.

This strategy involves a lot fewer computations than the agglomerative hierarchical method and is expected, when tested later on, to find better solutions.

3.2.3 Reordering

In the adaptive branching strategy, the samples are considered in the order they are stored in the data matrix D. However, the order in which the samples

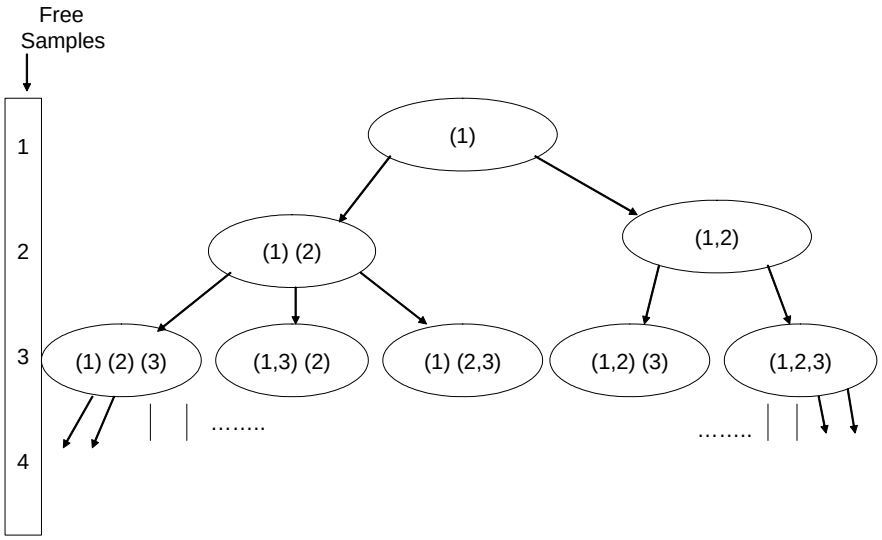

Figure 3.2: Adaptive branching complete tree.

are considered can affect the performance of the algorithm. If the samples that are very similar or very dissimilar are consecutive in consideration, this can quickly lead to the optimal solution. But if the order is not like this, the algorithm can proceed to a solution that is far from the optimal solution. The idea of reordering objects for cluster analysis problems is not new (Brusco 2006). However, here we use the information criteria to reorder the samples.

The reordering strategy should precede the adaptive branching strategy to change the order in which the samples are joined. Though, we discuss many reordering strategies in this subsection, we will implement only the first one. An extensive testing of these methods is the only way to show the superiority of one reordering strategy over the other.

The following is a list of possible reordering methods:

1. Put all samples in one cluster and take one sample out at a time (af-

ter returning the previous sample) and score AIC. Pick the sample with the minimum/maximum AIC score (farthest from/closest to the mean). Based on the choice of picking the sample with the minimum or maximum AIC score we name this reordering strategy as the ascending or descending strategy respectively. This reordering strategy should be performed before running the branch and bound algorithm. There are two advantages to this approach over the next approaches: We pick the sample that is the farthest from/closest to the mean. We save computation time because this reordering strategy is done only once before running the branch and bound algorithm.

2. At each branching step, put the remaining free samples in one cluster and take one free sample at a time and put it in its own cluster and score AIC. We can then pick the free sample with the minimum/maximum AIC value (the farthest from or closest to the mean of the remaining free samples). The farthest choice may lead us to choose an initial sequence of samples that are away from each other and, therefore, form the most clear clusters from the beginning. The closest choice may lead us to forming a cluster in the center of all samples which may be an incorrect cluster.

3. At each branching step, put each of the remaining free samples in its own cluster and all clustered samples in one cluster. Then join one free sample at a time to the cluster of clustered samples and score AIC. We can pick the free sample with the minimum/maximum AIC (the farthest from or closest to the mean of clustered samples). This strategy has

almost the opposite effect of the previous strategy.

4. At each branching step, put the remaining free samples in one cluster and all clustered samples in one cluster. Then take one free sample at a time from the cluster of free samples and join it to the cluster of clustered samples and score AIC. We can pick the free sample with the minimum/maximum AIC value. This strategy shares some aspects of both of the previous two strategies and, therefore, has unclear effects.

3.2.4 Complete Adaptive Branching

In the adaptive branching strategy, we backtrack to the best unchecked sibling node, according to the bounding strategies, in the enumeration tree or to the parent node if no unchecked sibling node is available. Unfortunately, this strategy could trap us in a branch of the enumeration tree that is far from the optimal solution. This can happen because the early branching decisions were not optimal. To avoid this situation, we propose a new branching strategy, which we call the complete adaptive branching strategy. In this strategy, we backtrack to the best node, according to the bounding strategies, in the complete formed tree as long as we do not exceed a given limit on the size of the formed tree. This limit depends on the memory of the computer used. The complete adaptive strategy will lead the enumeration process back to branch from higher-level nodes and, hopefully, will lead to different parts of the enumeration tree that could have the optimal solution. Figure 3.3 shows the difference in the expected performance between the adaptive and complete adaptive branching strategies.

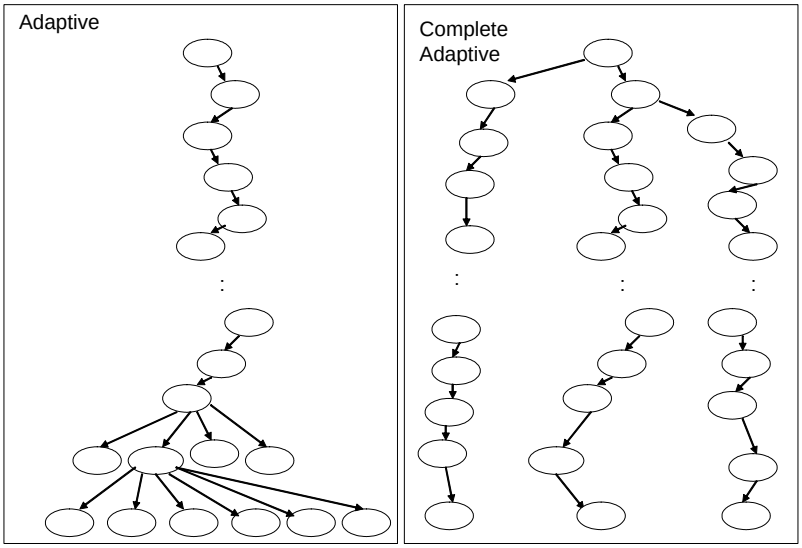

Figure 3.3: Expected performance of adaptive and complete adaptive branching strategies.

3.3 Bounding Strategies

When developing the following bounding strategies we assume we are using the adaptive branching strategy. At any point in the adaptive branching strategy we have k formed partial clusters and f free samples.

3.3.1 Upper Bound

The upper bound is the minimum objective function value of all of the complete solutions that have been found. In all branching and bounding calculations, we update this upper bound whenever possible, especially when computing the lower bounds, as we will discuss in the next subsections.

3.3.2 Extension of Bao et al (2005) Lower Bounds

The following are extensions of the bounds by Bao et al. (2005):

First Lower Bound

A lower bound on any partial solution can be obtained by adding the smallest possible values of the lack of fit part and the penalty part for the complete solutions of this partial solution. These values are found by using the monotonic conditions as follows. The lowest possible value of the lack of fit part is the lack of fit part value after assigning all free samples to their own clusters (i.e., having the maximum number of clusters, $k + f$). The lowest possible value of the penalty part is the penalty part value of keeping only the already-formed k cluster (i.e., assuming the free samples are assigned to these clusters). Following are the computational steps to get this first lower bound:

1. Assign all f free samples to their own clusters to have $k + f$ clusters. Score the lack of fit part for this case.

2. Score the penalty term for the minimum of k clusters.

3. Add the lack of fit part and the penalty part values to get the first lower bound.

Second Lower Bound

The second lower bound is the same as the first lower bound with one exception. In this bound, we use the next lowest value of the lack of fit part. This value is obtained by having $k + f - 1$ clusters (i.e., joining two free samples

together or joining a free sample to an already-formed cluster). We still have to use the penalty part of k clusters because the free samples can always be joined to the already-formed clusters. Following are the computational steps to get this second lower bound:

1. Find the minimum lack of fit part of $k + f - 1$ complete solutions by
   - Joining each free sample one at a time to each formed cluster (This involves checking $f \times k$ complete solutions).
   - Joining every free sample to every other free sample one at a time (This involves checking $\binom{f}{2}$ complete solutions).
2. Use the same penalty part of k clusters.
3. Add the lack of fit term and the penalty term values to get the second lower bound.

Expensive Third Lower Bound

We can get a third lower bound by following the same approach as in the first and second lower bounds. However, this approach is prohibitively expensive because getting the lowest lack of fit part value for $k + f - 2$ clusters, requires checking $(\binom{f}{2} \times k^2) + (\binom{f}{3})$ complete solutions.

3.3.3 New Upper and Lower Bounds

Local Upper Bound

The motivation behind developing the local upper bound is to try to know, as quickly as possible in the branch and bound algorithm, if a free sample can

never join a partial cluster in an optimal solution. A possible condition for this situation to occur is when there is a great increase in the lack of fit term's value when this free sample joins this partial cluster. If we know that this situation will occur, we will be able to prune all partial solutions that join the considered free sample and partial cluster. This strategy is expected to save a lot of time. The local upper bound requires that the penalty term of the information criterion to be linearly dependent on the number of clusters in the clustering alternative as it is the case for AIC, $CAIC$, and SBC. However, to derive the local upper bound we need to present some definitions, notations, and two assumptions.

We will need to use the following definitions and notations:

1. $(.., h, ..)$ represents a cluster that contains the samples whose indexes are included between the parentheses.

2. A_c^i is the matrix of sums of squares and products (SSP) of a sample or partial cluster i when it is included in the cluster c:

$$A_c^i = \sum_{j=1}^{n_i} (d_{ij} - \overline{d}_c)(d_{ij} - \overline{d}_c)'.$$

Hence, the SSP matrix of cluster c is the sum of SSP matrices of all the samples or partial clusters that cluster c contains:

$$A_c^c = \textstyle\sum_{i=1}^{k_c} A_c^i,$$

where k_c is the number of samples or partial clusters included in cluster c.

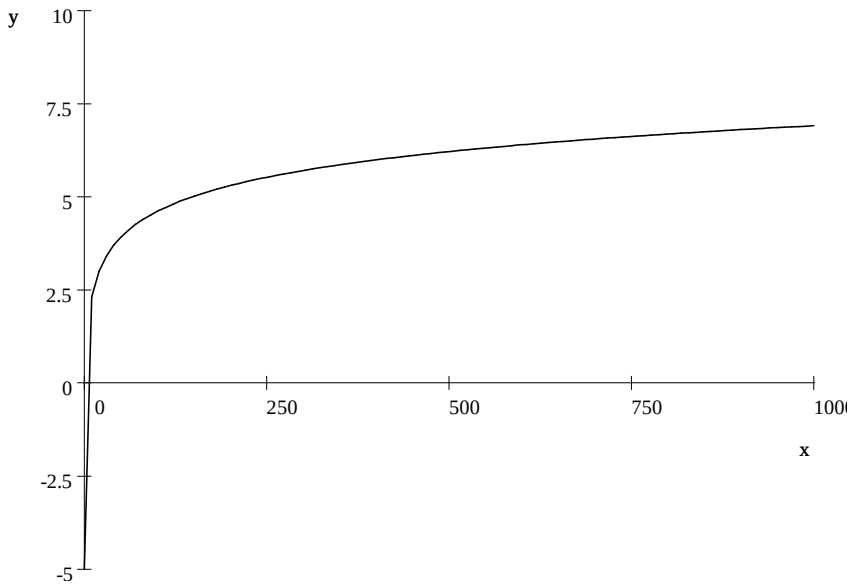

Figure 3.4: The log graph.

3. We will use the following simplified lack of fit term expressions of the MANOVA model and the varying model, respectively, as in Bozdogan (1986):

$$n \log |W| ,$$

$$\sum_{g=1}^{k} n_g \log |A_g| .$$

The Log Effect The simplified expressions of the lack of fit term for both the MANOVA model and the varying model involve using the log function. The $\log(x)$ function is concave and increasing at a decreasing rate as shown in Figure 3.4. The latter property of the $\log(x)$ function causes the same increase in x to cause a lower increase in $\log(x)$ as we go up the curve as shown in Figure 3.5.

When developing the local upper bound condition we will control for this

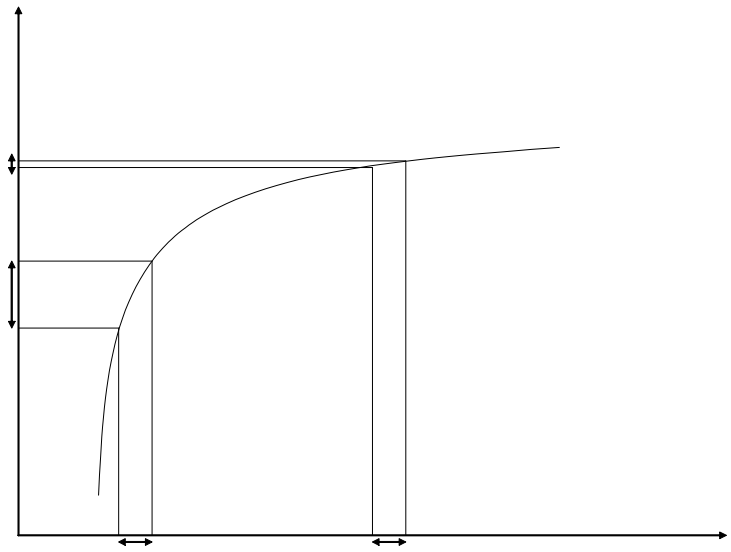

Figure 3.5: The log effect.

effect by making our comparisons at the highest possible point on the log curve. However, we begin by defining the variation of the MANOVA model and the varying model. As in Section 1.2 the simplified lack of fit term for the MANOVA model and the varying model are defined, respectively, as follows:

$$l(A_{c_1}^{c_1}, .., A_{c_k}^{c_k}) = n \log \left| A_{c_1}^{c_1} + .. + A_{c_k}^{c_k} \right|,$$

$$l(A_{c_1}^{c_1}, .., A_{c_k}^{c_k}) = n_{c_1} \log \left| A_{c_1}^{c_1} \right| + .. + n_{c_k} \log \left| A_{c_k}^{c_k} \right|.$$

The variation of the MANOVA model and the varying model, respectively, is defined as their lack of fit terms after dropping the log function and using the log properties as follows:

$$v(A_{c_1}^{c_1}, .., A_{c_k}^{c_k}) = \left| A_{c_1}^{c_1} + .. + A_{c_k}^{c_k} \right|^n,$$

$$v(A_{c_1}^{c_1}, .., A_{c_k}^{c_k}) = \left|A_{c_1}^{c_1}\right|^{n_{c_1}} \times .. \times \left|A_{c_k}^{c_k}\right|^{n_{c_k}} .$$

The Contribution The variation of a clustering alternative includes the contributions of all samples within the clusters. To discuss the local upper bound, we need a definition of this contribution that isolates the effect of a single sample or a single partial cluster within a clustering alternative.

For a specific clustering alternative, we define the contribution of a sample or partial cluster i, which is contained in a cluster $\overline{c}$, as the decrease in the variation of this clustering alternative after taking this sample or partial cluster completely out of the clustering alternative:

$$C_{\overline{c}}^{i} = v(A, A_{\overline{c}}^{i}) - v(B),$$

where A is the sum of the SSP matrices of all other samples and clusters when i is in the parent cluster $\overline{c}$, and B is the sum of the SSP matrices of all other samples and clusters when i is taken completely out of the clustering alternative.

As described in Subsection 1.2.8 and in Bao et al (2005), the monotonic condition holds for the lack of fit term of the information criteria and, subsequently, for the variation expression (i.e., as the number of clusters increases, the variation value decreases). This condition can be explained by the effect of the change in the average value used in A_c^i for some i and c. The minimum contribution of a sample or partial cluster i is given when this sample or partial

cluster is in a cluster by itself:

$$\min_{\overline{c}} C^i_{\overline{c}} = C^i_i.$$

Therefore, as a sample r is added to a sample or partial cluster i to create a new cluster $\overline{c}$, the average used for the SSP matrix of i changes. Hence, i's contribution is guaranteed to increase unless i and r already have the same average when each is in a cluster by itself.

The Nesting Property To develop our local upper bound condition, we need to make the first assumption, the nesting property. Although this assumption seems intuitive, proving it is difficult.

The nesting property states that if a sample i and a partial cluster c are joined in one cluster $\overline{c}$, then one of these two components, the sample or the partial cluster, will account for at least half of the increase in the variation in the clustering alternative. The other component, i or c, will account for the rest of the increase in the variation.

The nesting property is important when more samples r are added to the cluster $\overline{c}$ to make the new cluster c^*. In this case, by the monotonic condition the contribution of the original cluster $\overline{c}$ and the variation of the clustering alternative will increase. By the nesting property, $\overline{c}$, which contain i and c, will account for some of this new increase in the variation. Therefore, i or c will account for a portion of the total increase in the variation that is greater than half of the original increase in the variation after joining i and c together. Hence, when either i or c is taken out to form its own cluster, the variation will

decrease in an amount that is greater than half of the original increase in the variation when i and c were joined together. Using the MANOVA variation expression, the nesting property is as follows:

$$\begin{aligned} \left|A_{c,i,r}^{c,i,r}\right|^n - \left|A_c^c + A_{i,r}^{i,r}\right|^n &\geq \frac{1}{2}\left[\left|A_{c,i}^{c,i} + A_r^r\right|^n - \left|A_c^c + A_i^i + A_r^r\right|^n\right], \\ &\text{or} \\ \left|A_{c,i,r}^{c,i,r}\right|^n - \left|A_i^i + A_{i,r}^{i,r}\right|^n &\geq \frac{1}{2}\left[\left|A_{c,i}^{c,i} + A_r^r\right|^n - \left|A_c^c + A_i^i + A_r^r\right|^n\right]. \end{aligned}$$

By the monotonic condition, the contribution of cluster (i, r) will increase after adding c to it. In the same way, the contribution of cluster (c, r) will increase after adding i to it. This effect will increase the difference by an amount greater than the original case of separating cluster (c, i) to c and i.

In the same way, for the varying model, the nesting property is as follows:

$$\begin{aligned} &\left[\left|A_{c,i,r}^{c,i,r}\right|^{(n_c+n_i+n_r)} - \left(|A_c^c|^{n_c} \times \left|A_{i,r}^{i,r}\right|^{(n_i+n_r)}\right)\right] \geq \\ &\qquad \frac{1}{2}\left[\left(\left|A_{c,i}^{c,i}\right|^{(n_c+n_i)} \times |A_r^r|^{n_r}\right) - \left(|A_c^c|^{n_c} \times \left|A_i^i\right|^{n_i} \times |A_r^r|^{n_r}\right)\right], \end{aligned}$$

or

$$\begin{aligned} &\left[\left|A_{c,i,r}^{c,i,r}\right|^{(n_c+n_i+n_r)} - \left(\left|A_i^i\right|^{n_i} \times \left|A_{c,r}^{c,r}\right|^{(n_c+n_r)}\right)\right] \geq \\ &\qquad \frac{1}{2}\left[\left(\left|A_{c,i}^{c,i}\right|^{(n_c+n_i)} \times |A_r^r|^{n_r}\right) - \left(|A_c^c|^{n_c} \times \left|A_i^i\right|^{n_i} \times |A_r^r|^{n_r}\right)\right]. \end{aligned}$$

The Extended Monotonic Condition The second assumption, which is needed to develop the local upper bound condition, is the extended monotonic condition. Again, this assumption is intuitive but is difficult to prove.

The extended monotonic condition states that joining a free sample i and a partial cluster c will cause the variation to increase *more* when the other clusters have a greater contribution. This condition is represented mathematically under the MANOVA model as follows:

$$\left[\left|A_{c,i}^{c,i}+A_{(c_1,c_2)}^{(c_1,c_2)}\right|^n-\left|A_c^c+A_i^i+A_{(c_1,c_2)}^{(c_1,c_2)}\right|^n\right]\geq \left[\left|A_{c,i}^{c,i}+A_{c_1}^{c_1}+A_{c_2}^{c_2}\right|^n-\left|A_c^c+A_i^i+A_{c_1}^{c_1}+A_{c_2}^{c_2}\right|^n\right].$$

This condition is valid because its violation may lead to a violation of the original monotonic condition:

$$\left|A_{c,i}^{c,i}+A_{(c_1,c_2)}^{(c_1,c_2)}\right|^n<\left|A_{c,i}^{c,i}+A_{c_1}^{c_1}+A_{c_2}^{c_2}\right|^n.$$

Under the varying model, this condition is represented mathematically as follows:

$$\left[\left|A_{c,i}^{c,i}\right|^{(n_c+n_i)}\times\left|A_{(c_1,c_2)}^{(c_1,c_2)}\right|^{(n_{c_1}+n_{c_2})}-|A_c^c|^{n_c}\times\left|A_i^i\right|^{n_i}\times\left|A_{(c_1,c_2)}^{(c_1,c_2)}\right|^{(n_{c_1}+n_{c_2})}\right]\geq$$
$$\left[\left|A_{c,i}^{c,i}\right|^{(n_c+n_i)}\times\left|A_{c_1}^{c_1}\right|^{n_{c_1}}\times\left|A_{c_2}^{c_2}\right|^{n_{c_2}}-|A_c^c|^{n_c}\times\left|A_i^i\right|^{n_i}\times\left|A_{c_1}^{c_1}\right|^{n_{c_1}}\times\left|A_{c_2}^{c_2}\right|^{n_{c_2}}\right].$$

Like in the MANOVA model, this condition is valid because its violation may

lead to a violation of the original monotonic condition:

$$\left|A_{c,i}^{c,i}\right|^{(n_c+n_i)} \times \left|A_{(c_1,c_2)}^{(c_1,c_2)}\right|^{(n_{c_1}+n_{c_2})} < \left|A_{c,i}^{c,i}\right|^{(n_c+n_i)} \times \left|A_{c_1}^{c_1}\right|^{n_{c_1}} \times \left|A_{c_2}^{c_2}\right|^{n_{c_2}}.$$

The General Local Upper Bound Condition In stating the general local upper bound condition we will assume that we are controlling for the log effect. We will explain how this is done in the subsequent subsections. The general local upper bound condition is as follows: when joining a free sample i with a partial cluster c, if the increase in the lack of fit term's value of the first lower bound (i.e., after assigning all other free samples to their own clusters) is greater than the penalty of adding 2 clusters ($2T$), then the optimal clustering alternative will not include this free sample and this partial cluster in one cluster.

This conclusion is made because the nesting property states that if this free sample is joined to this partial cluster in one new partial cluster, then at the optimal completion of this new partial cluster, after adding r more samples, the separating of either the original free sample or the original partial cluster will reduce the lack of fit term value more than the penalty of adding 1 cluster.

Next, we will derive the specific local upper bound condition for the MANOVA model and the varying model.

The MANOVA Model Local Upper Bound Condition We will start by stating the local upper bound condition for the variation term of the MANOVA model, and then show how it is valid. Next, we will derive the

local upper bound condition for the lack of fit term after controlling for the log effect.

The variation expression under the MANOVA model is as follows:

$$\left|A_{c_1}^{c_1} + .. + A_{c_k}^{c_k}\right|^n,$$

where k is the number of clusters in the considered clustering alternative. Assume that at an adaptive branching step, the considered partial solution has k partial clusters, including c, and f free samples, including i. The free samples will be called $a_1, .., a_f$. Assume that r of the f free samples, called $a_1, .., a_r$, are joined to the (i, c) partial cluster at the optimal completion of the considered partial solution. The local upper bound condition in terms of the variation expression states that if joining i and c increases the variation term's value of the first lower bound by at least the value of the penalty of adding 2 clusters $(2T)$, then the optimal solution will not join i and c in one cluster:

$$\left|A_{(c,i)}^{(c,i)} + A_{c_1}^{c_1} + ... + A_{c_{k-1}}^{c_{k-1}} + A_{a_1}^{a_1} + .. + A_{a_{f-1}}^{a_{f-1}}\right|^n$$
$$- \left|A_c^c + A_i^i + A_{c_1}^{c_1} + ... + A_{c_{k-1}}^{c_{k-1}} + A_{a_1}^{a_1} + .. + A_{a_{f-1}}^{a_{f-1}}\right|^n \geq 2T.$$

We will consider the difference between the solution that joins i and c in one cluster and the solution that separates c in its own cluster. The conclusion will not depend on whether i or c is separated in its own cluster. The best solution's variation value where i and c are joined in one cluster is as follows:

$$\left|A_{(c,i,r)}^{c} + A_{(c,i,r)}^{i,r} + A_{c_1}^{c_1} + ... + A_{c_{k^*-1}}^{c_{k^*-1}}\right|^n,$$

where k^* is the number of clusters in this solution. The solution's variation value after separating c in its own cluster is as follows:

$$\left| A_c^c + A_{(i,r)}^{i,r} + A_{c_1}^{c_1} + ... + A_{c_{k^*-1}}^{c_{k^*-1}} \right|^n .$$

Hence, the difference between these two expressions is as follows:

$$\left| \overbrace{A_{(c,i,r)}^{c} + A_{(c,i,r)}^{i,r}}^{E1} + \overbrace{A_{c_1}^{c_1} + ... + A_{c_{k^*-1}}^{c_{k^*-1}}}^{E2} \right|^n$$
$$- \left| \overbrace{A_c^c + A_{(i,r)}^{i,r}}^{E3} + \overbrace{A_{c_1}^{c_1} + ... + A_{c_{k^*-1}}^{c_{k^*-1}}}^{E2} \right|^n \geq T.$$

By the nesting property in terms $E1$ and $E3$, c accounts for an increase in the variation's value by an amount greater than the penalty of 1 cluster. Therefore, if the rest of the terms in this difference does not offset this increase, separating c in its own cluster will decrease the variation value by more than the penalty of 1 cluster (i.e., will give a better solution).

Term $E2$ is in both sides of the subtraction but is unknown. This term represents the SSP matrices of the other clusters that contain the rest of the free samples that are not joined to (c, i). By the extended monotonic condition, we assume that this term will increase the difference more than it did at the adaptive branching step because it has more contribution than it had at the adaptive branching step. This assumption is valid because otherwise the original monotonic condition can be violated between the complete solution that is used in the branching step and the optimal solution.

However, to take into account the effect of the log function, we will add the largest value possible in all possible solutions to the arguments of the log functions in the local upper condition. This change will give us the minimum difference possible at the highest point in the log function curve. The largest value possible is the determinant of the SSP matrix W of the cluster that joins all K samples in one cluster. Hence, the local upper bound condition under the MANOVA model is:

$$\log\left[|W|^n + \left|A^{(c,i)}_{(c,i)} + A^{c_1}_{c_1} + ... + A^{c_{k-1}}_{c_{k-1}} + A^{a_1}_{a_1} + .. + A^{a_{f-1}}_{a_{f-1}}\right|^n\right]$$
$$-\log\left[|W|^n + \left|A^{c}_{c} + A^{i}_{i} + A^{c_1}_{c_1} + ... + A^{c_{k-1}}_{c_{k-1}} + A^{a_1}_{a_1} + .. + A^{a_{f-1}}_{a_{f-1}}\right|^n\right] \geq 2T. \quad (3.1)$$

The Varying Model Local Upper Bound Condition We develop the local upper bound conditions for the varying model in the same way as we did for the MANOVA model. However, in the varying model the effects of the clusters on the lack of fit term value are independent. This fact can be realized by looking at the simplified expression of the varying model's lack of fit term as in Bozdogan (1986):

$$\sum\nolimits_{g=1}^{k} n_g \log|A_g| = n_1 \log|A_1| + .. + n_k \log|A_k|\,.$$

The effects of the clusters are added together. The variation expression under the varying model is as follows:

$$|A_1|^{n_1} \times .. \times |A_k|^{n_k}\,,$$

where k is the number of clusters in the clustering alternative considered.

Following the same notation as in the MANOVA case, the local upper bound condition in terms of the variation expression states that if joining i and c increases the variation's value of the first lower bound by at least the value of the penalty of adding 2 clusters ($2T$), then the optimal solution will not join i and c in one cluster:

$$\left[\left|A^{(c,i)}_{(c,i)}\right|^{(n_c+n_i)} \times \left|A^{c_1}_{c_1}\right|^{n_{c_1}} \times .. \times \left|A^{c_{k-1}}_{c_{k-1}}\right|^{n_{c_{k-1}}} \times \left|A^{a_1}_{a_1}\right|^{n_{a_1}} \times .. \times \left|A^{a_{f-1}}_{a_{f-1}}\right|^{n_{a_{f-1}}}\right]$$
$$- \left[|A^c_c|^{n_c} \times \left|A^i_i\right|^{n_i} \times \left|A^{c_1}_{c_1}\right|^{n_{c_1}} \times .. \times \left|A^{c_{k-1}}_{c_{k-1}}\right|^{n_{c_{k-1}}} \times \left|A^{a_1}_{a_1}\right|^{n_{a_1}} \times .. \times \left|A^{a_{f-1}}_{a_{f-1}}\right|^{n_{a_{f-1}}}\right]$$
$$\geq 2T.$$

By the nesting property and the monotonic condition, either i or c will always account for an increase in the variation value by an amount greater than the penalty of 1 cluster (T). We will assume that c is the component that will be separated in its own cluster. However, the conclusion will not depend on whether i or c is separated in its own cluster.

To derive this fact, we will consider the difference between the best solution that joins i and c in one cluster and the solution that separates c in its own cluster. The best solution's variation value when joining i and c is as follows:

$$\left|A^c_{(c,i,r)} + A^{(i,r)}_{(c,i,r)}\right|^{(n_c+n_i+n_r)} \times \left|A^{c_1}_{c_1}\right|^{n_{c_1}} \times .. \times \left|A^{c_{k^*-1}}_{c_{k^*-1}}\right|^{n_{c_{k^*-1}}}.$$

The solution's variation value after separating c in its own cluster is as follows:

$$|A^c_c|^{n_c} \times \left|A^{(i,r)}_{(i,r)}\right|^{(n_i+n_r)} \times \left|A^{c_1}_{c_1}\right|^{n_{c_1}} \times .. \times \left|A^{c_{k^*-1}}_{c_{k^*-1}}\right|^{n_{c_{k^*-1}}}.$$

Hence, the difference between these two expressions is as follows:

$$\overbrace{\left|A^{c}_{(c,i,r)} + A^{(i,r)}_{(c,i,r)}\right|^{(n_c+n_i+n_r)}}^{E1} \times \overbrace{\left|A^{c_1}_{c_1}\right|^{n_{c_1}} \times .. \times \left|A^{c_{k^*-1}}_{c_{k^*-1}}\right|^{n_{c_{k^*-1}}}}^{E2}$$
$$-\overbrace{|A^{c}_{c}|^{n_c} \times \left|A^{(i,r)}_{(i,r)}\right|^{(n_i+n_r)}}^{E3} \times \overbrace{\left|A^{c_1}_{c_1}\right|^{n_{c_1}} \times .. \times \left|A^{c_{k^*-1}}_{c_{k^*-1}}\right|^{n_{c_{k^*-1}}}}^{E2} \geq T.$$

By the nesting property in terms $E1$ and $E3$, we know that c accounts for an increase in the variation value by an amount greater than the penalty of 1 cluster (T). The rest of the two terms, $E2$, are identical, which represents the SSP matrices of the other clusters that contain the rest of the free samples that are not joined to (c, i). By the extended monotonic condition, we assume that this term will increase the difference between the variations of these optimal solutions by more than it did at the adaptive branching step because it has more contribution than it had at the adaptive branching step. This assumption is valid because otherwise the original monotonic condition can be violated between the complete solution that was used in the branching step and the optimal solution. Therefore, separating c in its own cluster will decrease the variation by an amount greater than the penalty of 1 cluster.

Again, to take into account the effect of the log function, we will add the largest value possible in all possible solutions to the arguments of the log functions in the local upper condition. This change will give us the minimum difference possible at the highest point in the log function curve. The largest value possible is the determinant of the SSP matrix W_v of the cluster that joins c, i, and all the f free samples in one cluster. Hence, the local upper

bound condition under the varying model is:

$$\log\left(|W_v|^{n_v} + \left|A^{(c,i)}_{(c,i)}\right|^{(n_c+n_i)}\right) - \log\left(|W_v|^{n_v} + |A^c_c|^{n_c} \times \left|A^i_i\right|^{n_i}\right) \geq 2T. \quad (3.2)$$

Saving Technique We use a technique, thereafter called the saving technique, that is expected to save a great amount of computations. This technique takes advantage of the independence of the local upper bound condition under the varying model, for a partial cluster and a free sample, from the way the previous samples are clustered. From Equation 3.2 we see that the rest of the partial clusters have no effect on the local upper bound condition, not like under the MANOVA model as in Equation 3.1. Therefore, once a local upper bound condition is met for a partial cluster and a free sample we save the indexes of the samples in the partial cluster and the index of the free sample in a list of inequalities. Then at each branching step we check the list of inequalities to avoid recalculating the values required for the local upper bound condition when we already know that a free sample will never join a partial cluster from a previous encounter.

A heuristic From the local upper bound condition, we can develop a heuristic, thereafter called the heuristic local upper bound condition. After experimentation with a number of data sets, we found that the local upper bound condition is too conservative and can be relaxed to a good heuristic. First, the act of controlling for the log effect can be eliminated because it has a minor effect on the experiments we ran. Second, comparing the increase in the value

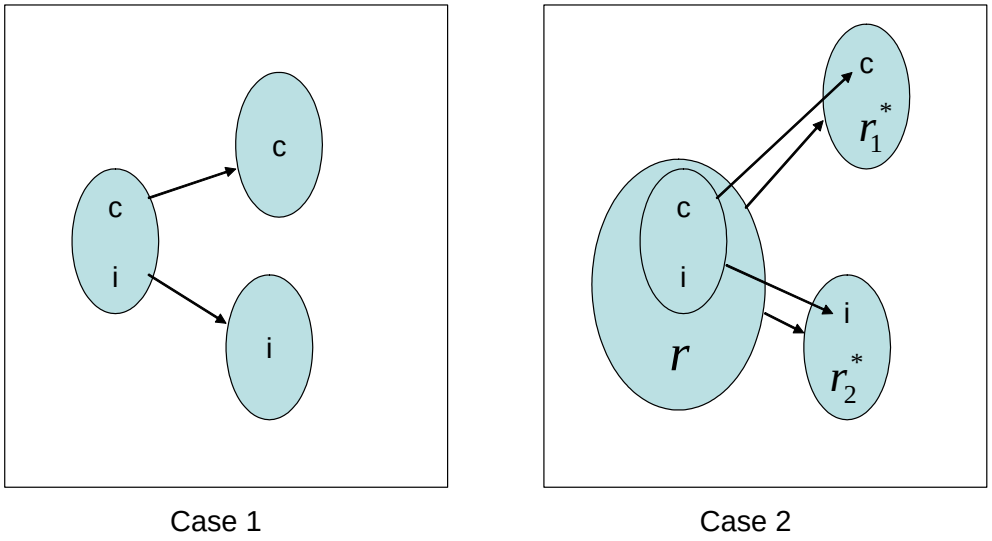

Figure 3.6: The heuristic local upper bound idea.

of the lack of fit term after joining a free sample and a partial cluster to the penalty of 2 clusters is too conservative. We can compare this increase only to the penalty of 1 cluster. The logic behind this approach is explained in Figure 3.6. In this figure, case 1 represents the comparison at the branching step and case 2 represents the comparison at the best completion after joining the partial cluster c and the free sample i. If the increase in the value of the lack of fit term after joining c and i is more than the penalty of 1 cluster, then it is very likely that we can decrease the value of the lack of fit term by more than the penalty of 1 cluster by splitting the cluster that joins c and i, including the other joined samples r, into two clusters that separate c and i and split r into r_1^* and r_2^* optimally. This action will decrease the overall information criterion value.

Therefore, the heuristic local upper bound condition states that if the increase in the value of the lack of fit term of the first lower bound after joining

a free sample and a partial cluster is greater than or equal to the penalty of 1 cluster, then this free sample and this partial cluster should not be joined in one cluster.

New Lower Bound

If the assignment of at least one free sample to every formed partial cluster meets the local upper bound condition, then we can use the penalty term of $k+1$ clusters for the lower bound of this partial solution. This is possible because this free sample must create a new cluster no matter what the completion of this partial solution is.

More New Lower Bounds

If more than one free sample meets the new lower bound condition, then we can apply the local upper bound condition between each two of these free samples. If at least one of these free samples meets the local upper bound condition with every other sample of these free samples then we can use the penalty term of $k+2$ clusters for the lower bound on this partial solution. This new lower bound is possible because the free sample that meets its condition must create a new cluster without including any of the other free samples that met the earlier condition.

This lower bound procedure can apply repeatedly to the free samples that meet the previous conditions.

Remark

Because the number of clusters in a node is the same as or greater than the number of clusters in its parent node, the penalty part value of the lower bound for any node should be either this node's penalty part value or its parent node's penalty part value, whichever is greater. This fact is important because the results of the computations of the new lower bounds for a parent node can be reused for all of its child nodes without the need for any additional computations.

3.4 Complete Enumeration Algorithm

The variables used in the complete enumeration algorithm are defined as follows:

$$x_{ij} = \left\{ \begin{array}{l} 0 \text{ if samples } i \text{ and } j \text{ are not in the same cluster.} \\ 1 \text{ if samples } i \text{ and } j \text{ are in the same cluster.} \\ 2 \text{ if not decided.} \end{array} \right\}$$

$$, 1 \leq i < j \leq K;$$

$$p_{ij} = \left\{ \begin{array}{l} \text{1 if the } x_{ij} \text{ value has been set by one of} \\ \text{the constraints.} \\ \\ \\ \text{0 otherwise.} \end{array} \right\}$$

$$, 1 \leq i < j \leq K.$$

The complete enumeration algorithm begins by assigning 0 to all decision variables x_{ij} and then uses sequential branching to join the samples sequentially by changing the decision variables' values from 0 to 1. The complete enumeration algorithm is presented in Algorithm 3.1. This algorithm has 4 steps: Set, Check, Lower Level, and Backtrack steps. The Set step changes the values of the decision variables in order to enumerate all possible clustering alternatives. The Check step evaluates the objective function value for each feasible solution. The Lower Level and Backtrack steps perform the sequential branching strategy. The Set and Free by Constraint functions, which are used in the complete enumeration algorithm, are presented in Algorithm 3.2. These functions implicitly set or free the decision variables by the triangle constraints.

Initialization: $x_{ij} := 2\ \forall i, j$, $p_{ij} := 0\ \forall i, j$, $i := 1, j := 2$
Set: if ($i = 1$, $j = 2$, and $x_{12} = 1$), **Stop**
if ($x_{ij} = 2$), $x_{ij} := 0$, **Set By Constraint**$(x, 0, i, j, p)$, go to **Check**
if ($x_{ij} = 0$), $x_{ij} := 1$, **Set By Constraint**$(x, 1, i, j, p)$, go to **Check**
if ($x_{ij} = 1$), go to **Backtrack**
Check: if ($x_{fg} \neq 2$) $\forall f, g$,
Evaluate Feasible Solution Objective Value(x)
go to **Set**
else go to **Lower Level**
Lower Level: if ($j < K$), $j := j + 1$
else $i := i + 1$ and $j := i + 1$
if ($p_{ij} = 1$) go to **lower leve**l
else go to **Set**
Backtrack: if ($i = 1$ and $j = 2$) , go to **Set**
if ($p_{ij} = 0$) , **Free by Constraints**(x, i, j, p)
if ($j > (i + 1)$), $j := j - 1$
else $i := i - 1$ and $j := K$
if ($p_{ij} = 1$) go to **Backtrack**
else go to **Set**

Algorithm 3.1: Complete enumeration algorithm.

3.5 Sequential Branch and Bound Algorithm

3.5.1 Introduction

The sequential branch and bound algorithm, like the complete enumeration algorithm, branches on the decision variables sequentially. However, for each partial solution, the upper and lower bounds are evaluated. Then, if the lower bound value exceeds the upper bound value, the algorithm prunes this partial solution and goes to the next value of the considered decision variable. In this algorithm, we use only the first lower bound and the general upper bound.

There are two ways to apply the sequential branch and bound algorithm: the agglomerative and the divisive ways. In the agglomerative way, we begin by assigning 0 to all decision variables and then joining the samples sequentially

Set by Constraint(x,0 or 1,i,j,p)
    There are two 2 cases:
        x_{ij}:2 $\rightarrow$ 0
            for $f = i + 1$ to $j - 1$
                if ($x_{if} = 1$) then
                    $x_{fj} := 0$
                    $p_{fj} := 1$
        x_{ij}:0 $\rightarrow$ 1
            for $f = i + 1$ to $j - 1$
                if ($x_{if} = 1$)
                    $x_{fj} := 1$
                    $p_{fj} := 1$
                else if ($x_{if} = 0$ and $p_{if} = 0$)
                        $x_{fj} := 0$
                        $p_{fj} := 1$
Free by Constraint(x,i,j,p)
    for $f = i + 1$ to $j - 1$
        if ($x_{if} = 1$)
            $x_{fj} := 2$
            $p_{fj} := 0$
        else if ($x_{if} = 0$ and $p_{if} = 0$)
                $x_{fj} := 2$
                $p_{fj} := 0$

Algorithm 3.2: Set and free by constraint functions.

by changing the decision variables values from 0 to 1. In the divisive way, we begin by assigning 1 to all decision variables and then dividing the samples sequentially by changing the decision variables values from 1 to 0. The rest of the algorithm is the same for both ways except for the constraint functions, as will be shown subsequently. The agglomerative way will be used when we expect the population to have many clusters, and the divisive way will be used when we expect the population to have few clusters. This strategy will make the algorithm starts searching from the extreme that is more likely to be closer to the optimal solution.

3.5.2 Agglomerative Sequential Branch and Bound Algorithm

The agglomerative sequential branch and bound algorithm is presented in Algorithm 3.3. The agglomerative sequential branch and bound algorithm uses the same steps of the complete enumeration algorithm except for the Check step. In this step, the bounds are checked and the flow of the algorithm is directed according to the results of the bounding strategies. This algorithm uses the same functions in Algorithm 3.2.

Initialization: $x_{ij} := 2 \ \forall i,j, \quad p_{ij} := 0 \ \forall i,j, \quad i := 1, j := 2,$ Upper Bound= $+\infty$
Set: if ($i = 1$, $j = 2$, and $x_{12} = 1$), **Stop**
if ($x_{ij} = 2$), $x_{ij} := 0$, **Set By Constraint**$(x, 0, i, j, p)$, go to **Check**
if ($x_{ij} = 0$), $x_{ij} := 1$, **Set By Constraint**$(x, 1, i, j, p)$, go to **Check**
if ($x_{ij} = 1$), go to **Back Track**
Check: if ($x_{fg} \neq 2 \quad \forall f, g$),
Evaluate Feasible Solution Objective Value(x)
if Solution Objective Value < Upper Bound,
Upper Bound := Solution Objective Value
go to **Set**
else
evaluate Partial Solution First Lower Bound(x)
if (Partial Solution Lower Bound > (Local) Upper Bound), go to **Set**
else go to **Lower Level**
Lower Level: if ($j < K$), $j = j + 1$
else $i := i + 1$ and $j := i + 1$
if ($p_{ij} = 1$), go to **Lower Level**
else go to **Set**
Back Track: if ($i = 1$ and $j = 2$), go to **Set**
if ($p_{ij} = 0$), **Free by Constraints**(x, i, j, p)
if ($j > (i + 1)$), $j := j - 1$
else $i := i - 1$ and $j := K$
if ($p_{ij} = 1$), go to **Backtrack**
else go to **Set**

Algorithm 3.3: Agglomerative sequential branch and bound algorithm.

Initialization: $x_{ij} := 2\ \forall i, j, \quad p_{ij} := 0\ \forall i, j, \quad i := 1, j := 2,$
Upper Bound $:= +\infty$
Set: if ($i = 1$, $j = 2$, and $x_{12} = 0$), **Stop**
if ($x_{ij} = 2$), $x_{ij} := 1$, **Set By Constraint**$(x, 1, i, j, p)$, go to **Check**
if ($x_{ij} = 1$), $x_{ij} := 0$, **Set By Constraint**$(x, 0, i, j, p)$, go to **Check**
if $x_{ij} = 0$, go to **Backtrack**

Algorithm 3.4: Divisive sequential branch and bound algorithm.

3.5.3 Divisive Sequential Branch and Bound Algorithm

Only the parts of the divisive sequential branch and bound algorithm that are different from the agglomerative algorithm are presented in Algorithm 3.4. These parts are the Initialization and the Set steps where the order of the assignment of 0 or 1 is reversed. The Set and Free by Constraint functions for the divisive algorithm are presented in Algorithm 3.5. These functions perform the same task as in the complete enumeration algorithm except for the order of the assignment of 0 or 1.

3.6 Adaptive Branch and Bound Algorithm

In the adaptive branch and bound algorithm, we use the adaptive branching strategy (Subsection 3.2.2), the first lower bound (Subsection 3.3.2), and the general upper bound (Subsection 3.3.1). The variables of the adaptive branch and bound algorithm are defined as follows:

Set by Constraint(x,0 or 1,i,j,p)
There are two 2 cases:

x_{ij}:2 $\rightarrow$ 1
    for $f = i + 1$ to $j - 1$
        if ($x_{if} = 1$)
            $x_{fj} := 1$
            $p_{fj} := 1$
        else if ($x_{if} = 0$ and $p_{if} = 0$)
            $x_{fj} := 0$
            $p_{fj} := 1$

x_{ij}:1 $\rightarrow$ 0
    for $f = i + 1$ to $j - 1$
        if ($x_{if} = 1$)
            $x_{fj} := 0$
            $p_{fj} := 1$
        else if ($x_{if} = 0$ and $p_{if} = 0$)
            $x_{fj} := 2$
            $p_{fj} := 0$

Free by Constraint(x,i,j,p)
    for $f = i + 1$ to $j - 1$
        if ($x_{if} = 1$)
            $x_{fj} := 2$
            $p_{fj} := 0$

Algorithm 3.5: Set and free by constraint functions for the divisive sequential branch and bound algorithm.

$$x_{ij} = \left\{ \begin{array}{l} 1 \text{ if samples } i \text{ and } j \text{ are in the same cluster.} \\ 0 \text{ if samples } i \text{ and } j \text{ are not in the same cluster.} \end{array} \right\}$$

$$, 1 \leq i < j \leq K$$

$$p_i = \left\{ \begin{array}{l} 0 \text{ if sample } i \text{ is free.} \\ 1 \text{ if sample } i \text{ is a sample that started a new cluster.} \\ 2 \text{ if sample } i \text{ is joined to an already established cluster.} \end{array} \right\}$$

$$, i = 1, .., K$$

$$s_{ij} = \left\{ \begin{array}{l} 1 \text{ if the algorithm has enumerated the value 1 for } x_{ij}. \\ 0 \text{ otherwise.} \end{array} \right\}$$

$$, 1 \leq i < j \leq K.$$

When using x_{ij} or s_{ij}, we will always assume i and j are in the right order (i.e., $i < j$).

$$s_{ii} = \left\{ \begin{array}{l} \text{1 if the value 0 has been enumerated at the same} \\ \text{time for all } x_{ij} \text{ with the already established clusters.} \\ \\ \text{0 otherwise.} \end{array} \right\}$$

$$, 1 \leq i \leq K$$

$clusterorder = []$ ($clusterorder$ is a multi-dimensional matrix holding the lower bound and AIC values for all branches of each free sample in the order they are selected.)

$current = 1, .., K$ ($current$ is the index of the currently considered free sample.)

The adaptive branch and bound algorithm is presented in Algorithm 3.6. This algorithm has only three steps: Set, Lower Level, and Backtracking. The x_{ij} and s_{ij} variables are used to enumerate either implicitly or explicitly all possible clustering alternatives. s_{ij} keeps track of the x_{ij} values that has been already enumerated and hence the algorithm will not enumerate them again. The Set step evaluates either the AIC value of a complete solution or the lower bound of a partial solution. The Lower Level step picks the next unchecked partial solution to branch on. The Backtrack step shifts the branching to the closest unchecked parent partial solution. The functions used in this algorithm are presented in Algorithm 3.7. These functions are the Update, Order, and Free functions. The Update function updates the x_{ij} and s_{ij} values as the algorithm progresses. The Order function evaluates the lower bounds and

orders them in the *clusterorder* matrix. The Free function frees the x_{ij}, p_i, and s_{ij} values when the algorithm backtracks to the closest unchecked parent partial solution.

Initialization: $x_{ij} := 0\ \forall i,j$, $p_i := 0\ \forall i$, $s_{ij} := 0\ \forall i,j$, *clusterorder* is *empty*, $current := 1$, $p_{current} := 1$, $current := 2$

Set: if ($current = 2$ and $s_{2,1} = 1$ and $s_{2,2} = 1$), **Stop**

if ($current = K$),

pick minimum AIC for all possible values of $x_{i,current}$ where $p_i = 1$:$AIC_{\min}$

if ($AIC_{\min}$ is at all $x_{i,current} = 0$), $i_{\min} := current$

else $i_{\min} := i$ where $x_{i,current} = 1$ gives $AIC_{\min}$

Update$(x, i_{\min}, current, s)$

if ($AIC_{\min} <$ Upper Bound), Upper Bound $:= AIC_{\min}$, $x^* := x$

go to **Backtrack**

else

if ($p_{current} = 0$)

$\forall i$ where $p_i = 1$ find lower bound AIC for all possible values of $x_{i,current}$

Order(*clusterorder*, AIC values,current,s,Upper Bound)

if (lowest lower bound $\geq$ Upper Bound), go to **Backtrack**

else go to **Lower Level**

Lower Level: Pick minimum lower bound in *clusterorder* where $s_{i,current} = 0$ *or* $s_{current,current} = 0 : i_{\min}$, **Update**$(x, i_{\min}, current, s)$

if ($i_{\min} = current$), $p_{current} := 1$

else $p_{current} := 2$, $current := current + 1$, go to **Set**

Backtrack: if ($current = 2$), go to **Set**

Free$(x, s, current)$, $current := current - 1$

if ($\forall i$ where $p_i = 1 : s_{i,current} = 1$ and $s_{current,current} = 1$), go to **Backtrack**

else go to **Lower Level**

Algorithm 3.6: Adaptive branch and bound algorithm.

Update$(x, i_{\min}, current, s)$

if $(i_{\min} \neq current)$,

$x_{i_{\min},current} := 1$

$s_{i_{\min},current} := 1$

for $(j = 1 : K/\{i_{\min}, current\})$

if $(x_{i_{\min},j} = 1)$

$x_{current,j} := 1$

else $s_{current,current} := 1$

Order$(clusterorder, AIC$ values,$current, s$, Upper Bound)

Order the indexes i where $p_i = 1$ and $current$ (starts a new cluster) according

to their AIC lower bound values.

Store the ordered AIC lower bound values in $clusterorder(current)$

$\forall i$ where $p_i = 1$ and lower bound$_i \geq$ Upper Bound, $s_{i,current} := 1$

Free$(x, s, current)$

$\forall i$ where $p_i = 1$

$s_{i,current} := 0$ (free current)

$s_{current,current} := 0$

$p_{current} := 0$

for $(f = 1 : K/current)$

$x_{f,current} := 0$

Algorithm 3.7: Functions of the adaptive branch and bound algorithm.

Sample Order$(D, pickorder)$
Put all samples in one cluster.
Take one sample at a time out of this cluster (after returning the previous sample)
Score lack of fit part for each case
Order samples in *pickorder* vector according to lack of fit values in ascending order

Algorithm 3.8: Sample order function.

3.7 Adaptive Branch and Bound Algorithm With Reordering

The adaptive branch and bound algorithm with reordering is the same as the adaptive branch and bound algorithm with the exception that this algorithm is preceded by a function that reorders the way in which the free samples are considered.

We use a new variable, $pickorder = []$, which is an array holding the indexes of the free samples in the order they are selected. Before beginning the algorithm, the Sample Order function, as shown in Algorithm 3.8, is run. This function orders the samples according to the first reordering strategy discussed in Subsection 3.2.3. The rest of the algorithm is the same as in the original adaptive algorithm except that all appearances of the *current* variable as a subscript are replaced with $pickorder(current)$ (i.e., we use $x_{i,pickorder(current)}$, $p_{pickorder(current)}$, and $s_{i,pickorder(current)}$).

3.8 Complete Adaptive Branch and Bound Algorithm With Reordering

The complete adaptive branch and bound algorithm with reordering is the same as the adaptive branch and bound algorithm with reordering with the exception of using the complete adaptive branching strategy as discussed in Subsection 3.2.4. The complete adaptive branch and bound algorithm with reordering has the same variables as the adaptive branch and bound algorithm with reordering for each saved branch except for the *pickorder* matrix. These variables are defined as follows:

t is the index of the saved branches.

$\max t$ is the largest number of branches that can be saved.

$last^t$ is the last node explored in branch t.

$parent^t$ is the parent node for each saved branch.

$$x_{ij}^t = \left\{ \begin{array}{l} 1 \text{ if samples } i \text{ and } j \text{ are in the same cluster.} \\ 0 \text{ if samples } i \text{ and } j \text{ are not in the same cluster.} \end{array} \right\}$$

$$, 1 \leq i < j \leq K$$

$$p_i^t = \left\{ \begin{array}{l} 0 \text{ if sample } i \text{ is free.} \\ 1 \text{ if sample } i \text{ is a sample that started a new cluster.} \\ 2 \text{ if sample } i \text{ joined an already established cluster.} \end{array} \right\}$$

$$, i = 1, .., K$$

$$s_{ij}^t = \left\{ \begin{array}{l} 1 \text{ if the algorithm has enumerated the value 1 for } x_{ij}. \\ 0 \text{ otherwise.} \end{array} \right\}$$

$$, 1 \leq i < j \leq K$$

When using x_{ij}^t or s_{ij}^t, we will always assume i and j are in the right order (i.e., $i < j$).

$$s_{ii}^t = \left\{ \begin{array}{l} 1 \text{ if the the value 0 has been enumerated at the same} \\ \text{time for all } x_{ij} \text{ with the already established clusters.} \\ \\ 0 \text{ otherwise.} \end{array} \right\}$$

$$, 1 \leq i \leq K$$

$pickorder = []$ ($pickorder$ is an array holding the indexes of the free samples in the order they are selected, as in Algorithm 3.8).

$clusterorder^t = []$ ($clusterorder^t$ is a multi-dimensional matrix holding

the current information for all levels in branch t for each free sample in the order it is selected.)

$clusterorder = []$ ($clusterorder$ is a multi-dimensional matrix holding all $clusterorder^t$ matrices.)

$current^t = 1, .., K$ ($current^t$ is the index of the current free sample considered in $pickorder$.)

The complete adaptive branch and bound algorithm is preceded by the same Sample Order function in Algorithm 3.8. We present only the steps of the complete adaptive branch and bound algorithm with reordering that are different from the adaptive branch and bound algorithm with reordering in Algorithm 3.9. These steps include the Backtracking step. In this step, the algorithm chooses between starting a new branch, backtracking to a parent branch, or backtracking to a parent partial solution. The rest of the steps are the same as in the adaptive branch and bound algorithm with reordering except for using the superscript t. The functions used in the complete adaptive branch and bound algorithm with reordering are the same functions in Algorithm 3.7.

3.9 The Lower Bounds Modules

Here, we present the bounds modules for the adaptive and the complete adaptive branching strategies. These bounds modules are used in the previous algorithms but are presented here in greater detail.

The number of partial clusters k in a partial solution is the number of p_i's that are equal to 1. The number of free samples f in a partial solution is the number of p_i's that are equal to 0.

Initialization: $t := 1$, $x_{ij}^t := 0\ \forall i, j$, $p_i^t := 0\ \forall i$, $s_{ij}^t := 0\ \forall i, j$, $clusterorder$ is empty,
$current^t := 1, p_{pickorder(current^t)} := 1, current^t := 2,\ \ t_{\max} := 0,$

Backtrack: if $(current^t = 2)$, go to **Set**
Free($x^t, s^t, current^t$), $\quad current^t := current^t - 1$
if $(t_{\max} < \max t)$
$t := t + 1,\ \ t_{\max} := t$
$clusterorder := [clusterorder\ clusterorder^t]$
pick minimum lower bound in $clusterorder$ for all t:
$t_{\min}, current_{\min}, j_{\min}$ where $s^{t_{\min}}_{pickorder(current_{\min}),j_{\min}} = 0$
$current^t := current_{\min}, \quad parent^t := current^t$
$x^t := x^{t_{\min}},\ \ s^t := s^{t_{\min}},\ \ p^t := p^{t_{\min}}, s^{t_{\min}}_{pickorder(current_{\min}),j_{\min}} := 1$
$for\ (f = current^{t_{\min}} : current^t)$
Free(x^t, s^t, f)
go to **Lower Level**
else if $(current^t = parent^t)$
$t := t - 1$
go to **Lower Level**
else
if $(\forall i$ where $p_i^t = 1 : s^t_{i,pickorder(current^t)} = 1$ and
$s^t_{pickorder(current^t),pickorder(current^t)} = 1)$
go to **Backtrack**
else go to **Lower Level**

Algorithm 3.9: Complete adaptive branch and bound algorithm with reordering.

First Lower Bound$(xb, p, sb, current)$:
$\forall i$ where $p_i = 1$ leave all $xb_{i,current} := 0$ (start a new cluster)
$part1$:
Compute Lack of Fit part for x (free samples in own cluster $xb_{ij} = 0$).
$part2$:
Compute AIC Penalty for $k + 1$ clusters (add one for this new cluster).
lower bound$_{current} := part1 + part2$
For each i where $p_i = 1$
Set $xb_{i,currrent} := 1$, update$(xb, i, current, sb)$
$part1$:
Compute Lack of Fit for xb (free samples in own cluster $xb_{ij} := 0$).
$part2$:
Compute AIC Penalty for k clusters.
lower bound$_i := part1 + part2$

Second Lower Bound(xb, pb, sb):
For each i where $pb_i = 0$ and each j where $pb_j = 1$
Set $xb_{i,j} = 1$, update(xb, i, j, sb)
Compute lack of fit for xb (for all other i where $pb_i = 0$ set $xb_{ij} = 0$)
For each two i and j where $pb_i = 0$ and $pb_j = 0$
Set $xb_{i,j} = 1$
Compute lack of fit for xb (for all other i where $pb_i = 0$ set $xb_{ij} = 0$)
Pick the minimum of these Lack of Fit parts: $Part1$
Compute AIC Penalty for k clusters: $part2$
lower bound$_i = part1 + part2$

Algorithm 3.10: Functions of the first and second lower bounds.

3.9.1 The Modules

The functions of the first and second lower bounds are presented in Algorithm 3.10. The new bounds can be implemented as discussed in Subsection 3.3.3.

3.9.2 Computational Remarks

The computational results for the lower bounds can be reused in the adaptive and complete adaptive branching strategies for all lower-level free samples except for the clusters that have changed their content.

3.10 Experimental Results

We conducted a series of experiments in order to find the branch and bound strategies that are more likely to perform well on any given data set. First, we conducted preliminary experiments using few branch and bound strategies in order to show the difficulty of the MSCA problem. Then, we tested all the branch and bound strategies in stages on a selected data set to verify the best performing strategies. Finally, we tested the best performing strategies on other data sets.

All strategies were coded using MATLAB Version 7.0 and were run on a computer with a CPU speed of 2.8 GHz. We used AIC as the objective function in all experiments. All considered data sets required using the varying model.

3.10.1 Preliminary Experiments

We coded only the complete enumeration and the sequential branch and bound algorithms to understand the difficulty of the MSCA problem and evaluate the performance of the first lower bound and the general upper bound.

We used two real data sets and two simulated data sets. Table 3.1 is a summary of the results of these experiments.

IRIS Data Set

The IRIS data set was used first by Fisher (1936) and was then used widely by many researchers in developing algorithms for observation-wise cluster analysis. This data set consists of 150 observations on 4 variables. It's known that

Table 3.1: Results of the preliminary experiments.

Data Set	K	n_i	n	p	Algorithm	Time (sec.)	Checked
IRIS	15	30	150	4	Complete	$20,000^*$	$1,000,000$
IRIS	15	30	150	4	Sequential	$12,599$	1762
Bank Customers	15	100	1500	9	Sequential	$84,400^*$	-
Simulated (Well Separated)	15	100	1500	9	Sequential	166	18
Simulated (Poorly Separated)	15	100	1500	9	Sequential	237	393

*did not converge

this data set comes from three groups (Fisher 1936). Because we are developing algorithms for Multi-Sample Cluster Analysis, we need to group these 150 observations in a certain way in order to apply our algorithms and allow others in the future to use this same benchmark data set to compare their algorithms to ours. We decided to group the 150 observations into 15 groups of 10 observations each in the order they appear in the data set. This grouping divides each of the known three groups of 50 observations into 5 subgroups.

Complete Enumeration We ran the complete enumeration algorithm for $20,000$ seconds (about 5 hours and 30 minutes). In this time, the algorithm checked a little more than 1 million complete solutions, less than 0.1% of all possible solutions. These results mean that we need $5,500$ hours (about 229 days) to find the optimal solution.

Sequential Branch and Bound Because we know that the optimal solution of the grouped IRIS data set is not close to the two extremes (1 or 15

clusters), running the sequential branch and bound algorithm with no initial upper bound will require a lot of time. We decided to run the sequential branch and bound algorithm with the upper bound equal to the optimal AIC value. This run will test the performance of the first lower bound.

The algorithm converged after about $12,000$ seconds (about 3 hours and 20 minutes). The algorithm checked only $1,762$ complete solutions. The optimal solution value is 119.47. This performance tells us that the first lower bound is good but not good enough. We expect that with the addition of the branching strategies and the other bounding strategies, we will be able to improve this performance.

SPSS Bank Customers Data Set

This data set consists of $1,500$ observations with 9 variables. These observations are divided into 15 samples of 100 observations each. Each sample represents customers of one of the 15 branches of this bank.

Because we have no information about the optimal solution, we ran the sequential branch and bound algorithm with no initial upper bound. The algorithm ran for more than 24 hours and did not converge. We expect that with the branching strategies and other bounding strategies, we will be able to improve this performance.

Simulated Data Sets (Separability Factor)

An important factor that may affect the performance of any algorithm that tries to solve any cluster analysis problem, is the degree to which the samples, objects, or clusters are separated. This is known as the separability factor

(Koontz, Narendra, and Fukunaga 1975; Diehr 1985; Brusco 2006). A good separation means that the structure of the populations involved is clear (i.e., the clusters' boundaries are obvious). A bad separation means that there is no clear cluster structure and that the samples or objects can be joined in many good ways. We expect this factor to play an important role in the performance of our algorithms.

Therefore, we simulated two data sets with the same characteristics as the Bank Customers data set (i.e., $K = 15, n_i = 100, n = 1500, p = 9$). The first simulated data set had strongly separated clusters and the second one had slightly separated clusters. We ran the sequential branch and bound algorithm for both simulated data sets. The algorithm converged in about 166 seconds and checked only 18 complete solutions for the data set with the well-separated clusters. On the other hand, it converged in about 237 seconds and checked 393 complete solutions for the data set with the slightly separated clusters. This shows the effect of the actual structure of the data set. However, if a real population is slightly separated, finding the optimal solution is not important because there are a lot of good solutions close in information criterion value to the optimal solution. In this case, a good heuristic can efficiently find a good enough solution.

3.10.2 Evaluation of Strategies Using the IRIS Data Set

We selected the IRIS data set for testing the branch and bound strategies because it is a well known bench-mark data set. Table 3.2 shows the results of 6 experiments which use the branch and bound strategies that performed well

Table 3.2: Branch and bound strategies that performed well on the IRIS data set.

#	Branch.	Reorder	Bounds	Time (Error)	Nodes	Solutions	Opt.
1	Seq.	-	First	12,599	376,782	1,762	Given
2	Adap.	Desc.	First	2,877	27,289	811	3
3	Adap.	Asc.	First	652	6,474	559	262
4	Adap.	Asc.	+Local	529	4,790	559	262
5	Adap.	Asc.	+Save	529	4,790	559	262
6	Adap.	Asc.	+Heuristic	146 (0.04%)	1,138	284	6
Optimal Clustering Alternative: (1,2,3,4,5)(6,7,8,9)(10)(11,12,13,14)(15)							

(a row for each experiment). The last row shows the actual optimal clustering alternative selected by AIC. This solution includes three big clusters and two small clusters that contain only one sample each. The big three clusters are consistent with the known real structure of the IRIS data set which has only three clusters. The two small clusters are a result of the low penalty value that AIC applies to the increase in the number of clusters. We expect that other information criteria with bigger penalty part, like $CAIC$, SBC, or $ICOMP$, will choose the right clustering alternative that has three clusters only.

In this table the first column shows the branching strategy used in each experiment (Sequential or Adaptive). The first experiment is a repetition of the second experiment in Table 3.1. In this experiment only the first lower bound and the general upper bound were used. In addition, in the first experiment the optimal solution value was used as the initial upper bound. The second column in Table 3.2 specifies which reordering strategy is being used (the ascending or the descending reordering strategy). The third column specifies the bounding strategies used (first lower bound, local upper bound, saving technique, and/or

the heuristic local upper bound). The last four columns report the performance measures on the algorithms in each experiment: execution time in seconds, number of enumerated nodes (or number of enumerated partial solutions), number of complete solutions checked, and the order in which the optimal solution was found among the complete solutions checked.

We see that the adaptive branching with descending reordering in the second experiment outperforms the sequential branching in the first experiment on all measures and finds the optimal solution early among the complete solutions checked. When only the reordering strategy is changed to the ascending reordering strategy in the third experiment, the performance of the branch and bound algorithm improves significantly on all measures except for the last column. The ascending reordering strategy finds the optimal solution later, in the execution, than the descending reordering strategy does. We can see here a trade-off between the speed of convergence and the speed of finding the optimal solution in the execution. The descending reordering strategy finds the optimal solution quickly but is slow in convergence. The ascending reordering strategy has the opposite performance. The speed of convergence is important because we need fast algorithms. A fast convergence means that the algorithm is good at pruning more partial solutions. The speed of finding the optimal solution is also important because some problems require a very long execution time. In this case if the algorithm is fast at finding the optimal solution, then even if we stop the algorithm before convergence, we are more likely to have found the optimal solution. We will see in the next subsections how the complete adaptive branching strategy can solve this problem and achieve a good performance on both of these measures.

The fourth experiment adds the local upper bound strategy. In this experiment we see that this strategy reduces the time and the number of enumerated nodes but keeps the other two performance measures the same. This performance tells us that this strategy prunes the bad branches of the enumeration tree earlier in the process but still enumerates the same complete solutions. However, this improvement is acceptable.

The saving technique is added in the fifth experiment. The saving technique, as explained in Subsection 3.3.3, tries to save computations by storing the pruning decisions of the local upper bound. We see that this technique did not change the performance at all. This result can be viewed as a good performance because it shows that the time required to check the list of saved decisions is equal to the time required to recalculate the local upper bound condition. For different data sets we expect that this strategy will save time, especially when there are more pruning decisions.

The sixth experiment shows the performance after adding the heuristic version of the local upper bound, as explained in Subsection 3.3.3, along with the first lower bound and saving technique. This heuristic has a superior performance on all measures except that it finds a suboptimal solution. However, this suboptimal solution has only an error of 0.04% compared to the optimal solution found by the other strategies.

Table 3.3 summarizes the results of the experiments on the branch and bound strategies that did not perform well. This table follows the same format as Table 3.2. The first experiment is a repetition of the experiment that used the adaptive branch and bound algorithm with the first lower bound and the local upper bound as in Table 3.2. The second experiment adds the second

Table 3.3: Branch and bound strategies that did not perform well on the IRIS data set.

#	Branch.	Reorder.	Bounding	Time	Nodes	Solutions	Opt.
1	Adap.	Asc.	First,Local	529	4,790	559	262
2	Adap.	Asc.	First,Local,Sec.	2,594	3,103	544	247
3	Adap.	Asc.	First,Local,New	1,344	4,790	559	262

lower bound as explained in Subsection 3.3.2. In this experiment, although the number of enumerated nodes decreases, the execution time increases dramatically. The reason for this is that the added bound does not make enough extra pruning decisions while requiring many more computations. Although some of these computations can be avoided by saving their results as we did in the saving technique, any technique to save all of the computations will require other extra computations. Hence, we do not expect that this technique will offset the huge increase in execution time.

The third experiment adds the new lower bound, as explained in Subsection 3.3.3, to the first lower bound and the local upper bound. This added bound did not prune any extra nodes but required an increase in time. This performance can be due to the fact that there is a small chance of finding a sample that can not join any of the already formed partial clusters as required by this bound.

The results of the experiments that use the complete adaptive branching strategies are shown in Table 3.4. All of these experiments use the first lower bound, the local upper bound, and the saving technique. The first experiment uses the adaptive branching strategy with descending reordering. The second experiment uses the adaptive branching strategy but with ascending reordering. Again, we can see, from these two experiments, the trade-off between fast

Table 3.4: The complete adaptive branching performace on the IRIS data set.

#	Branch.	Reorder.	Time	Nodes	Solutions	Optimal
1	Adap.	Desc.	2,328	20,188	811	3
2	Adap.	Asc.	529	4,790	559	262
3	Comp. t=2	Asc.	529	4,797	559	465
4	Comp. t=3	Asc.	531	4,787	559	19
5	Comp. t=4	Asc.	533	4,787	559	19
6	Comp. t=5	Asc.	534	4,787	559	19
7	Comp. t=10	Asc.	534	4,787	559	19
8	Comp. t=50	Asc.	540	4,787	559	19

convergence and fast finding of the optimal solution. The third experiment starts the use of the complete adaptive branching strategy with ascending reordering but only saves two branches. We can see that saving two branches does not change any of the performance measures except that the optimal solution is found later in the execution which is the opposite of the goal of using this branching strategy.

However, the fourth and later experiments increase the number of branches saved gradually and show a great improvement in the speed of finding the optimal solution. In these experiments the optimal solution is found much earlier than in the second and third experiments. We can see also that the time shows a very small increase with the increase of the number of branches saved in the complete adaptive branching strategy. Hence, we can increase the number of branches saved as much as the computer memory allows. Therefore, with the complete adaptive branching strategy we achieved the qualities of the ascending and descending reordering strategies simultaneously: we find the optimal solution early in the execution of the algorithm without affecting the speed of convergence.

Based on this performance we recommend using one of two algorithms. The first algorithm, called the complete algorithm, uses the complete adaptive branching strategy, the ascending reordering strategy, the first lower upper bound, the local upper bound, and the saving technique. The second algorithm, called the heuristic algorithm, is exactly the same as the first algorithm except that it uses the heuristic local upper bound. The first algorithm guarantees to find the optimal solution but can take a longer time. The second algorithm requires less time and finds a near optimal solution or the actual optimal solution.

3.10.3 Other Data Sets

We applied the two algorithms chosen in the previous subsection to other data sets. Table 3.5 shows the results of these experiments. The first two experiments on the IRIS data set are repeated from the previous subsection. These experiments show how the heuristic algorithm outperforms the complete algorithm in all measures except that it finds a slightly suboptimal solution.

The next two experiments are on the SPSS Bank data set which is described in Subsection 3.10.1. We saved 65 branches in the complete algorithm. This number of branches was enough to find the optimal solution as fast as possible. Because the heuristic algorithm performed very well on the IRIS data set without saving any branches, we did not save any branches in the heuristic algorithm on the Bank data set. These two experiments show the superior performance of the heuristic algorithm over the complete algorithm on all performance measures. The heuristic algorithm reduces the time from

Table 3.5: Branch and bound algorithms' results on other data sets.

Data Set	Algorithm	Time (Error)	Nodes	Solutions	Optimal
IRIS K=15	Comp. t=50	540	4,787	559	19
	Heuristic	146 (0.04%)	1,138	284	6
Banks K=15	Comp. t=65	21,808	81,936	5,319	92
	Heuristic	72.26	147	978	33
Banks K=30	Comp. t=100	30,600* (0.02%)	41,916	40	10
	Heuristic t=100	28,800*	30,411	1,103	559

*did not converge

about 6 hours for the complete algorithm to only 1.2 minutes and finds the optimal solution (i.e., there is no error). We must note that this performance must be due to the match between the heuristic local upper bound condition and the real cluster structure of this data set. This data set must have a cluster structure in which the clusters are separated in a way that satisfies the 1 cluster penalty condition of the heuristic local upper bound. We expect this algorithm to have the same performance for all data sets that have a similar cluster structure.

Because getting real current data sets is difficult, to test our algorithms on data sets that have a large number of samples we divided the observations in each sample in the Bank data set sequentially into two samples to get a data set of $K = 30$ samples. The last two experiments in Table 3.5 use this data set. Because preliminary experiments showed the difficulty of this data set, we saved 100 branches in both algorithms. Neither of the two algorithms converged and we had to stop their execution. This performance can be due to

the increase in the number of samples and the reduction in the separability of the real cluster structure of the data set because of the division of the samples. Although the heuristic algorithm was run for a shorter time, it found a better solution than the complete algorithm, which had an error of 0.02% compared to the heuristic algorithm solution. This performance can be explained by the fact that the heuristic algorithm pruned many branches that the complete algorithm had to enumerate and hence the complete algorithm did not have enough time to reach a better solution.

The last two experiments of Table 3.5 show that the branch and bound algorithms can not find the optimal solution for all kinds of data sets in a reasonable amount of time, although it performed very well for certain types of data sets. Therefore, in the next chapter we try to test how a genetic algorithm performs on all of the tested data sets.

3.10.4 Upper Bound Improvement Charts

Figures 3.7-3.12 show how the general upper bound value improves over time for each of the experiments in Table 3.5. These figures show that in a few seconds all algorithms find a solution with an objective function value that is very close to the best found solution's objective function value (the optimal value in case of convergence). This performance is due to the adaptive branching and to the fact of having many solutions close in objective function value to the optimal solution.

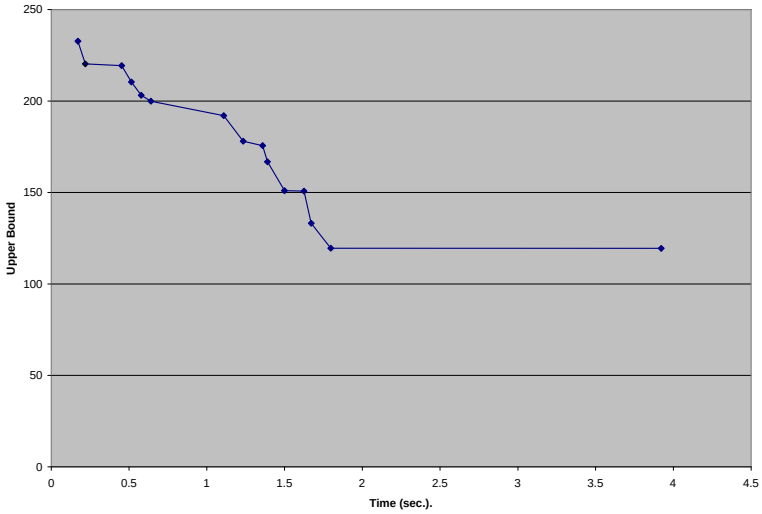

Figure 3.7: Upper bound improvement using the complete algorithm on the IRIS data set.

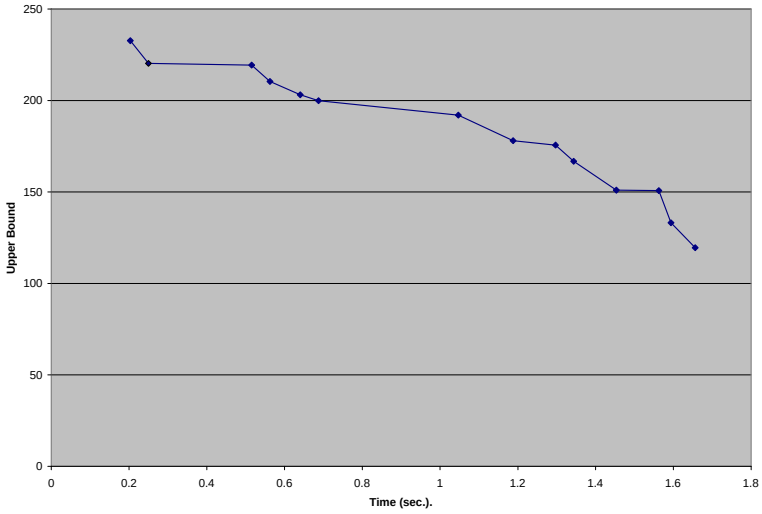

Figure 3.8: Upper bound improvement using the heuristic algorithm on the IRIS data set.

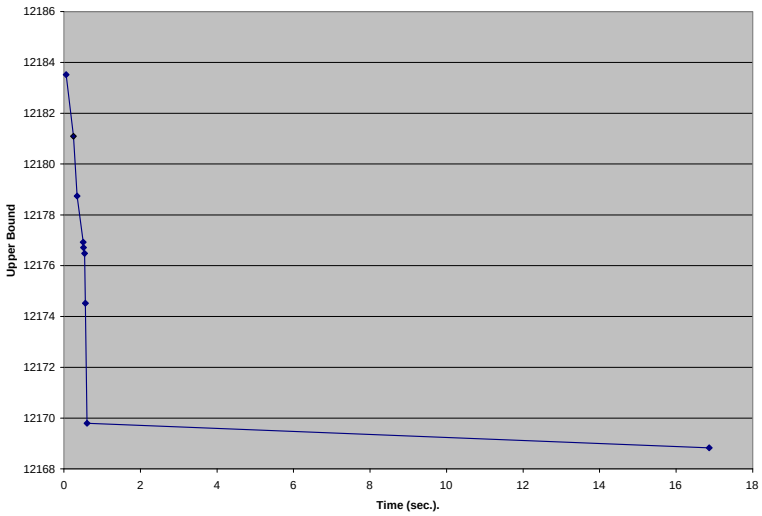

Figure 3.9: Upper bound improvement using the complete algorithm on the Bank (15 samples) data set.

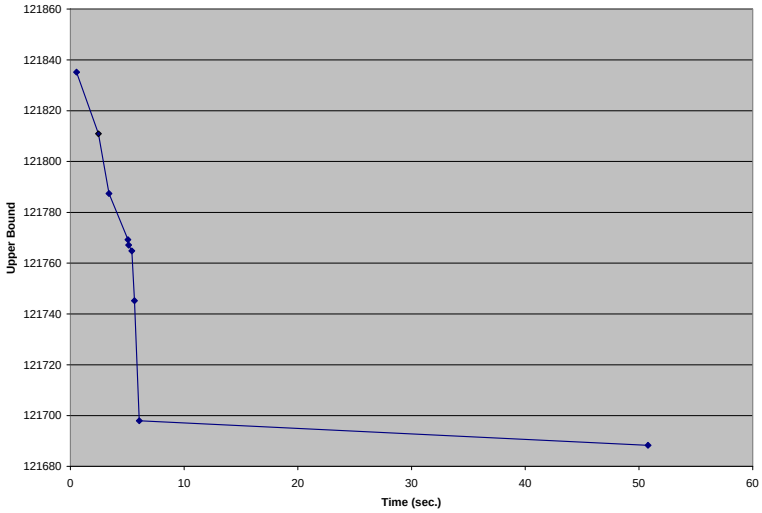

Figure 3.10: Upper bound improvement using the heuristic algorithm on the Bank (15 samples) data set.

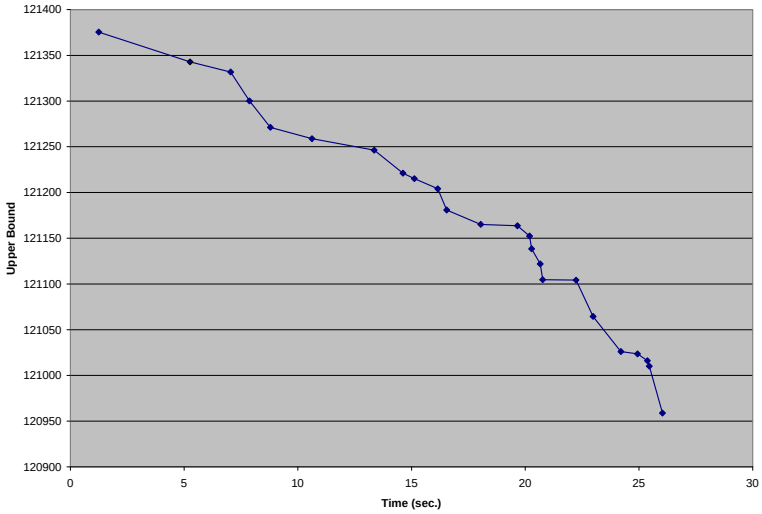

Figure 3.11: Upper bound improvement using the complete algorithm on the Bank (30 samples) data set.

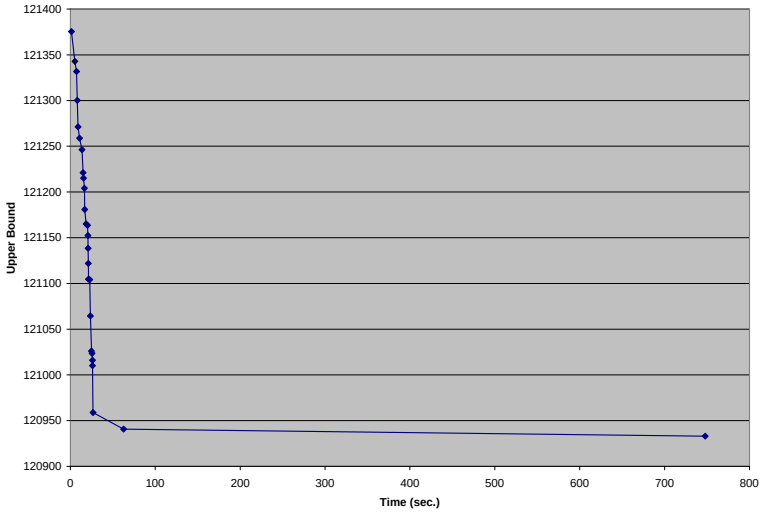

Figure 3.12: Upper bound improvement using the heuristic algorithm on the Bank (30 samples) data set.

3.11 Computational Remarks

We have presented a number of bounding and branching strategies and their expected and real performances. Here we consider some pure computational aspects of the algorithms that can affect their performance but are not based on any kind of theoretical analysis.

3.11.1 Sequential Branching

After we perform several steps of the adaptive branching strategy, we can branch sequentially and evaluate the bounds after a reasonable number, thereafter called the saving number, of steps of sequential branching. Depending on the performance of the bounding strategies, we can decrease or increase the saving number. This strategy will save a lot of computations because the bounding computations that are more likely not to lead to a pruning decision will not be carried out. If a better upper bound (feasible or complete solution) has been found, we can reactivate the adaptive branching strategies for a few steps. The use of this strategy is motivated by the fact that there are a much greater number of partial solutions than complete solutions to be evaluated. Therefore, using the sequential branching strategy until enough assignments have been made before trying to prune is a valid approach.

3.11.2 The A Matrices

The information criteria requires the computation of a number of matrices and a number of determinants. The determinants' computation is the most expensive type of computation and its cost depends on the dimensions of the

problem. We can save the A matrix or its determinant for each partial cluster and update it only when the content of this partial cluster changes. This technique is similar to the saving technique and is expected to improve the times of execution of the algorithms.

3.12 Conclusions and Future Work

We developed and tested a series of branch and bound strategies and algorithms that improve on each other. We showed how the complete adaptive branching strategy, with the ascending reordering strategy, outperforms the adaptive and sequential branching strategies by achieving the advantages of both the ascending and descending reordering strategies: fast convergence and fast finding of the optimal solution. The complete adaptive branching strategy does not require any extra computations because it only saves the enumerated branches of the enumeration tree to explore the next most promising branches of the tree.

We also showed the good performance of the first lower bound and the local upper bound. These bounds use the properties of the information criteria to prune the partial solutions. The second lower bound and the new lower bound did not perform as well and were dropped from the algorithms.

A heuristic was derived from the local upper bound that showed a superior performance. The heuristic local upper bound condition relaxes the local upper bound condition but does not guarantee finding an optimal solution. In most of the experiments conducted, this heuristic makes the algorithm converges in a fraction of the time required by the local upper bound. This performance

depends highly on the real cluster structure of the considered data set.

Some data sets that have a very large number of samples and/or poorly separated clusters can cause the complete and the heuristic algorithms to require a very long time to converge. However, using the complete adaptive branching strategy with a large number of saved branches will give us a high opportunity of finding the optimal or near optimal solution even in the case of stopping the algorithm before convergence. In many cases of poorly separated clusters there are many good clustering alternatives that have an information criterion value that is very close to the optimal solution information criterion value and using any of these solutions is widely acceptable.

In the complete adaptive branching strategy, after saving the allowed number of branches, the algorithm backtracks sequentially to the previously created branch when the current branch is completely enumerated. A possible improvement is to backtrack to the most promising branch rather than backtracking sequentially. A promising branch can be the one that has the complete solution with the minimum information criterion value among the remaining branches. This improvement will make it more likely to find the optimal solution faster than backtracking sequentially because of two possible reasons. First, we might have missed the branch leading to the optimal solution by making a suboptimal branching decision at a lower-level node in the first enumerated branch. Second, we might have created all of the saved branches from higher-level nodes in the enumeration tree that have the minimum first lower bound but do not contain the optimal solution.

We saw that the heuristic local upper bound can make the algorithm find a near optimal or the optimal solution a lot faster than the local upper bound.

This means that the local upper bound is too conservative and enumerates many bad solutions. This also means that the heuristic local upper bound can converge to a solution that is suboptimal. Future research can look into the possibility of finding a bound that is less conservative than the local upper bound and more accurate than the heuristic local upper bound. A possible consideration is to compare the increase in the value of the lack of fit term after joining a partial cluster and a free sample to the penalty of 1 cluster multiplied by a factor between 1 and 2 but closer to 1. This strategy can still prune almost the same number of partial solutions as the heuristic local upper bound does but is more likely to find the optimal solution.

The saving technique can be extended to include the negative outcome of the local upper bound condition. This extended saving technique can work as follows: If the local upper bound condition for a partial cluster and a free sample is not met then we can save the indexes of the free sample and the samples in the partial cluster in a list of inequalities. When the branch and bound algorithm reaches another node it checks if the considered free sample and partial cluster are already in the list of inequalities. If this is the case, the algorithm does not have to evaluate the local upper bound condition again and proceeds to branch on this node. However, because the original saving technique did not improve the performance of the branch and bound, we may expect that this extended saving technique can cause a weaker performance. This can be due to the fact that we have a very long list of inequalities to check and then it may require less computation time to evaluate the local upper bound condition rather than searching through the list of inequalities.

Cluster analysis branch and bound algorithms in general are very suitable

for parallel computing techniques. Parallel computers can enumerate different branches of the enumeration tree and share the best upper bound found and the pruning decisions. This strategy is expected to save a lot of time.

Reducing the number of variables and scaling the variables is an essential part in many cluster analysis studies. We can use Factor Analysis to reduce the number of variables. This reduction in the number of variables will consequently reduce the computations required by reducing the sizes of the A matrices and the determinant computations. We can also scale the categorical variables because they are more likely to create poorly separated clusters. The choice of the method of scaling may depend on the user's assigned value for each categorical variable. Methods that perform scaling include Principal Component, Factor Analysis, and optimal scaling.

In the next chapter we explore the performance of a specialized genetic algorithm on the same set of MSCA problems

Chapter 4

Adaptive Clustering Genetic Algorithm With Re-initialization

4.1 Introduction

Because MSCA is an NP-hard problem and branch and bound algorithms may take a long time to guarantee finding an optimal solution for large problems, we considered developing a heuristic that can find a good solution quickly. Genetic algorithms (GAs) have been widely used for solving cluster analysis problems. A possible reason for using genetic algorithms for solving cluster analysis problems is that these problems are naturally structured in a genetic form. The good clustering alternatives (chromosomes) share some of the same good clusters (genes) of samples (objects). If a good cluster is added to a clustering alternative, the clustering alternative objective function value will

improve. A minor change in a clustering alternative (chromosome), like moving an object to a different cluster, can improve the clustering alternative.

Genetic algorithms use the principles of natural evolution and genetics to search for a good solution of the considered problem. They improve a group, or a population, of possible solutions, or chromosomes, of the problem by an evolution process that moves from one generation to the next. GAs are more valuable when the considered problem has many local optimal solutions and we need to search for the global optimal solution among these many solutions. In addition, GAs have the advantage of searching in multiple areas of the solution search space simultaneously and are not limited to one local search at a time. These algorithms are mostly used when analytical optimization methods fail to find the global optimal solution in a reasonable amount of time. Another advantage of using GAs is that they do not require any restrictions, like the monotonic conditions described in Subsection 1.2.8, on the objective function of the considered problem. There are two general objectives of the GA process. The first objective is the exploitation of the current good areas of the search space where a local or global optimum is possibly to be found. The second objective is the exploration of different areas of the search space other than the current ones. GAs generally start with a random generation of the first population of chromosomes. Then the evolution process proceeds by repeating 4 important steps:

1. Evaluation: evaluate the fitness of each member of the current population according to the selected objective function to identify the best solutions.

2. Selection: select the members, which will survive to the next generation,

of the population according to the fitness values. This step will make the best solutions survive to the next generation. Some of the methods of selection are the roulette wheel and tournament selection.

3. Crossover: A number of pairs of chromosomes are selected and combined in a certain way to produce a new chromosome or an off-spring. The objective of this operator is to join the good parts of the chromosomes into one chromosome to make a better solution. It resembles the natural mating of individuals of any population.

4. Mutation: Some of the members of the population are changed in a random way to produce the next population. The goal of this operator is to move the search out of the area where the current local optimal solution was found in order to search for a better local optimal solution or the global optimal solution in other areas.

This process is repeated for a certain number of times which is decided based on the performance of the genetic algorithm. There are a number of parameters that must be decided within each genetic algorithm based on their performance: evaluation criterion, selection criterion, crossover method, and mutation method. For more information on GAs and recommended parameters values, please see Wolsey (1998), Reeves and Rowe (2003), or Haupt and Haupt (2004).

All of the early research on clustering genetic algorithms use the UFL representation, as described in Subsection 2.1, or the permutation representation of the cluster analysis problem. All of these encoding schemes have the

redundancy problem, as described in Subsection 2.1.1, unless a renumbering algorithm is used (Cole 1998; Jones and Beltramo 1991).

Another problem with early clustering genetic algorithms is that they use the standard crossover and mutation operators (Falkenauer 1993). These operators do not consider the special structure of the cluster analysis problem. A good clustering alternative, solution, or chromosome must have good clusters that join similar objects. A standard crossover or mutation operator will most likely disrupt the formed good clusters because it is not context sensitive. Special types of GA operators must be designed to utilize the structure of the cluster analysis problem.

There has been many attempts to modify and improve the standard GA. Michalewicz (1992) pointed out that in many studies the practitioners modify the problem to fit the standard GA process by applying the standard binary representation and the standard crossover and mutation operators. The right course of action instead is to modify the GA to fit the problem. Cluster analysis problems have a special structure that need to be taken into account when designing the GA.

Falkenauer (1993) was the first to introduce a special GA, called Grouping Genetic Algorithm or GGA, that considers the context of the cluster analysis problem. In this algorithm, the crossover and mutation operators are applied to the clusters instead of the objects. The crossover operator copies the clusters from the parents to the off-springs. Shared objects between two clusters are taken out of their old clusters and joined to the new injected clusters. The mutation operator either creates or deletes a cluster randomly. However, GGA still uses a modified UFL representation which has the problems of redundancy

and variable length chromosome. Hruschka and Ebecken (2003) solved the variable length chromosome problem in the GGA with another modified UFL representation and avoided the redundancy problem by using a renumbering algorithm.

A new attempt to avoid the redundancy problem used the linked-list encoding scheme (Du, Korkmaz, Alhajj, and Barker 2005). In this scheme each object is represented by a node that holds an integer value which is the index of the next higher index object in the same cluster. The last object's node in a cluster holds its own index. However, only the standard crossover and mutation operators were used on this encoding scheme.

Another problem in standard genetic algorithms is the use of static parameters' values (Pal and Wang 1996). A number of modifications are possible to link the parameters' values to how the GA process is performing in previous generations. For example, the mutation probability could be changed dynamically to get a balance between the exploitation and exploration objectives of the GA at different stages of the GA process. A possible modification to standard GA is to make the mutation probability depend on the current generation performance.

We did not find any genetic algorithm developed specifically for solving the MSCA problem using information criteria. In addition, the Clique Partitioning Problem (CPP) formulation was not used in any of the GA algorithms that we surveyed. In the next section we present a genetic algorithm that joins many nice features to solve the MSCA problem using both the information criteria and the CPP formulation.

4.2 The Genetic Algorithm

4.2.1 Overview

The GA we are proposing includes many special features that are expected to improve the performance of the GA in solving the MSCA problem. Our algorithm is called an adaptive clustering genetic algorithm with re-initialization, also can be called a repetitive adaptive clustering GA. We use the information criteria (derived by Bozdogan (1981, 1986)) as the objective function to find the best clustering alternative of the samples. The number of clusters is not fixed as it is in many clustering GAs. The GA we are proposing uses, for the first time, the Clique Partitioning Problem (CPP) formulation as its encoding scheme to avoid the problems faced by previously used representations. We use the pair-wise closeness of the samples as found by the information criteria to guide the random initialization process to find good initial solutions. We also utilize crossover and mutation operators that are specially designed for clustering problems. The mutation probability is not static but adapts to the current performance of the GA process. We also use the elitism technique to insure that the best solution among the current generation survives to the next generation. Finally, we use a re-initialization step when all the previous steps fail to move the search out of a local optimum.

We explain each step of the GA fully in the following subsections. We recommend certain values for the parameters needed in each step in the GA in the last subsection.

4.2.2 Encoding

We use the CPP formulation, as explained in Subsection 2.3, as the encoding scheme of our GA. This encoding scheme does not have the redundancy problem and hence does not require a renumbering algorithm as do most encoding schemes (e.g., the UFL as explained in Subsection 2.1). Therefore, each clustering alternative has exactly one representation in this encoding scheme. This feature reduces the search space for the GA tremendously. The CPP encoding scheme also easily allows the use of context-sensitive crossover and mutation operators as will be explained later. It also easily allows for the guided random initialization of the GA as we compare the pair-wise closeness of the samples, which will be explained in the next subsection.

4.2.3 Guided Random Initialization

The GA begins by generating an initial population with a given population size n_p. The initialization process begins by assigning to each decision variable x_{ij} of the CPP formulation a decision probability s_{ij} at which this decision variable is assigned a value of 1 (i.e., the samples i and j are joined in one cluster). These probabilities are developed in a way to guide the random initialization to find good initial solutions. Each s_{ij} probability is the sum of a given fixed probability p_f and a multiplication of a pair-wise cumulative probability pc_{ij} and a given fixed extra probability p_e,

$$s_{ij} = p_f + (pc_{ij} \times p_e).$$

The pair-wise cumulative probability is calculated by a modified roulette wheel method. First, we develop the pair-wise clustering alternatives by joining every two samples at a time and leaving each of the other samples in its own cluster (i.e., we set each $x_{ij} = 1$ at a time and leave the other decision variables equal to 0). Then, we compute the value of the lack of fit term for each of the pair-wise clustering alternatives. We subtract each of the lack of fit values from a very large positive number to get a fitness value f_{ij}. This step is done to avoid the possibility of having negative values. We use these fitness values to find the density probability of each pair of samples p_{ij} by dividing the pair-wise fitness value by the sum of the fitness values,

$$p_{ij} = \frac{f_{ij}}{\sum_{h=1}^{K-1} \sum_{g=h+1}^{K} f_{hg}}.$$

Then we reorder these probabilities in an ascending order t and calculate the pair-wise cumulative probabilities by summing all previous probabilities in this order,

$$pc_{ij} = p_{ij}^{t} + \sum_{u=1}^{t-1} p_{hg}^{u}.$$

This method will make the x_{ij} with the largest pair-wise lack of fit value (well separated samples) have the smallest pair-wise cumulative probability pc_{ij} and the smallest decision probability s_{ij}. It will also make the x_{ij} with the smallest pair-wise lack of fit value (closest two samples) have the largest pair-wise cumulative probability $pc_{ij} = 1$ and the largest decision probability $s_{ij} = p_f + p_e$.

For each chromosome of the initial population we randomly set its x_{ij}'s

values to 1 or 0 using the decision probabilities s_{ij}'s. Some of these random assignments will violate the transitive property or the triangle constraints of the CPP formulation. A recovery process is required to fix this problem. We randomly select a sample and fix its cluster as given by the random assignments by taking out its cluster members from other clusters. Then we repeatedly go randomly to another unfixed sample and fix its cluster in the same way until each sample is assigned to exactly one cluster.

There is a trade-off in selecting the size of the population. A large population size enables the GA to search in many areas in the search space which increases the probability of finding better solutions. However, a large population size also increases the computation time. Recommending a certain population size depends on the required performance and the allowed time.

4.2.4 Roulette Wheel Selection

We use the roulette wheel selection method. This method works as follows: We evaluate the fitness function of each chromosome in the current population as we did in the initialization process. We also calculate the cumulative probability of each solution as we did in the initialization process but using the information criteria and not the lack of fit term alone. We do not perform the reordering step used in the initialization process. Then we generate a random number n_p times. Each random number will fall between the cumulative probabilities of two solutions. We select the solution, which will survive to the next generation, that has the largest cumulative probability among these two solutions.

4.2.5 Crossover

Our crossover operator is specially designed for clustering problems. It closely follows Falkenauer's (1993) crossover operator but is essentially different by giving the user the ability to manipulate the result of the crossover to produce fewer or more clusters. The crossover probability p_c specifies how many of the solutions in the current population will go through the crossover operation. For each chromosome of the population we generate a random number. If this random number is less than or equal to p_c, then this chromosome is selected for crossover. If the number of selected chromosomes is odd, we disregard the last selected chromosome to get an even number of selected chromosomes. We randomly join the selected chromosomes in pairs. Each pair of chromosomes generates two off-springs. For each pair a random number r between 0 and 1 is generated and converted to a percentage. In the beginning, the first off-spring will be a copy of the second parent whole chromosome and the second off-spring will be a copy of first parent whole chromosome. Then the first r of the clusters of the first parent are copied to the first off-spring and the last $1 - r$ of the clusters of the second parent are copied to the second off-spring. The clusters in any chromosome are ordered according to the lowest sample index that each cluster contains. The off-springs will replace the selected chromosomes in the population.

When copying a cluster to an off-spring there are two choices. The first choice is to split the objects of the copied cluster from the off-spring's clusters and join them in one cluster (i.e., force the exact cluster into the off-spring). Hence, the off-spring will have the same or a greater number of clusters. The

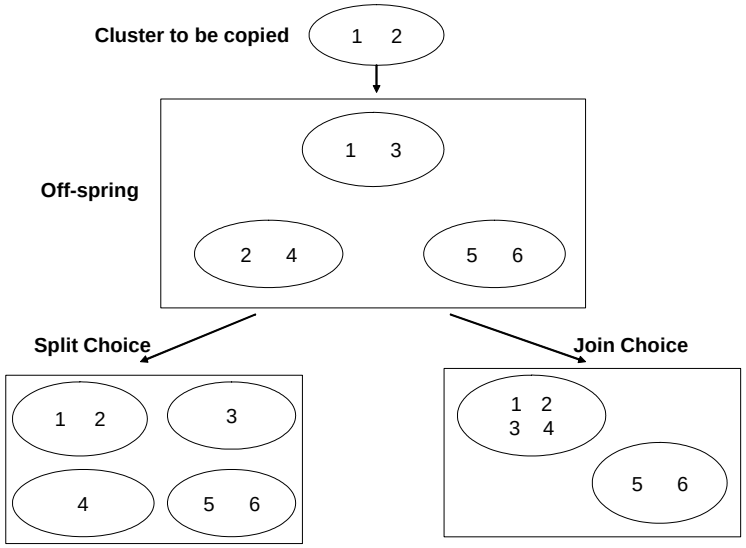

Figure 4.1: An example of a split and a join crossovers.

second choice is to join the clusters in the off-spring that contain the objects of the copied cluster. Hence, the off-spring will have the same or a smaller number of clusters.

These two choices are explained in Figure 4.1. In this figure the split choice forced the $(1, 2)$ cluster into the off-spring by taking them out of their clusters while the join choice joined the clusters of samples 1 and 2 into one cluster $(1, 2, 3, 4)$.

We define the split probability p_s in which the user specifies the probability of using the split choice, and consequently the join choice, for each crossover. This parameter is used to guide the GA to produce the good solutions that have few or many clusters as desired. However, both of the split and join choices can produce better clusters but only experimentation can show the superiority of using one over the other.

4.2.6 Adaptive Mutation

We use a different mutation operator than the one used by Falkenauer (1993). Falkenauer's (1993) mutation operator is applied directly to the clusters by deleting or creating a cluster. This kind of mutation operator can disrupt good clusters completely. Our mutation operator is applied to the clusters through the samples. According to a mutation probability, we move a sample to another cluster or a new cluster. This way we do not dramatically disrupt the content of any cluster and achieve the basic goal of the mutation operator of a random minimal change in the chromosome.

Adaptive mutation is not a new technique (Pal and Wang 1996). However, we link the value of the mutation probability to the improvement in the information criterion value across consecutive generations. In the beginning we specify a small mutation probability p_{ml}. This small probability will avoid losing good clusters quickly in the early generations and give time for the selection and crossover operators to find and exploit these good clusters. After a certain number, called the convergence number C, of consecutive generations pass without improvement in the best solution information criterion value, we use a very high mutation probability value p_{mh} for one generation and then reuse the small mutation probability value p_{ml}. The goal of the one generation use of a high mutation probability is to try to force the GA to move out of a local optimum in order to find better solutions.

4.2.7 Elitism

We use the elitism technique. This technique copies the best chromosome found so far in the GA process to the next generation regardless of the outcome of the selection, crossover, mutation, and other operators. This technique insures the survival of the best solution and the use of its good clusters to find better solutions.

4.2.8 Re-initialization

The idea of re-initializing the population during a GA execution has been used previously, and it has been shown that this strategy improves the GA performance (Koumousis and Katsaras 2006). However, we link the use of the re-initialization step with the improvement of the best solution value in the same way we did in the adaptive mutation operator. After C consecutive generations pass without improvement in the best solution value we use the adaptive mutation step as described previously. When another C consecutive generations pass without any improvement in the best solution value we re-initialize the population in the same way we did in the beginning of the GA. The best solution found so far is still copied to the next generation according to the elitism technique. Afterwards, whenever C consecutive generations pass without any improvement in the best solution value we alternate between the adaptive mutation step and the re-initialization step.

The goal of the re-initialization step is to replace the exhausted population with a fresh population that may contain better clusters that have not been found by the GA operations on the previous population. Also, the guided

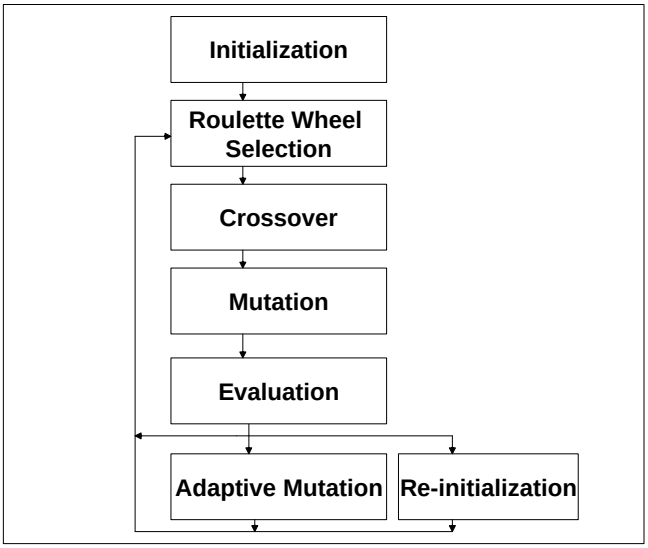

Figure 4.2: The GA flow chart.

random re-initialization may introduce new good solutions that will guide the GA search to even better solutions.

A flow chart that outlines the steps of our repetitive adaptive genetic Algorithm is shown in Figure 4.2. This is close to a typical genetic algorithm flow chart with the exception of modifying the evaluation process to make the decision of continuing with the low mutation probability, switching to the high mutation probability for the next generation, or re-initializing the population.

4.2.9 Recommended Parameters' Values

GA performance depends on the values we use for all the parameters that control the GA operators: population size, probability of crossover, probability of mutation, etc. It is difficult to find the optimal GA parameters' values. In fact, there may be different optimal parameters' values for different instances

Table 4.1: Recommended values of GA parameters.

Parameter	**Value**
n_p	150
p_f	0.3
p_e	0.4
p_c	0.9
p_s	0.9
p_{ml}	$\frac{0.225}{K}$
p_{mh}	0.5
C	25

of the problem. However, after preliminary experimentation with many different combinations of the parameters' values, we recommend the use of the parameters' values shown in Table 4.1.

The number of generations n_g depends on the problem size, which is a function of the number of samples, the number of observations per sample, and the number of variables. However, we can always set the number of generations to a high number and stop the GA when it takes a long time. Then we can use the best solution that was found.

The relatively high population size of 150 is needed to allow the GA to find good initial solutions and enough chromosomes to apply the GA operators to find better solutions without taking too much time per generation. The fixed initialization probability p_f of 0.3 gives a chance for the x_{ij} of well separated samples to have a value of 1 because there is always a possibility of having a good cluster that combines these samples when they are close to the same samples. The extra initialization probability p_e of 0.4 gives the samples that are close to each other up to $0.3 + 0.4 = 0.7$ probability of being joined in one cluster.

The high crossover probability is recommended in the literature (Haupt and Haupt 2004; Pal and Wang 1996) and has shown good performance in our experiments. The high split probability shows the superiority of copying the exact cluster in a crossover rather than joining the clusters that contain the members of the copied cluster. Because the mutation operator is applied to each sample, the initial low mutation probability is not fixed but is equal to a low probability of 0.225 divided by the number of samples K. If the initial low mutation probability was fixed and we have many samples, it will be very likely that a good solution become highly disrupted. Therefore, dividing this probability by the number of samples will avoid this situation. The high adaptive mutation probability p_{mh} serves the goal of adaptation, moving out of a local optimum, by trying to find better clusters in the population through many random movements of the samples to different or new clusters.

After experimentation we found that running the GA for 25 generations will give a reasonable time for the GA operators to find the best solution of a population and is the best time to use the adapted mutation probability or the re-initialization step. Next, we will show the results of the experiments where we used these parameters' values on problems of different sizes.

4.3 Experimental Results

As we did with the branch and bound algorithms, we first test the GA on the well known IRIS bench mark data set to show the performance of some of the important strategies. Then we test the best GA strategies on other data sets.

Table 4.2: Results of experiments using GA strategies.

Algorithm	Average Error	Average Time (sec.)
Basic Clustering GA	14.5%	3,050
Adaptive Clustering GA	12%	3,266
Repetitive Adaptive Clustering GA	5.4%	3,183

4.3.1 IRIS Data Set

Because the IRIS data set has a small total number of observations of 150, each generation of the GA will not need a long execution time. Therefore, we used a high number of generations of 350. We made three experiments. In the first experiment we used the basic GA without the adaptive mutation or the re-initialization steps. However, we use the fixed small mutation probability in the basic GA. Then, only the adaptive mutation step was used in the second experiment. Finally, both of the adaptive mutation and re-initialization steps were used, as described in the previous subsection, in the third experiment. For each experiment we ran the GA 10 times to avoid the effect of the random initialization.

The results of the experiments are shown in Table 4.2. There is a row for each experiment. The first column specifies the algorithm used in the experiment. The second column shows the average relative error of the 10 runs of the experiment between the best solution value found by the GA and the optimal solution value found by the branch and bound algorithms presented in Chapter 3. The last column reports the average execution time, in seconds, of the 10 runs of the experiment.

In the first experiment we ran the basic GA without the adaptive mutation and the re-initialization steps but with the initial low mutation probability.

The basic GA has a relatively high average error rate and execution time. The high error rate can be due to the low mutation probability of $\frac{0.225}{15} = 0.015$. This probability may not allow the GA to search in many areas of the search space. The high execution time can be due to the fact of having a relatively high population size, high crossover probability, and high number of generations.

In the second experiment only the adaptive mutation step is added. We see that the average error rate decreased by a higher percentage than the increase in the average execution time. This improvement can be due to the fact that we did not increase the mutation probability for all generations and hence the execution time did not increase by much. Also, we periodically allowed for a high mutation probability to find better search areas and give the GA the time to search in these areas.

The last experiment shows the large improvement in the average error rate after using both the adaptive mutation and the re-initialization steps. This improvement is achieved without an increase in the average execution time. In fact the execution time actually decreases. The decrease in the average error rate is due to the ability of the re-initialization step to find more good clusters. The decrease in the average execution time can be due to the fact that the adaptive mutation step can sometimes require more time than the re-initialization step and that the re-initialization step can reduce the number of times the adaptive mutation and the re-initialization steps are used by finding better solutions more often. The adaptive mutation step, as explained in Subsection 4.2.6, involves the search for the original cluster of the sample to be moved and the clusters that this sample can move to and then a recovery

process after moving this sample. The re-initialization step, as explained in Subsection 4.2.3, involves a straightforward formation of the clusters of each solution according to the initialization probabilities. Therefore, we recommend the use of the repetitive adaptive GA in the next experiments on other data sets.

We also include a graph of the improvement in the best solution value over the generations for one of the runs of the repetitive adaptive algorithm and the basic algorithm in Figure 4.3 and Figure 4.4 respectively. We choose the best repetitive adaptive GA run in order to make the difference obvious. We choose the basic GA run in which the value of the best solution in the first generation is closest to the value of the best solution in the first generation of the chosen repetitive adaptive GA run. Both graphs show how the best solution value improves very quickly in the first few generations of the GA. We can also see that the very long periods of no or minimal improvement start much later in the repetitive adaptive GA run than in the basic GA run.

4.3.2 Other Data Sets

We applied the repetitive adaptive clustering GA to two data sets: the Bank data set with $K = 15$, described in Subsection 3.10.1, and to the Bank data set with $K = 30$, described in Subsection 3.10.3. Because of the large number of observations in the case of the first data set and the large number of samples in the second data set we limited the number of generations to 250 to make the execution time reasonable. We made 10 runs for each data set. The errors in these experiments are computed in comparison to the optimal or best solutions

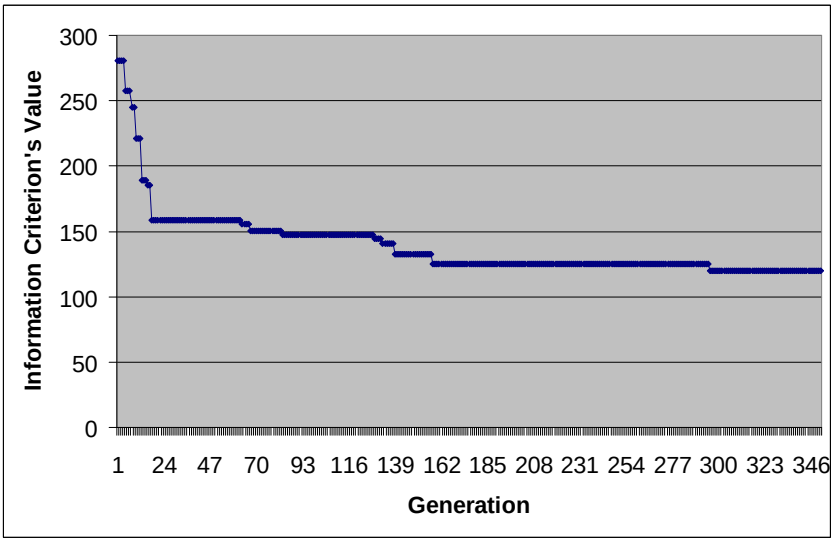

Figure 4.3: One run of the repetitive adaptive GA.

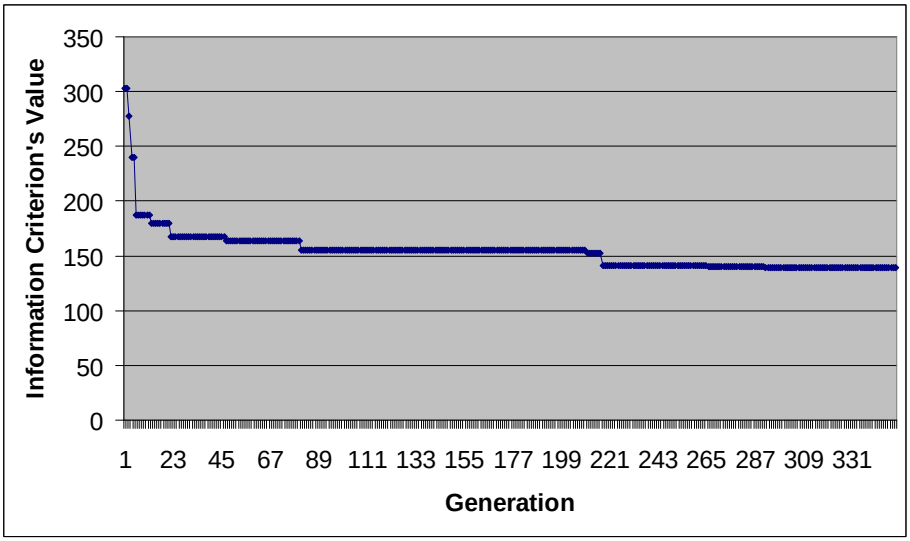

Figure 4.4: One run of the basic GA.

Table 4.3: Results of experiments on other data sets using the repetitive adaptive clustering GA.

Data Set	n_g	Average Error	Average Time (sec.)
IRIS	350	5.4%	3,183
Bank $K = 15$	250	0.028%	3,456
Bank $K = 30$	250	0.26%	8,681

found by the branch and bound algorithms in the experiments presented in Subsection 3.10.3. The results of these experiments are shown in Table 4.3.

We see that for both data sets the error is less than 1%. However, the execution times increased in comparison to the IRIS data set experiment. For the bank data set with 15 samples the execution time slightly increased because of the large number of observations (1500). For the bank data set with 30 samples the execution time increased a lot. This huge increase shows that one of the major factors in increasing the clustering GA execution time is the number of samples.

We see that even though the GA combines many of the best features in the GA field it needs a long time to find a solution that is close to the solution found by the complete adaptive branch and bound algorithm. However, we can always run the GA for few generations in few minutes and get a good enough solution as shown in the graphs of the best runs for these two experiments in Figure 4.5 and Figure 4.6. These figures again show that the best solution value decreases very quickly in the first few generations and then has periods of no improvement of increasing length.

For these two data sets we can see that there are a lot of solutions that have close information criterion values to the best found solution value. This tell us that there are a lot of good solutions that are close to the optimal

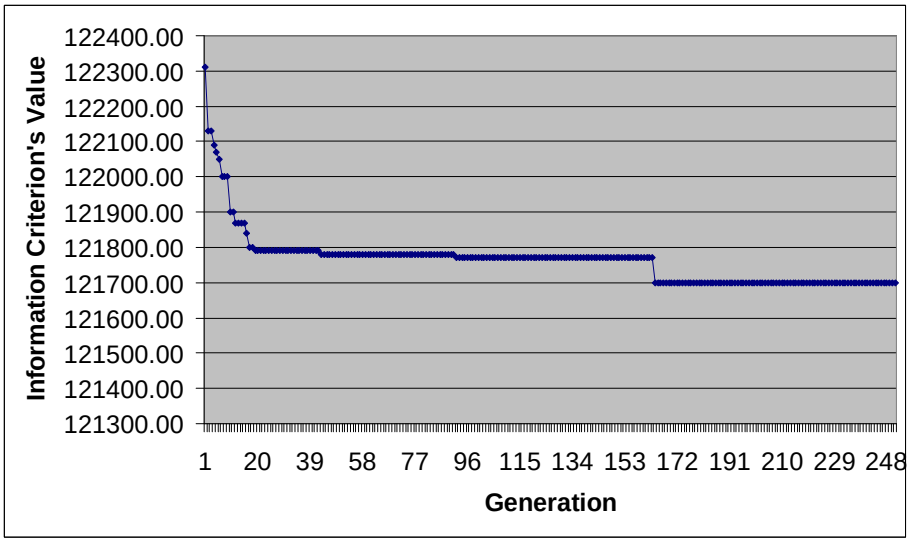

Figure 4.5: Best repetitive adaptive clustering GA run on the Bank data set (15 samples).

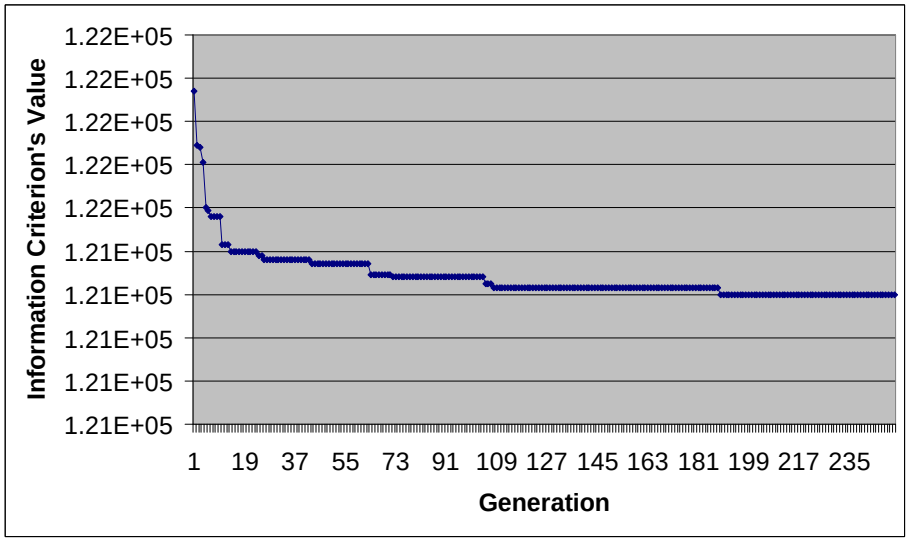

Figure 4.6: Best repetitive adaptive clustering GA run on the Bank data set (30 samples).

solution in the information criterion value. If a good solution is acceptable, we can stop the GA algorithm after fewer generations than we did in these two experiments and get a good enough solution.

4.3.3 Simulation Experiment

Here, we use the branch and bound algorithms and the genetic algorithm to cluster a simulated data set. The simulated data set has the same dimensions as the IRIS data set (150 observations and 4 variables). We chose to create a simulated data set with moderately separated clusters. The real cluster structure of the simulated data set consists of 3 groups of 50 observations each. We divided each of the 3 groups into 5 samples of 10 observations each to get a total of 15 samples. The mean and the variance-covariance matrix of each of the 3 groups are as follows.

$$\mu_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \Sigma_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$\mu_2 = \begin{bmatrix} 4 \\ 4 \\ 4 \\ 4 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 3 & 0.5 & 0.5 & 0.5 \\ 0.5 & 3 & 0.5 & 0.5 \\ 0.5 & 0.5 & 3 & 0.5 \\ 0.5 & 0.5 & 0.5 & 3 \end{bmatrix},$$

Table 4.4: Branch and bound algorithms' results on the simulated data set.

Data Set	Algorithm	Time (Error)	Nodes	Solutions	Optimal
Simulated	Comp. t=50	502	5,395	100	8
	Heuristic t=50	380	4,157	100	8
Optimal Clustering Alternative (1,2,4,5)(3)(6,7,8,9,10)(11,12,13,14,15)					

$$\mu_3 = \begin{bmatrix} 10 \\ 10 \\ 10 \\ 10 \end{bmatrix}, \Sigma_3 = \begin{bmatrix} 5 & 1 & 1 & 1 \\ 1 & 5 & 1 & 1 \\ 1 & 1 & 5 & 1 \\ 1 & 1 & 1 & 5 \end{bmatrix}.$$

Figures 4.7-4.9 and Tables 4.4-4.5 show the results of the simulation experiment. The complete branch and bound algorithm performed well. The heuristic algorithm improved performance but not as good as it did on the IRIS data set. Both algorithms found the optimal solution, which consists of 4 clusters, and has an information criterion value of 2,197. The optimal solution assigns sample 3 to a cluster by itself. The real clustering alternative, which consists of 3 clusters, has an information criterion value of 2,200, which is very close to the optimal value. As shown in the upper bound improvement charts, these algorithms find solutions with an information criterion values that are close to the optimal value in a few seconds. The genetic algorithm on the simulated data set has a good performance. The best found information criterion value improves quickly in the first few generations of the GA. Then, the GA takes a long time to find a better value.

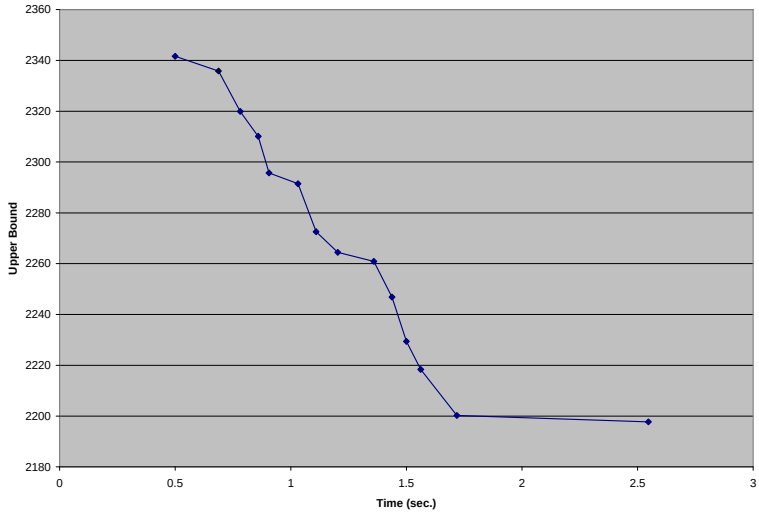

Figure 4.7: Upper bound improvement using the complete algorithm on the simulated data set.

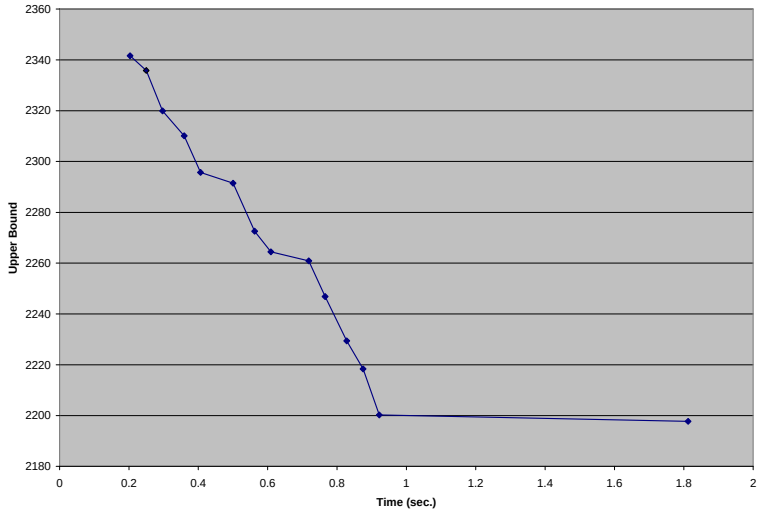

Figure 4.8: Upper bound improvement using the heuristic algorithm on the simulated data set.

Table 4.5: Results of the experiment on the simulated data set using the repetitive adaptive clustering GA

Data Set	Average Error	Average Time (sec.)
Simulated	0.59%	2,907

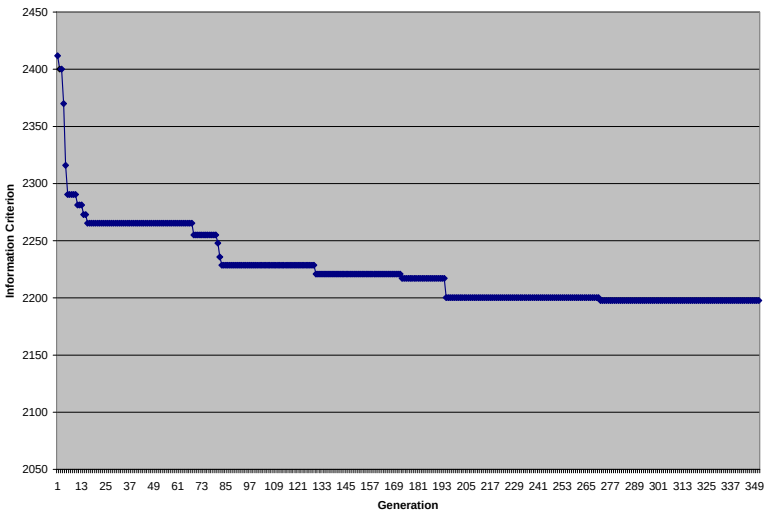

Figure 4.9: Best repetitive adaptive clustering GA run on the simulated data set.

4.4 Conclusions and Future Work

The adaptive clustering GA with re-initialization showed an improved performance over the basic GA. The CPP encoding has avoided the redundancy problem faced by most of the encoding schemes used previously for cluster analysis problems. It also enabled the use of the crossover and mutation operators that are specially designed for cluster analysis problems. The adaptive mutation operator has moved the GA more towards the balance between the exploitation and exploration goals. This has been done by using a low initial mutation probability that allowed the GA to exploit the good search areas found and then by using a one generation very high mutation probability to move the GA to different areas of the search space. The re-initialization step is more effective in doing the same job of the adaptive mutation operator because it uses the *guided* re-initialization process. The adaptive mutation and re-initialization steps combined with the CPP encoding and the clustering crossover and mutation operators contributed to this improvement in performance. The improved performance is evident by the very low error rates of the solutions found by this GA.

However, the GA takes a long time to find very good solutions. We found out that the complete adaptive branch and bound algorithm, either with the local upper bound or with the heuristic version of this bound, will find better solutions than this GA in the first few solutions found by the branch and bound algorithm. We can run the complete adaptive branch and bound algorithm and the GA for the same short time and the complete adaptive branch and bound algorithm will find a superior solution. This fact is easily explained by the

bounding and branching strategies used by the branch and bound algorithm to quickly sift through the most promising solutions according to the properties of the information criterion to find the optimal or near optimal solution. This sophisticated GA has helped us to verify the superiority of the complete adaptive branch and bound algorithm and can always be used to verify the superiority of the solution found by the branch and bound algorithm.

We recommend the use of the complete adaptive branch and bound algorithm to solve MSCA problems. The number of branches to save in this algorithm depends only on the memory limit of the computer used. We can use as many saved branches as long as we do not exceed the memory limit of the computer. The use of the heuristic version of the local upper bound depends on the user's acceptance of a near optimal solution in return for a shorter convergence time. Even when this algorithm is stopped before convergence it will find a near optimal solution, if not the optimal solution, as shown in the results of the experiments that we conducted.

The re-initialization step of the GA can be performed in different ways that may improve the performance of the GA. A possible way is not to copy only the single best solution found to the re-initialized population but to keep many of the best solutions found by the GA and re-initialize only a part of the population. This way, more good solutions are mixed with random solutions. This mix is more likely to find better solutions than having only the best found solution with a re-initialized population.

In the case of selecting an odd number of chromosomes to go through the crossover operation, we disregard the last selected chromosome. This technique can be improved by randomly deleting one of the selected chromosomes

to avoid disregarding always the chromosomes at the end of the population. However, the last selected chromosome was selected randomly and hence this random deletion technique may not improve the performance.

A possible direction of research is to combine the branch and bound algorithm with the GA to improve performance. A direct way of this hybridization is to start with a run of the GA and use the best solution objective function value found by the GA as the initial upper bound in the branch and bound algorithm. Another way is to start with the branch and bound algorithm and, if it does not converge in a short time, we can stop it and use the best solutions found as part of the initial population of the GA. However, because the complete adaptive branch and bound algorithm finds the optimal or near optimal solution much faster than the GA does, starting with the GA to find good initial solution is not expected to improve performance. In addition, because the complete adaptive branch and bound algorithm finds the optimal or near optimal solution, using the GA afterwards is not likely to find better solutions quickly. We can make this conclusion because our experiments show that after finding a very good solution, the GA needs a very long time to find a better solution. Therefore, a more sophisticated way of hybridization needs to be found in order to improve the performance of the branch and bound and the genetic algorithms.

A more promising direction of hybridization is to follow a few generations of the GA with a local search algorithm (Reeves and Rowe 2003). We need to develop a local search algorithm that is specially designed for the MSCA problem. The GA is very good in finding the most promising areas in the search space that may contain the global optimum. After finding these areas

the GA may not be the best way to find the optimal solution. A local search algorithm that uses the properties of the information criterion can quickly find the optimal solution starting from the best solutions found by the GA. This algorithm may try single movements of the samples, in the best solution found by the GA, to different or new clusters in order to improve this solution. It can also use the pair-wise closeness, used in the guided initialization in the GA, to develop these movements.

We need to test the developed algorithms and new ideas using different data sets. Every data set has its special real cluster structure. We need to find in what kinds of data sets a specific algorithm works best and in what kinds it needs improvements. More testing of the algorithms can also bring new ideas for better algorithms that can work for most of the problems. This testing can identify better combinations of the GA parameters' values that can improve the performance of the GA on specific kinds of the problem. We also need to adapt our algorithms to Multi-Sample Cluster-Wise Regression problems as described in 1.3.

Bibliography

Bibliography

Akaike, H. (1973). Information theory and an extension of the maximum likelihood principle. In B. Petrov and F. Csaki (Eds.), *Second international symposium on information theory*, Budapest, pp. 267–281. Academiai Kiado.

Bao, X., H. Bozdogan, V. Chatpattananan, and K. Gilbert (2005). An implicit enumeration algorithm for mining high dimensional data. *International Journal of Operational Research 1*(1/2), 123–144.

Beane, T. P. . and D. M. . Ennis (1987). Market segmentation: A review. *European Journal of Marketing 21*(5), 20–42.

Bijmolt, T. H. A., L. J. Paas, and J. K. Vermunt (2004). *Country and consumer segmentation: multi-level latent class analysis of financial product ownership*, pp. p. 323–340.

Bozdogan, H. (1981). *Multi-Sample Cluster Analysis and Approaches to Validity Studies in Clustering Individuals*. Ph. D. thesis, Department of Mathematics, the University of Illinois, Chicago Circle.

Bozdogan, H. (1986). Multi-sample cluster analysis as an alternative to multiple comparison procedures. *Bulletin of Informatics and Cybernetics Research Association of Statistical Sciences 22*(1–2).

Bozdogan, H. (1987). Model selection and Akaike's Information Criterion (AIC): The general theory and its analytical extensions. *Psychometrika V52*(3), 345–370.

Bozdogan, H. (1988). ICOMP: A new model-selection criterion. In H. H. Bock (Ed.), *Classification and Related Methods of Data Analysis*, Amsterdam, pp. 599–608. Elsevier Science Publishers B.V.

Bozdogan, H. (1990). On the information-based measure of covariance complexity and its application to the evaluation of multivariate linear models. *Communications in Statistics-Theory and Methods 19*, 221–278.

Bozdogan, H. (1994). Mixture-model cluster analysis using model selection criteria and a new informational measure of complexity. In H. Bozdogan (Ed.), *Multivariate Statistical Modeling, Vol. 2, Proceedings of the First US/Japan Conference on the Frontiers of Statistical Modeling: An Informational Approach*, Dordrecht, The Netherlands, pp. 69–113. Kluwer.

Bozdogan, H. (2004). *Statistical data mining and knowledge discovery.* Boca Raton, Fla.: Chapman and Hall/CRC. edited by Hamparsum Bozdogan.

Bozdogan, H. and D. Haughton (1998). Information complexity criteria for regression models. *Computational Statistics and Data Analysis 28*, 51–76.

Brusco, M. J. (2003). An enhanced branch-and-bound algorithm for a partitioning problem. *British Journal of Mathematical and Statistical Psychology 56*(1), 83.

Brusco, M. J. (2006). A repetitive branch-and-bound procedure for minimum within-cluster sums of squares partitioning. *Psychometrika V71*(2), 347–363.

Chopra, S. and M. R. Rao (1993). The partition problem. *Mathematical Programming V59*(1), 87–115.

Cole, R. M. (1998). Clustering with genetic algorithms. Master's thesis, University of Western Australia.

Cui, G. and Q. Liu (2000). *Regional market segments of China: opportunities and barriers in a big emerging market*, pp. p. 55–72.

Diehr, G. (1985). Evaluation of a branch and bound algorithm for clustering. *SIAM Journal on Scientific and Statistical Computing 6*(2), 268–284.

Dorndorf, U. and E. Pesch (1994). Fast clustering algorithms. *ORSA Journal on Computing 6*(2), 141.

Du, J., E. E. Korkmaz, R. Alhajj, and K. Barker (2005). Alternative clustering by utilizing multi-objective genetic algorithm with linked-list based chromosome encoding. *Lecture Notes in Computer Science 3587*, 346–355.

du Merle, O., P. Hansen, B. Jaumard, and N. Mladenovic (2000). An interior point algorithm for minimum sum-of-squares clustering. *SIAM Journal on Scientific Computing 21*(4), 1485–1505.

Duran, B. S. and P. L. Odell (1974). *Cluster Analysis: A Survey*. New York: Springer-Verlag.

Everitt, B. S. (1993). *Cluster analysis*. London: Arnold.

Falkenauer, E. (1993). The grouping genetic algorithms: widening the scope of the gas. *Belgian Journal of Operations Research, Statistics and Computer Science 33*(1), 79–102.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics VII part II*, 179–188.

Garfinkel, R. S. and Nemhauser, G. L. (1969). The set-partitioning problem: Set covering with equality constraints. *Operations Research 17*(5), 848–856.

Green, Paul E., Frank, Ronald E., and Robinson, Patrick J. (1967). Cluster analysis in test market selection. *Management Science 13*(8), B387–B400.

Grotschel, M. and Y. Wakabayashi (1989). A cutting plane algorithm for a clustering problem. *Mathematical Programming V45*(1), 59–96.

Hansen, P. and B. Jaumard (1997). Cluster analysis and mathematical programming. *Mathematical Programming 79*, 191–215.

Haupt, R. L. and S. E. Haupt (2004). *Practical Genetic Algorithms*. Wiley-Interscience.

Hofstede, F. T., M. Wedel, and J.-B. E. M. Steenkamp (2002). *Identifying spatial segments in international markets: a model accommodating various forms of spatial dependence on international segmentation*, pp. p. 160–177.

Hofstede, G. (1976). Nationality and espoused values of managers. *Journal of Applied Psychology 61*(2), 148–155.

Hofstede, Frenkel Ter, Steenkamp, Jan-Benedict E. M., and Wedel, Michel (1999, feb). International market segmentation based on consumer-product relations. *Journal of Marketing Research 36*(1), 1–17.

Hruschka, E. R. and N. F. F. Ebecken (2003). A genetic algorithm for cluster analysis. *Intelligent Data Analysis 7*, 15–25.

Hsu, J. C. (1996). *Multiple Comparisons Theory and methods*. Chapman and Hall.

Johnson, E. L., A. Mehrotra, and G. L. Nemhauser (1993). Min-cut clustering. *Mathematical Programming V62*(1), 133–151.

Jones, D. R. and M. A. Beltramo (1991). Solving partitioning problems with genetic algorithms. In *The Forth International Conference on Genetic Algorithms; University of California, San Diego*, pp. 442–449.

Kahle, L. R. (1986). The nine nations of north america and the value basis of geographic segmentation. *Journal of Marketing 50*(2), 37–47.

Kale, S. H. (1995). Grouping euroconsumers: A culture-based clustering approach. *Journal of International Marketing 3*(3), 35.

Klein, G. and J. E. Aronson (1991). Optimal clustering: A model and method. *Naval Research Logistics 38*, 447–461.

Koontz, W. L. G., P. M. Narendra, and K. Fukunaga (1975, 9). A branch and bound clustering algorithm. *IEEE Transactions on Computers 24*(9).

Koumousis, V. K. and C. P. Katsaras (2006). A saw-tooth genetic algorithm combining the effects of variable population size and reinitialization to enhance performance. *IEEE Transactions on Evolutionary Computation 10*, 19–28.

Kumar, V., A. Stam, and E. A. Joachimsthaler (1994). An interactive multicriteria approach to identifying potential foreign markets. *Journal of International Marketing 2*(1), 29.

Lilien, G. and A. Rangaswamy (1998). *Marketing Engineering*. Addison-Wesley.

Madsen, T. K. and S. Askegaard (1998). *The local and the global : exploring traits of homogeneity and heterogeneity in European food cultures*, pp. p. 549–568.

Marriott, F. H. C. (1982, aug). Optimization methods of cluster analysis. *Biometrika 69*(2), 417–421.

Mead, R. and D. Pike (1975). A review of response surface methodology from a biometric viewpoint. *Biometric 31*, 803–851.

Mehrotra, A. and M. A. Trick (1998). Cliques and clustering: A combinatorial approach. *Operations Research Letters 22*(1), 1–12.

Michalewicz, Z. (1992). *Genetic Algorithms + Data Structure = Evolution Programs*. Springer-Verlag.

Mittal, V., W. A. Kamakura, and R. Govind (2004). *Geographic patterns in customer service and satisfaction: an empirical investigation*, pp. p. 48–62.

Mulvey, J. M. and H. P. Crowder (1979). Cluster analysis: An application of lagrangian relaxation. *Management Science 25*(4), 329–340.

O'Neill, R. and G. B. Wetherill (1971). The present state of multiple comparison methods. *Journal of Royal Statistical Society 33*(218–241).

Pal, S. K. and P. P. Wang (1996). *Genetic Algorithms for Pattern Recognition*. CRC Press.

Palubeckis, G. (1997). A branch-and-bound approach using polyhedral results for a clustering problem. *INFORMS Journal on Computing 9*(1), 30.

Punj, G. and D. Stewart (1983). Cluster analysis in marketing research: Review and suggestions for application. *Journal of Marketing Research 20*(2), 134–148.

Rao, M. R. (1971). Cluster analysis and mathematical programming. *Journal of the American Statistical Association 66*(335), 622–626.

Reeves, C. R. and J. E. Rowe (2003). *Genetic Algorithms: Principles and Perspectives, A Guide to GA Theory*. KluwerAcademic Publishers.

Ronen, S. and A. I. Kraut (1977). Similarities among countries based on employee work values and attitudes. *Columbia Journal of World Business 12*(2), 89–96.

Ronen, S. and O. Shenkar (1985). Clustering countries on attitudinal dimensions: A review and synthesis. *The Academy of Management Review 10*(3), 435–454.

Schwarz, G. (1978). Estimating the dimension of a model. *Annals of Statistics 6*(461–464).

Sirota, D. and J. M. Greenwood (1971). Understand your overseas work force. *Harvard business review 49*(1), 53–60.

Steenkamp, J.-B. E. M. (2001). The role of national culture in international marketing research. *International Marketing Review 18*(1), 30.

Steenkamp, J.-B. E. M. and F. T. Hofstede (2002). International market segmentation: issues and perspectives. *International Journal of Research in Marketing 19*(3), 185–213.

Trotter, L. E. and C. M. Shetty (1974). An algorithm for the bounded variable integer programming problem. *J. ACM 21*(3), 505–513.

van Emden, M. (1971). *An Analysis of Complexity.* Mathematical Centre Tracts, Mathematisch Centrum Amsterdam, Amsterdam, 1971.

Vandermerwe, S. and M.-A. L'Huillier (1989). Euro-consumers in 1992. *Business Horizons 32*(1), 34–40.

Vinod, H. D. (1969). Integer programming and the theory of grouping. *Journal of the American Statistical Association 64*(326), 506–519.

Wakabayashi, Y. (1986). *Aggregation of Binary Relations: Algorithmic and Polyhedral Investigations.* Ph. D. thesis, University of Augsburg, Germany.

Wolsey, L. (1998). *Integer Programming.* Wiley-Interscience.

Xu, R. and D. Wunsch (2005). Survey of clustering algorithms. *IEEE Transactions on Neural Networks 16*(3), 645–678.

Vita

Fahad Almutairi was born in Kuwait on 10/10/1975. He earned his Bachelor's degree in computer engineering from Kuwait University in 1998 and his Master of Business Administration from the University of Memphis in 2002. Fahad completed the Ph.D. program in management science at the University of Tennessee at Knoxville in August 2007.