Variable Neighbourhood Search Based Heuristic for K-Harmonic Means Clustering

A thesis submitted for the degree of
Doctor of Philosophy

by

Abdulrahman Al-Guwaizani

Brunel
UNIVERSITY
LONDON

Department of Mathematical Sciences
School of Information Systems, Computing and Mathematics
Brunel University, London

Abstract

Although there has been a rapid development of technology and increase of computation speeds, most of the real-world optimization problems still cannot be solved in a reasonable time. Some times it is impossible for them to be optimally solved, as there are many instances of real problems which cannot be addressed by computers at their present speed. In such cases, the heuristic approach can be used. Heuristic research has been used by many researchers to supply this need. It gives a sufficient solution in reasonable time. The clustering problem is one example of this, formed in many applications.

In this thesis, I suggest a Variable Neighbourhood Search (VNS) to improve a recent clustering local search called K-Harmonic Means (KHM). Many experiments are presented to show the strength of my code compared with some algorithms from the literature.

Some counter-examples are introduced to show that KHM may degenerate entirely, in either one or more runs. Furthermore, it degenerates and then stops in some familiar datasets, which significantly affects the final solution. Hence, I present a removing degeneracy code for KHM. I also apply VNS to improve the code of KHM after removing the evidence of degeneracy.

Certificate of Originality

I hereby certify that the work presented in this thesis is my original research and has not been presented for a higher degree at any other university or institute.

Signed:________________________ Dated: _________________

Abdulrahman Al-Guwaizani

Acknowledgements

I present my best thanks to my God, whose response always helped me and given me the power to complete my work.

I would like to express my heartfelt gratitude to all those who made it possible for me to complete this thesis. I want to thank the Department of Mathematics for providing me with the most recent programs in my research, for giving me the permissions to access the relevant articles in my work and to use the departmental facilities.

I am deeply indebted to my supervisor, Dr. Nenad Mladenović, whose help, stimulating suggestions and encouragement have helped me throughout my research and the writing of this thesis. I am also grateful that he provided me with the FORTRAN code for Variable Neighbourhood Search (VNS) for the K-Means algorithm and the VNS extensions. Also, special thanks to Jasmina Lazić for providing me the classic heuristics and metaheuristics clustering codes with more details.

In particular, I would like to give my special thanks to my wife Areej, whose patient love enabled me to complete this work.

Author’s Publications

1. A. Alguwaizani, P. Hansen, N. Mladenović, and E. Ngai, *Variable neighborhood search for harmonic means clustering*, Applied Mathematical Modelling, 35 (2011), 2688-2694.

2. E. Carizosa, A. Alguwaizani, P. Hansen, N. Mladenović, *Degeneracy of harmonic means clustering*. (submitted to Pattern Recognition on 31-5-2011).

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Literature and Definitions

To keep up with the enormous strides made by science and technology, communities should deal accurately with the speedy transmission of information and data. Many countries have started to apply e-government systems. This is where the importance lies of analysing data and distributing and dealing with software applications. Clustering technology has become very important at present, especially with the increasing growth and steady fields of data analysis. It is applied in a variety of ways in the natural sciences, psychology, medicine, engineering, economics, marketing and other fields [75]. Scientists and researchers have not lost sight of the importance of clustering; tens of thousands of scientific papers have been published on various subjects related to clustering.

According to the web of knowledge [77], more than 6000 published papers titled by cluster analysis in 140 subject areas. Figure 1.1 on page (2) displays the rapid growth of cluster analysis research from the 1950s to our own day. We can infer that most of the cluster analysis literature has been written in the past three decades, although cluster methods have been recognized only in this century. The main reasons for the rapidly increasing number of publications on clustering are two: first, the actual needs of problems which have large

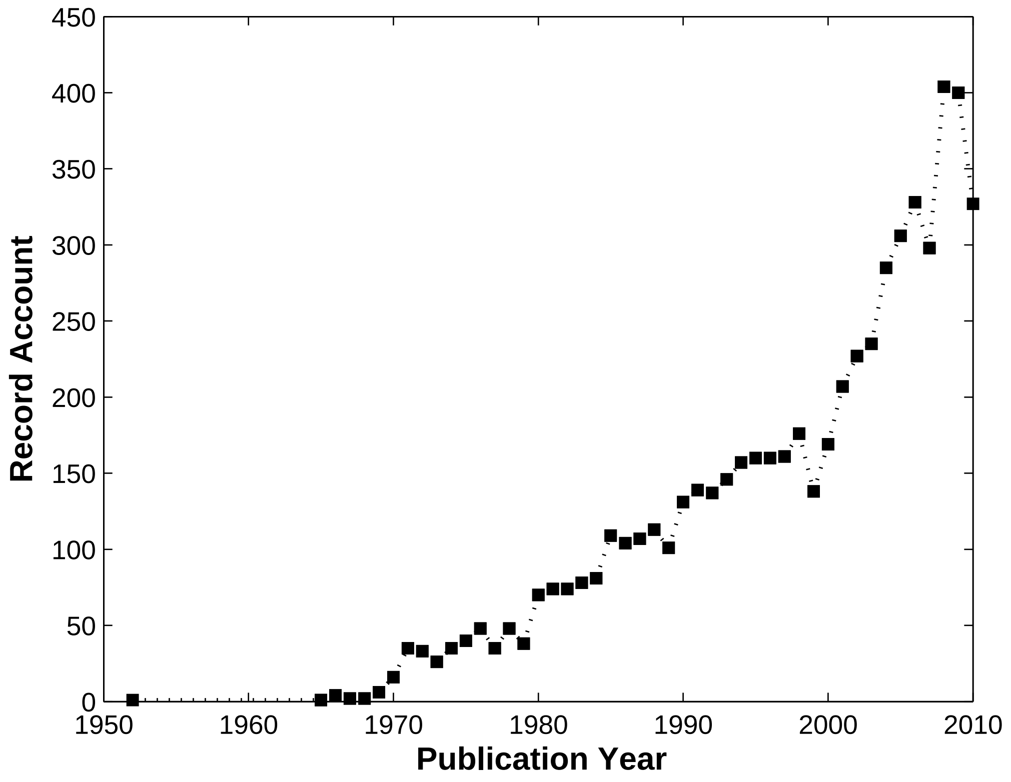


Figure 1.1: The growth of publications on clustering

data sets, needing to be calculated by very high-speed computers, which did not exist until this century. This in fact tempts researchers to apply their empirical programs commonly to real data, which expands the databases to get varied results. Second, the wide range of clustering applications and needs requires us to apply these methods to problems in various areas. Consequently, the clustering subject itself, as a result of these two reasons, needs to be improved. So, new methods have been devised.

Many researchers apply clustering algorithms, by means of various techniques. The reason for such different clustering methods is that they have a variety of uses. These objectives can be summarized [8, 15, 81, 2] as: finding a true typology, model fitting, hypothesis generating through data exploration, hypothesis testing and data reduction. All these purposes have given rise to a wide selection of applications. To see how data reduction can be applied, for example, MORRISON [61] showed as an assumption that if there is a sample of 100 cities which could be used as test markets, but the available budget was only to test in five cities, then we could reduce this number by clustering the cities into five clusters such that the cities within each group were very similar to the rest of the group. Then one city from each cluster could be selected and used as a test market.

Opinions differ on the definition of clustering. There are, for instance, many arguments over the precise definition of the concept: clustering or cluster analysis. These two terms refer to almost the same conception. But an acceptable definition which can be concluded from previous researches is that: **Clustering** [48, 57, 5] is a scientific method which addresses the following very general problem: given the data on a set of entities, find clusters, or groups of these entities, which are both homogeneous and well-separated. Homogeneity means that the entities in the same cluster should resemble one another. Separation means that entities in different clusters should differ from one another.

There are numerous ways to express homogeneity and/or separation by one or two criteria. In addition, various structures may be imposed upon the clusters, the two most common being the hierarchy and the partition. Choosing a criterion and constraints defines a clustering problem. If this is done explicitly and rigorously, it takes the form of a mathematical program [39]. Many methods exist for solving most clustering problems. In rare cases, there are exact algorithms which provide proven optimal solutions [64, 6].

Because there may be confusion between the concepts, I want to clarify the differences between clustering method and classification. Many sources indicate this, but for more details see [57, 15, 81]. Classification is called supervised learning because all classes are labeled and then the goal is that each entity must be assigned to the desired class. So, the task is to learn to assign entities to predefined classes by using training set from these labeled objects to design a classifier for future observations. This is the opposite of clustering, where no predefined class is required. The task is to learn a classification entirely from the data. These differences can be simplified as supervised learning and unsupervised learning.

One of the most popular clustering methods is K-means. The main principles of K-means clustering (see Figure 1.2 on page (4)) for K clusters can be given as: (1) initialization: by suggesting centres (centroids) from the dataset as representative for each cluster; (2) allocation: by calculating the members of each cluster; (3) location: by calculating the new centroids for each cluster; (4) assigning the objective function. These steps aim to find the minimum objective function, which is know as the sum of all the differences between the centroids and the members of each cluster. Because we need to assign the centroids in each cluster, we have to measure the distances between the centroids and the entities in each cluster. For this, we use a measurement tool called the distance function, which will be defined later.

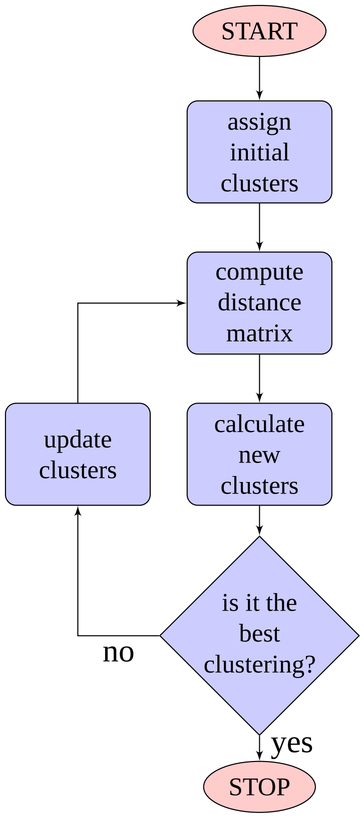


Figure 1.2: The K-means clustering algorithm.

1.2 Clustering Methods

Clustering techniques are mainly classified into partitional and hierarchical. In the partitional, the data points are directly divided into a desired number of partitions (or clusters): in the hierarchical clustering, a sequence of non-predefined number of partitions takes place, which run either from one cluster containing all the entities to *k* clusters each containing a single object, or vice versa. The first option is called agglomerative hierarchical clustering, and the second is known as divisive hierarchical clustering.

Before delving into the details of the former species I should define some terms which will be used later.

A **sample set** is a finite set $X = \{x_1, x_2, \ldots, x_N\}$ of N entities. which has to be divided into clusters.

Features are measured or observed in a variable of type character or numeric values. They are also called attributes, variables, or dimensions. Each entity has one or more features.

An $N \times q$ **data matrix** is obtained by measuring or observing q features of the entities of X.

An $N \times N$ **dissimilarities matrix** $D = (d_{ij})$ for $i, j = 1, 2, \ldots, N$ or distance function is a measurement tool used to compute the differences between entities of X; this matrix must satisfy:

1. Symmetry,

$$d(x_i, x_j) = d(x_j, x_i) \quad ;$$

2. Positivity,

$$d(x_i, x_j) \geq 0 \quad \text{for all } x_i \text{ and } x_j \text{ in } X \quad ;$$

3. Reflexivity,

$$d(x_i, x_j) = 0 \quad \Leftrightarrow x_i = x_j \quad .$$

   In this case, the distance function is called semimetric function. But if the condition:

4. Triangle inequality,

$$d(x_i, x_j) \leq d(x_i, x_k) + d(x_k, x_j) \quad \text{for all } x_i,\, x_j \text{ and } x_k \text{ in } X$$

is satisfied, it is called a metric.

The most popular dissimilarity measures are shown below:

- The Euclidean Distance $d(x_i, x_j) = \sqrt{\sum_{k=1}^{q}(x_{i_k} - x_{j_k})^2}$
- Manhattan Distance $d(x_i, x_j) = \|x_i - x_j\|_1 = \sum_{k=1}^{q} |x_{i_k} - x_{j_k}|$

1.2.1 Hierarchical Clustering

In hierarchical clustering the items (features) in the data matrix are not divided into a particular number of clusters. Thus, there is no predefined number of clusters but series of partitions have been applied. These partitions by either the agglomerative method or the divisive method produce a tree or dendrogram which may be represented by a two-dimensional diagram illustrating the fusions or divisions made at each successive level.

Agglomerative Hierarchical Clustering

Although agglomerative hierarchical clustering methods are considered the oldest, they are still used in many applications. Some claim that they are the most frequently used methods of cluster analysis [21, 34]. If the similarity or distance matrix is known, the agglomerative method starts by separating clusters which are each of size 1. So, if we have a dataset of N entities the technique begins with N clusters. Then the first two closest (most similar) pair of clusters are merged together, which reduce the number clusters to $N - 1$. There are three main ways to calculate the distance between clusters. single linkage, complete linkage and average linkage clustering. There are many other ways that can be applied such as: Equal-Variance Maximum Likelihood (EML) Method [9], and Ward's method [80]. In single linkage clustering, the distance between two clusters is equal to the minimum i.e., the distance between any two members of different two clusters must be minimum. The flowchart in Figure 1.3 on page (7) illustrates the process of the single linkage clustering method. In

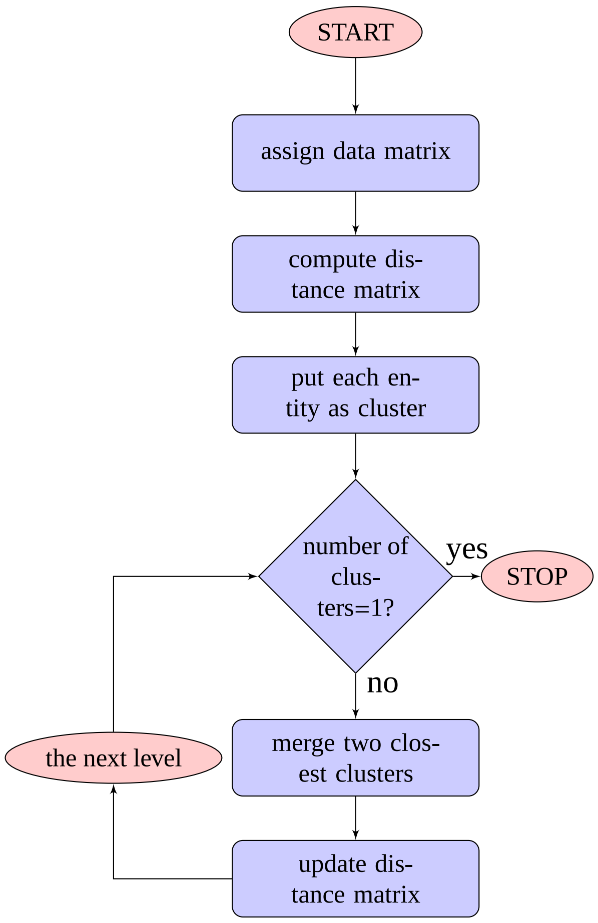


Figure 1.3: Flowchart of agglomerative clustering algorithm.

contrast, complete linkage clustering can occur when the distance between two clusters is equal to the maximum distance from any member of one cluster to any member of different cluster. In average linkage clustering, the distance is equal to the average distance from any member of one cluster to any member of the other cluster. To illustrate these concepts : Let $\delta(C_1, C_2)$ be the distance function between two clusters C_1 and C_2 . It can be computed as:

- $\delta(C_1, C_2) = \min \{ d(i, j) : i \in C_1 \quad , \quad j \in C_2 \}$. For single linkage.
- $\delta(C_1, C_2) = \max \{ d(i, j) : i \in C_1 \quad , \quad j \in C_2 \}$. For complete linkage.
- $\delta(C_1, C_2) = \dfrac{1}{|C_1| \cdot |C_2|} \sum_{i \in C_1} \sum_{j \in C_2} d(i, j)$. For average linkage.

Example 1.2.1

Consider Table 1.1 on page (8) which shows the distances in miles between some United States cities [14]. The method of clustering is single linkage. So, in the first stage BOS and NY are merged into a new cluster because 206 is the minimum distance. After applying the agglomerative algorithm, the rest of the solution can easily be concluded from the dendrogram in Figure 1.4 on page (8).

	1	2	3	4	5	6	7	8	9
	BOS	NY	DC	MIA	CHI	SEA	SF	LA	DEN
BOS	0	206	429	1504	963	2976	3095	2979	1949
NY	206	0	233	1308	802	2815	2934	2786	1771
DC	429	233	0	1075	671	2684	2799	2631	1616
MIA	1504	1308	1075	0	1329	3273	3053	2687	2037
CHI	963	802	671	1329	0	2013	2142	2054	996
SEA	2976	2815	2684	3273	2013	0	808	1131	1307
SF	3095	2934	2799	3053	2142	808	0	379	1235
LA	2979	2786	2631	2687	2054	1131	379	0	1059
DEN	1949	1771	1616	2037	996	1307	1235	1059	0

Table 1.1: *Distances in miles between U.S. cities*

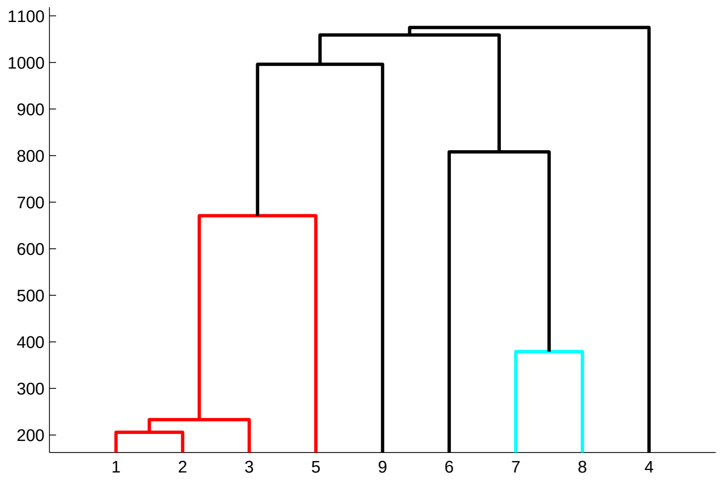


Figure 1.4: Dendrogram of single linkage clustering

Divisive Hierarchical Clustering

In contrast to agglomerative, the divisive hierarchical clustering starts with one cluster. So, the dataset of N entities belongs to a cluster in the first step. Then the procedure successively splits it until each cluster contains one object. For more details see [5, 81].

1.2.2 Partitioning

Cluster analysis deals with various types of criteria, but I am concerned only with the partitioning in Euclidean space $\mathbb{R}^q$. To explain in brief, let $X = \{x_1, \ldots, x_N\}$ be a set of objects or entities to be clustered ($x_i \in \mathbb{R}^q$) , and let C be a subset of X. Then $P_K = \{C_1, C_2, \ldots, C_K\}$ is a partition of X into K clusters if it satisfies: (i) $C_k \neq \emptyset; \quad k = 1, 2, \ldots, K$, (ii) $C_i \cap C_j = \emptyset; \quad i, j = 1, 2, \ldots, K; \quad i \neq j$, and (iii) $\bigcup_{k=1}^{K} C_k = X$. General principles for the partitioning criteria are presented in Figure 1.5 on page (10).

K-Means Algorithm

One of the most popular criteria for partitioning points in Euclidean space is called the minimum sum-of-squares clustering (MSSC), since it considers at the same time the homogeneous and the separation criteria. Minimizing the sum-of-squares errors criterion amounts to replacing each cluster by its centroid and minimizing the sum-of-squares from the entities to the centroid of their cluster. A mathematical formulation of the MSSC problem and its steps are given in Algorithm 3.1 on page 39 in Chapter (3).

Fuzzy Clustering Algorithm

While K-Means present hard clusters, the fuzzy clustering gives soft clusters. In the fuzzy clustering (also called fuzzy C-Means in some articles [46]), each entity has a degree of belonging to clusters depending of how far from the centroids. So, a particular entity may belong to more than one cluster.

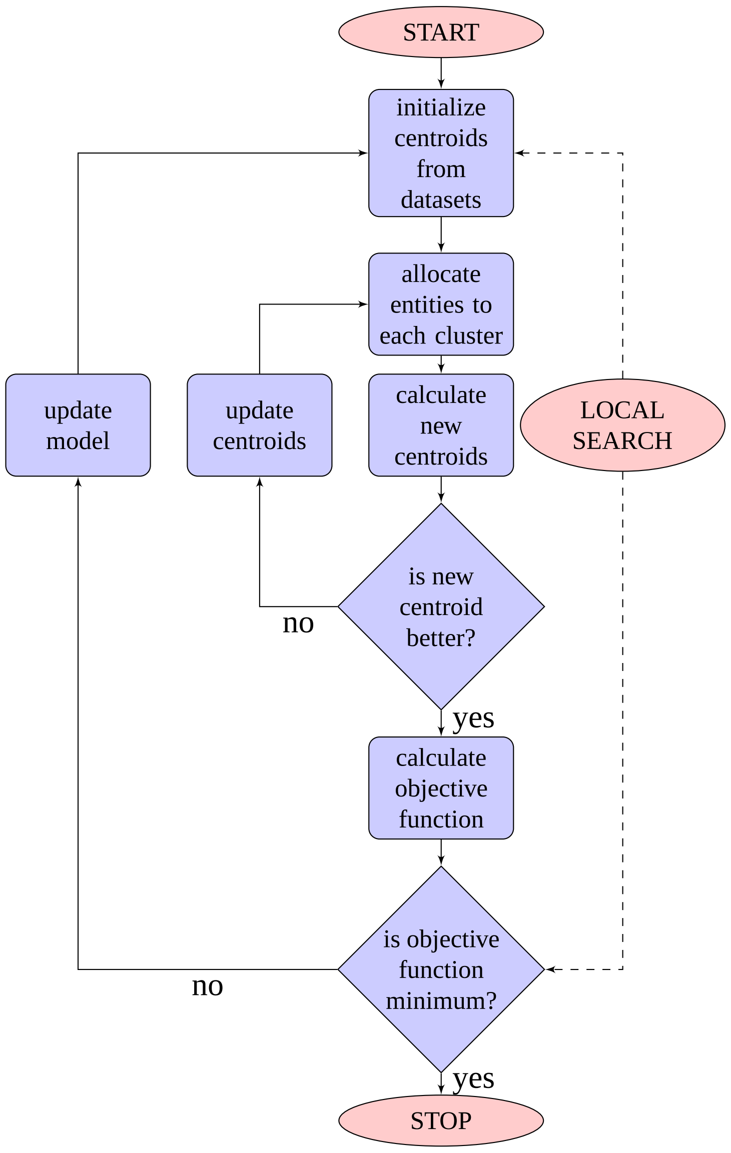

START
initialize centroids from datasets
allocate entities to each cluster
update model
update centroids
calculate new centroids
LOCAL SEARCH
is new centroid better?
no
yes
calculate objective function
is objective function minimum?
no
yes
STOP

Figure 1.5: The partitioning algorithm

Graph Theoretic Methods

In any weighted graph, the node represents the entity point in the dimension space or feature space. However the edge between any two pairs of nodes corresponds to their proximity. The constructed graph should be capable to detect the non-homogeneous edges. Therefore, good clustering can be assigned by those inconsistent edges [47].

1.3 Outline

This research is designed to improve the K-Harmonic Means (KHM) clustering by applying the basic Variable Neighbourhood Search. KHM, first proposed in [83, 82], is less sensitive to initialization than K-Means (KM). Some algorithms from the literature are compared with KHM after applying VNS. Although KHM surpasses KM in many faces as it is explained in the next chapters (See for example Table 4.1), it is shown that KHM may degenerate in some parts of its solution. In certain experiments, it could stop through this degeneracy. The algorithm for removing degeneracy has been applied for many familiar datasets and compared with those results obtained by degeneracy. The remaining chapters of this thesis are organised as follows.

In Chapter 2, a brief overview of metaheuristics is provided. The main concepts of heuristics are shown by some illustrations. Most metaheuristics have been structured to provide high level frameworks for building heuristics for further classes of problem, since certain problems cannot be solved by heuristics. The main and most used metaheuristics in this research are then covered, including: Tabu-Search (TS), Simulated-Annealing (SA), Genetic-Algorithm (GA), and Variable Neighbourhood Search (VNS).

In Chapter 3, an illustration of a definition of KHM is presented beside the KM algorithm. The main parts of the KHM algorithm, including: membership function, weight function, centroids and objective function are covered in a code. The variable neighbourhood search heuristic is suggested as a method for improving KHM. This heuristic has been tested on numerous datasets from the literature. To assess the strength of the code, some comparisons with recent ones from Tabu Search and Simulated Annealing heuristics have been made.

In Chapter 4, counter examples show the degeneracy in a KHM local search. An algorithm is applied to avoid the degeneracy in KHM and used within recent variable neighbourhood search (VNS) based heuristic. Computational results are presented to show the improvement obtained with the degeneracy correcting method, which is performed on the normal test instances from the literature.

Chapter 2

Metaheuristics

Combinatorial optimization problems have attracted much interest, due to the advancements made in operational research. Since most of these problems are NP hard, heuristics and other approximate solution approaches with performance guarantees are required. This chapter includes a detailed discussion on metaheuristics and classical heuristics. Many branches of the metaheuristic family are mentioned in this chapter. The most commonly used methods are Simulated Annealing (SA), Tabu Search (TS), Genetic Algorithm (GA), Particle Swarm Optimization (PSO) and Variable Neighbourhood Search (VNS) which are discussed in more detail.

2.1 Introduction

Despite the rapid growth and developments of computation, in speed and size in particular, the exact solution of many decision and optimization problems is obtained in an unreasonable amount of time. This is due to the complexity of these problems, in particular, those involving large sizes. In certain problems, the exact algorithms take too long (maybe days or more) to get an optimal solution. As a result, many researchers prefer to use heuristic algorithms in practical applications. Because it is impossible to continue such searches to the end, these approaches maybe trapped in a local optimum. The main shortcoming of heuristic algorithms can be amended by applying metaheuristics.

Optimization problems can be classified into many categories. The classification may be based on the types of variable. They may include integer, discrete, zero-one, or real variables. However there are only two major categories: continuous variables if the solution space is real numbers; and discrete variables or combinatorial, if the set of the solution space is finite, or infinite but enumerable.

Discrete optimization, which is also known as combinatorial optimization, is much more common and is the kind used in the present research. The combinatorial optimization problem can be defined as that of finding the best solution among a finite number of possible solutions. Many real-world problems may be modelled as combinatorial optimization problems. These problems can appear in various assignments such as: scheduling problems, location problems, set partitioning/covering, vehicle routing, travelling salesman problems and many other more. Formally, the combinatorial optimization problem P can be defined as [66]:

Definition 2.1.1 (optimization problem) *An optimization problem P is given by a set of instances I. An instance $i \in I$ of an optimization problem is a pair (S, f), where S is the solution space; f denotes the objective function that maps $f : S \rightarrow \mathbb{R}^+$ The problem is to find $s^* \in S$ such that $f(s^*) \leq f(s), \forall s \in S$. Such a point s^* is called a globally optimal solution of (S, f), s is called a feasible solution.*

Most of these problems can be considered as **NP**-hard, that is they cannot be solved in a polynomial time. That means it is not possible to guarantee that an optimal solution to the problem can be found within an acceptable timeframe. For more details on the concepts of **P** and **NP** complexity, see [26, 43].

This chapter outlines the main metaheuristics approaches and gives an illustration of traditional heuristics.

2.2 Classical heuristics

All combinatorial optimization solution methods can be classified as either exact or approximate. The first kind is the algorithm which gives an exact solution for a predefined problem. There are many exact methods but the ones most commonly used are dynamic programming and branch-and-bound. However, an approximate algorithm does not necessarily give an optimal solution to an input problem. The approximate solution can be classified mainly in two ways: approximation algorithms or heuristics. The approximation algorithm always provides a feasible solution (if it exists) of a certain quality [18, 78]. However, there are plenty of **NP**-hard optimization problems which cannot be approximated. Therefore, one must apply heuristic methods which do not guarantee either the solution quality, or the time limitations. The definition of a heuristic is proposed in [70] as "a method which seeks good solutions at a reasonable computational cost without being able to guarantee optimality, and possibly not feasibility. Unfortunately, it may not even be possible to state how close to optimality a particular heuristic solution is".

Because of these shortcomings, some heuristics may perform badly due to the initialisation of a given problem. But this does not nullify the benefits of heuristics, since they perform well in plenty of problems. The most common heuristic methods based on generating a problem for a solution can be classified as follows [53]:

- constructive methods
- local search methods
- inductive methods
- problem decomposition/partitioning
- methods which reduce the solution space
- evolutionary methods
- mathematical programming based methods

Constructive methods. Constructive heuristics are designed to construct one single feasible solution. It is constructed step by step by using structure information from the given problem. The most commonly used approaches are the greedy [33] and look-ahead [11] approaches.

Local search methods. Local search methods use an iterative process to gradually improve a given feasible solution $s \in S$ until a local optimum is reached. The neighbourhood for each solution is considered a set of all the feasible solutions in the vicinity of s. At each iteration, a neighbourhood of the current candidate solution is explored and the current solution is replaced with a better solution from its neighbourhood, if one exists. If there are no better solutions in the observed neighbourhood, a local optimum is reached and the solution process terminates.

Inductive methods. The main principle of inductive methods is to generalise a simple problem solution to be used for harder problems of the same type.

Partitioning. The problem is decomposed or partitioned into a number of smaller/simpler subproblems, each of them being solved separately. The solution processes for the subproblems can be either independent or intertwined, with a view to exchanging the information about the solutions of different subproblems.

Methods which reduce the solution space. Some parts of the feasible solution region are ignored from further consideration in such a way that the quality of the final solution is not significantly affected. The most common ways of reducing the feasible region include the tightening of the existing constraints or introducing new constraints, such as fixing some variables at reasonable values.

Evolutionary methods. As opposed to single-solution heuristics (sometimes also called trajectory heuristics), which consider only one solution at a time, evolutionary heuristics operate on a population of solutions. At each iteration, different solutions from the current population are combined, either implicitly or explicitly, to create new solutions which will form the next population. The general goal is to make each created population better than the previous one, according to some predefined criterion.

Mathematical programming based methods. In this approach, a solution of a problem is generated by manipulating the mathematical programming (MP) formulation of the problem. Generally speaking, mathematical programming may be used in two different ways: (i) aggregation of variables; (ii) relaxation of variables. Popular relaxation technique is so-called Lagrangian relaxation.

2.3 Metaheuristics

Heuristic methods were first initiated in the late 1940s [69]. These heuristics relay on the structure of a certain problem and cannot be applied to others. In the 1980s [27], metaheuristics were structured to provide high level frameworks for building heuristics for further classes of problem. Many advances have been made in the last few years in both the theory and application of metaheuristics. They are used to find approximate solutions for hard optimization problems. According to [79], "A metaheuristic is an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality solutions. It may manipulate a complete (or incomplete) single-solution or a collection of solutions at each iteration. The subordinate heuristics may be high (or low) level procedures, or a simple local search, or just a construction method ". To understand this definition of metaheuristics, some of the main concepts of metaheuristics are discussed below [53].

Diversification vs. intensification. The first term means the exploration of the search space. In this, the algorithm shifts to different parts (depending on the distance function used) of the search space looking for the best local optimal. The second means the exploitation of the current solution. In this, the algorithm focuses on the current search area, by exploiting all the available information from the search experience. It is essential in search process to keep an adequate balance between the diversification and intensification.

Randomisation. As an application of diversification process, randomisation allows the algorithm to select one or more candidates by a random mechanism from a solution space.

Memory usage. Some metaheuristics save certain information during the search process in storage, to be used in further steps of the search. Such information could be the feasible solutions, number of iterations, or solution properties. Although the Tabu Search method is a very significant example, since memory is used mainly in the search process, as explained later, some other metaheuristics, for instance, the Genetic Algorithm [72] and Ant Colony Optimization [23, 22], use it less, since it is incorporated implicitly.

According to these principles, most metaheuristics try by different means to avoid the locality (see Figure 2.1 on page 19) in the solution process. Before describing the main metaheuristics, a neighbourhood structure and a local optimal solution should be defined as [53]:

Definition 2.3.1 (neighbourhood structure) *Let P be a given optimization problem. A neighbourhood structure for problem P is a function $\mathcal{N} : S \to \mathcal{P}(S)$, which maps each solution $x \in S$ from the solution space S of P into a neighbourhood $\mathcal{N}(x) \subseteq S$ of x. A neighbour (or a neighbouring solution) of a solution $x \in S$ is any solution $y \in \mathcal{N}(x)$ from the neighbourhood of x.*

Definition 2.3.2 (local optimal solution) *Let $\mathcal{N}$ be a neighbourhood structure for a given optimization problem P as defined by 2.3.1. A feasible solution $x \in S$ of P is a locally optimal solution (or a local optimum) with respect to $\mathcal{N}$, if $f(x) \leq f(y), \forall y \in N(x) \cap S$ (see Figure 2.1).*

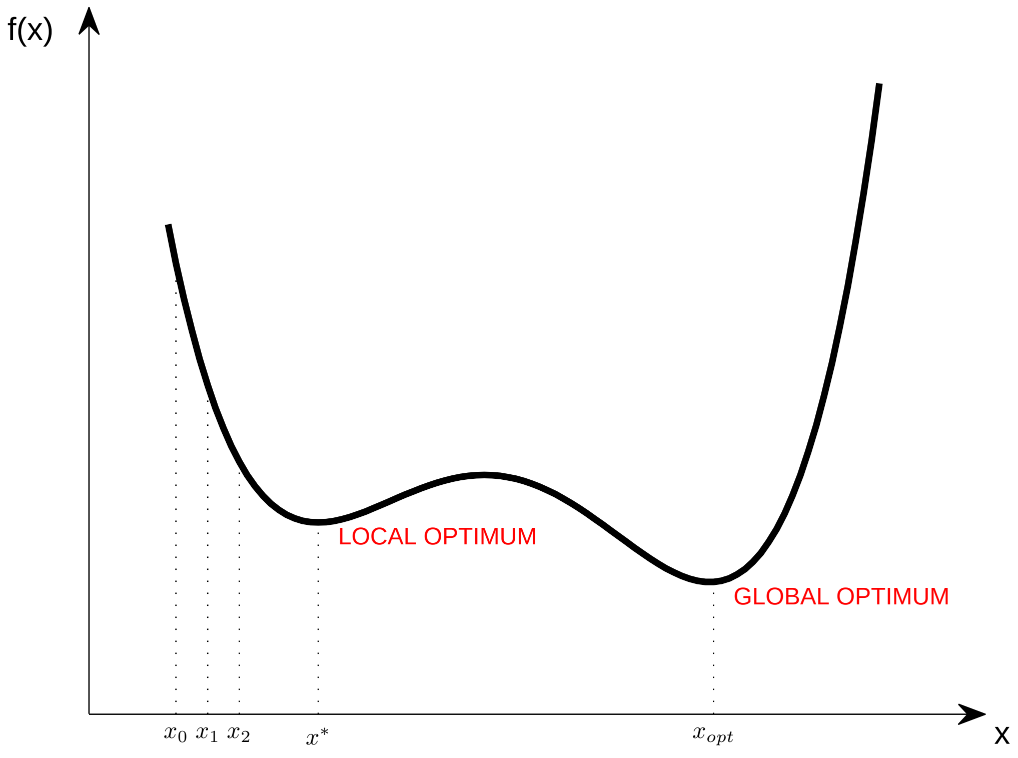


Figure 2.1: The local minimum trap in the local search

2.3.1 Simulated Annealing

The Simulated Annealing (SA) is a kind of a metaheuristic which uses the principles of a probabilistic approach by Monte Carlo [38] and the basic local search. It is considered to be one of the oldest techniques in metaheuristics. The process of annealing is also used in metallurgy which inspired its use in metaheuristics. Kirkpatrick [51] and Cerny [19] invented this independently. Every iteration of Simulated Annealing is the neighbour of a current solution, which is randomly generated. Next, it is moved to the solution of the neighbour which is based on the value of the objective function and criteria of Metropolis Algorithms [56]. The current neighbour is considered to be true if the neighbour which we have selected has an objective function of greater value than the presently selected solution. If this case does not hold true, then the Metropolis criterion is used to determine a new solution.

The Simulated Annealing was also used in the annealing of solid materials. In this process material is subjected to increasing temperatures to the point where it actually melts. The previous solid state is retained by reducing the temperature. In order to achieve a successful annealing, the gradual lowering of the temperature is very important. An inappropriate shape is obtained if the cooling is done too fast. Conversely, if the cooling is done properly a more symmetric solid shape is obtained, with an energy sate which is very low. With respect to the combinational optimization value of objective function being equivalent to its energy, the solution of the problem that is generated is equivalent to the state of the material, and a move to any solution of the neighbour is equivalent to a change in the energy state.

The first solution is obtained randomly or by using some constructive heuristic. The end condition of the algorithm is represented by a certain variable. Generally, the stopping condition or end condition is based on the maximum time allowed to keep running, the total number of iterations allowed or the total number of iterations allowed without making improvements. There is a variable which is used to find the probability (p) of success, which can be found out by comparing the similarities of physically annealed solids. The values of parameters for temperature in a simulated annealing algorithm can be defined by positive numbers (t_n) such that $t_0 \geq t_1 \geq \ldots$ and $\lim_{n \to \infty} t_n = 0$. The cooling schedule is the name given to this sequence of positive numbers (t_n). Acceptance is obtained for the huge temperature values that are used in the initial stage. However, small values used at the end give us very detailed results which reject almost every solution that is non-improving. Geometric sequence is the most commonly used cooling schedule. There is a great decrease in temperature values because of the cooling schedule. If the temperature is changed, say, after M iterations a stronger algorithm is generated. Here $M \in \mathbb{N}$ is considered to be a predefined variable.

The process described above is memory-less, since a trajectory is being followed in the state of the space which chooses the successor state. This is dependent of the incumbent, without keeping tracing of the history of search process. The SA pseudo-code is illustrated in Algorithm (2.1 on page 21).

Algorithm 2.1: Simulated Annealing

```
Function SA ($S$, $f(x)$, $t_n$, $Maxit$);
1  Choose initial solution $s$ from the solution space $S$ ;
2  Select a neighbourhood structure $\mathcal{N} : S \rightarrow \mathcal{P}(S)$ ;
3  Set $i = 0$ ;
4  while $i \leq Maxit$ do
5      Choose $s^{'} \in \mathcal{N}(s)$ at random;
6      if $f(s^{'}) \leq f(s)$ then
7          $s = s^{'}$ ;
8      else
           Choose $p \in [0, 1]$ at random ;
9          if $p < exp(\frac{f(s)-f(s^{'})}{t_n})$ then
10             $s = s^{'}$ ;
11     $i = i + 1$ ;
12 return $s$ ;
```

2.3.2 Tabu Search

Tabu search (TS) is a metaheuristic method of learning, which is based on the concepts of discovery and problem-solving with the use of reasoning and past experience. It is a computer program which uses methods based on its previous memory or, say, experience in order to solve a given problem, instead of using a mathematical procedure. This method was basically proposed in 1986 by Glover (see [28]). Unlike the Simulated Annealing process, it is not stochastic in nature but like the Simulated Annealing process, it avoids traps which bring the search to a dead end. This is the basic form of the Tabu Search method. In this, a Tabu list (TL) is formed which has short memory span. This is a list of forbidden solutions, which saves and stores all the solutions that have been previously used to prevent them from being repeated. This method of eluding local optima is more of a deterministic approach. The most important point to be noted in Tabu Search is its flexible and automatically adjustable system which stores all the search history. The present form of the Tabu Search method has a much broader memory span and storage system than its predecessors. This program makes the search for a solution easier. The program explores the aspects of the most feasible solution of a problem, also making sure that it does not coincide with the previous solutions stored in its memory i.e. the Tabu list. This list also has an automatic update system which works on the principle of adding the current solution to the Tabu list and deleting the oldest one from its memory. It accepts even the worst solution, because it does not have an objective method of analyzing them. This helps to escape from local optima. The most appropriate and suitable solution is stored separately during this process. The complexity of the solution list and its diversification is controlled by the crucial boundary of the list, which deletes old solutions within the length of the Tabu list. This parameter is the length of the Tabu List which checks the increasing number of progressing solutions and makes sure that unsuitable solutions are removed and only the most appropriate ones are saved and worked upon. The length of the Tabu list is also known as the tabu tenure. The length of Tabu list is permanent or can be changed dynamically and automatically at every step. A short Tabu list focuses on less complex solutions, according to the space provided by the small data structure without any big moves to increase the broad array of solutions, whereas a lengthy Tabu list provides diverse solutions and focuses more on exploring wider aspects of correct solutions. It allows for more complex and diverse solutions; thus, the length is kept constant under a process of upgrading. The length of this list can also be altered, which gives the method more strength. However, the Tabu list also takes up a great deal of time in searching for the right solution from the list and this can make this system ineffectual. This weakness of the method can be remedied by storing only the particular parts of a solution that are important, instead of the whole solution. The attributes to be looked for by the Tabu search program are fed into it. These attributes look out for matching solutions to store. This helps by making the system less inefficient and thus more useful. It filters the important attributes of the solutions into the Tabu List. This can however, cause very important information to be missed, because some important parts of the Tabu List can be lost due to having few attributes. Very fine solutions can sometimes be missed in the search. This problem can be solved by setting up Aspiration Criteria (AC) which store all solutions that meet the criteria. It allows any better solution to take over from the best solution so far.

Tabu Search has the ability to steer solutions away from dead end traps, which is modelled on the memory programs of humans. The methodology starts with some basic solution which is formulated randomly. At every step, it then improves the solution from a given number of solutions which are called the ’allowed set’ and that are not present in the Tabu List. The ’allowed set’ is a list of admissible solutions. This method can then be altered to a first improving or best improving procedure. In the first improving procedure, it searches for a solution and gives the first one that is stronger than the original one. In the best improving procedure, the program searches all the ’allowed set’ extensively to find the best solution. When an improved solution is found, the Tabu List replaces the previous basic solution with the better one by the FIFO method. The FIFO (first-in-first-out) method, as previously stated, adds the newest possible solution to the Tabu List and deletes the oldest solution. Thus the Tabu Search method can be termed explorative, having a broad range of programming with low memory. This procedure is repeated and again the most suitable solution replaces the last one, and so on.

Other extended versions of this Tabu Search program have been developed since its origin in 1986. It has been enhanced by a long-term memory [31]. This long-term memory memorizes every recent solution and its relevant up grading in a process called Recency. It also provides information on the number of visits made to each solution, called Frequency. The quality of the solution and its parameters are also recorded within the memory; this is called Quality. The memory also shows the influence during the search and puts forward the inclinations which showed during the search for the solution termed Influence. These are the four dimensions of this metaheuristic [13]. The long-term memory can be used within Tabu research through the use of frequency measures, such as the ’residence’ and ’transition’ processes. The residence process is about the number of observations of an attribute, while the Transition process reveals how many times the value of the attribute was changed during the search. This provides more objectivity to solutions. It diversifies and intensifies our search by selecting solutions which match the attributes we set and by putting forward the solutions with the best attributes which are known as the *elite subsets*. The quality of a Tabu Search tends to be more objective in solutions. A great number of such solutions causes a greater search into the most relevant attributes and solutions present in the Tabu List. Influence, however, refers to the amount of change that comes in every progressive solution. The Aspiration criterion plays a very important role in this regard. It also helps to develop the most suitable candidate list for a job. It tells us the decisions which we have for finding the right solution and helps in making moves according to these critical indications which we have made.

Many parts of this search are in use along with other metaheuristic procedures for more efficient use and the discovery of more efficient solutions to problems. A more recent development in the Tabu Search method has been made by using it along with other metaheuristic programs to form a hybrid, such as Genetic Algorithms [29] and Ant Colony Optimization [7], among many others. Another modified use of the Tabu Search algorithms is to combine it with the path re-linking method. The path re-linking method provides newer solutions by analysing between the elite subsets. By the combination that they form, the solutions are formulated by choosing them randomly from a proper data structure instead of deterministically, as is the norm for the original Tabu Search method. This makes the search of solutions much faster and also increases the diversity of the solutions. Certain improved algorithms of the Tabu Search are called Reactive techniques, which allow the automatic changing and adjusting of attributes and boundaries during the Tabu Search method. The most important parameter is the *tabu tenure*, i.e. the length of the Tabu List. Glover and Kochenberger [30] say that recency based Tabu Search with basic structure if used for a restricted topic is a strategy which can give very accurate and best solutions/results. The basic algorithm for TS is illustrated in Algorithm 2.2 on page (25).

2.3.3 Genetic Algorithm

Genetic algorithms were derived from the research by Holland on cellular automata in 1975 [44]. They were further used in combinatorial optimization, linear and non-linear, which rendered them the most evolved algorithms [32, 45]. The concept of genetic algorithms is a biological similarity, according to which the selection of the most competent individuals can be used for the evolution of genetically stronger species. This raises the related question whether this procedure can be used for correcting optimization difficulties. In the above mentioned process of selective breeding, the offspring of the species retain the optimum characteristics of their species, which are determined by the genes of the selected parents. Genetic

Algorithm 2.2: Tabu Search

```
Function TS ($S$, $f(x)$, $Maxit$);
1  Choose initial solution $s$ from the solution space $S$ ;
2  Let $s^* = s$ be the best solution so far ;
3  Select a neighbourhood structure $\mathcal{N} : S \rightarrow \mathcal{P}(S)$ ;
4  Initialise Tabu List $TL$ ;
5  Initialise Aspiration Criteria $AC$ ;
6  Set $i = 0$ ;
7  while $i \leq Maxit$ do
8      Choose the best solution within the allowed set:
       $s^{'} \in \mathcal{N}(s) \cap \{s \in S | s \notin TL\}$ ;
9      $s = s^{'}$ ;
10     if $f(s) \leq f(s^*)$ then
11         $s^* = s$ ;
12     Update TL and AC ;
13     $i = i + 1$ ;
14 return $s^*$ ;
```

Algorithms make use of chromosomes to find the combination of genes in offspring. Genetic algorithms also focus on problems within generations and chromosomes are used in finding answers to these problems. A single component of a chromosome is called a gene and these genes can have various combinations or values, known as alleles. These combinations are also named ’genotype’ and ’phenotype’ maps of a generation or species or individual, which constitutes a fine Genetic Algorithm. In evolution, the probability that certain chromosomes will be passed down to offspring depends on its fitness i.e., not only with respect to its subjective components, such as its nature, but also on objective components, such as functions. Then these selective chromosomes are bred into the genes of the offspring, who get all the dominant genes and characteristics from their family line. This selective breeding promulgates the ’survival of the fittest’ concept. The chromosomes are assigned values of 0 and 1 at different loci on them. The locus is the point on a chromosome where the binary value is present. It affects the fitness of a chromosome. A fit chromosome is readily passed down to the next generation to replace the weaker offspring. The term fitness has great importance in the concept of the Genetic Algorithm which gets its name from the genetic nature of the process.

The mechanisms of crossover and mutation take place when there are two or more than two parents. Crossover involves putting certain genes of a parent in place of the other's, resulting in offspring. Mutation involves only one set of genes in which the binary values are changed and the procedure is repeated until the process starts giving weaker results than before. The Genetic Algorithm entails stronger genes in every succeeding generation. The steps involve selecting the size and composition of the population. The size should display the characteristics of efficiency and durability over time and the efficacy of the solutions being used. The size can be changed during the process or can be kept unchanged, according to the needs of the process. The composition of the population is mostly kept random but nowadays certain heuristic procedures are in use for selecting only those which meet the required criteria of solutions. In the next step, the processes of mutation and crossover are selectively applied on those parents who are the fittest, in terms of genes. The roulette-wheel method is used in these processes, which implies that only the fittest of parents should be used for the process of reproduction.

Other methods are also used for selecting individuals. The stochastic universal selection method lessens the increased number of variables which became involved in the roulette-wheel method. The procedure of tournament selection includes choosing a set of parents and selecting those which are most appropriate for the process. Unstructured and structured populations also come into play in Genetic Algorithms. The former involves a combination of any two individuals and the latter involves the recombination of any individual with one selected from a set with higher fitness value.

After the process of selection, genetic operators come into play, i.e., mutation and crossover, as stated before. It is not always necessary to use both these procedures on the selected population. These procedures can be used one at a time, or both together, or a different procedure can also be operated on the population which selects for the purposes of the particular study that we are conducting; although crossover is more often applied than mutation to the selected population. The reason for this choise is that mutation weakens the already present solutions and also reduces the strength of any new solutions that can be found.

As regards the crossover method, it involves putting the genes of one parent in place of the other to produce offspring, as noted above. Crossover can take several forms. Two-parent crossover [65] involves few persons as the source for the genes. Multiple-parent crossover [24] involves the offspring produced by recombining the genes of more than two parents. New developments in this area are constantly being made and modified forms of the crossover method have been introduced. Gene Pool Recombination [63] makes use of the whole current population to formulate the next line of population. Bit-Stimulated Crossover [52] formulates the next line of a population from an already existing probability within that population.

Sometimes an early inclination in a new generation towards the required result is seen and this can cause problems. This situation should if possible be avoided, by the right functioning of the genetic operator of the mutation. The process of mutation, as noted above, involves changing the values of a gene and the respective chromosomes by the effect of certain factors such as noise. A general selected population is passed through a certain factor which changes the allele value and further generations are reproduced which have the new allele value in their gene pool. Immigration theory can also be used for this purpose, this includes those individuals who were not previously present in the selected population and who might belong to certain areas not previously included in the study. This asks for an updated and fresh review of the research and previous research of the same kind in the Genetic Algorithm.

Sometimes undesirable results are also produced, resulting from the genetic operators used. These undesirable results can be manipulated in any of three ways. Such individuals, who are part of the undesirable outcome of a Genetic Algorithm, can be ”turned down”, ”punished”, i.e., given a weak fitness referral so that they are rejected for any further study in this regard and/or they can be ”fixed”, but this may be impossible. The new population now reproduced is called the current population. The procedure is deferred until certain individuals have to be eliminated to meet the criteria of the Genetic Algorithm. The final result or output is the individual who remains at the end, due to the mechanism of ’survival of the fittest’ in the Genetic Algorithm. Nowadays certain Genetic Algorithms are used which strengthen those individuals. These are formulated by combining other metaheuristics with local methods of carrying out these procedures. These combinations, also known as hybrids, are essential for correcting many of the difficulties faced during the Genetic Algorithm process with regard to probability. A large population produces more diversity in results, while a simple, local procedure with this population strengthens these very results. Memetic Algorithms were introduced for this purpose, in which the process of the Genetic Algorithm is combined with the particular solution of the difficulty in question. This method was developed by Moscato in 1989 [62]. These algorithms devise better populations through genetic operations of the already existing populations. This method is used when other metaheuristic methods must be involved in the study. It is basically a form of Genetic Algorithm including a Tabu Search.

2.3.4 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a nature-inspired algorithm which was first established by [49]. PSO is based on population of solutions as in GA. It is inspired from the individuals (called particles) behavior inside the swarms such as birds or school of fishes. Solutions of the optimization problem can be modelled as the individuals of the swarm which move in the solution space. Improvements of the swarm are obtained from each particle's movement that compile the swarm, based on the effect of inertia and the attraction of the members who lead the swarm. Thus, PSO also belongs to the evolutionary algorithms class.

2.3.5 Variable Neighbourhood Search

A new concept in metaheuristics is the Variable Neighbourhood research, which is broadly applied to data. It analyzes every aspect of a variable before concluding the result and then it moves on to better neighbourhoods if found around some data or variable under study. It gives a more intensified result by going through several neighbourhoods of the data whereas other metaheuristics which pre-date this method go through one at a time. At every stage a number of different neighbourhoods are explored, which adds to the information about the results. In 1997, the Variable neighbourhood search method was developed by Mladenović and Hansen [60]. This led to the finding of the factual information upon which this method is based. The factual information stated that the local optimum of a single neighbourhood of data may not be the local optimum of another neighbourhood. Therefore, the global optimum will be the local optimum of all neighbourhood structures. Last, it states that the local optima of several neighbourhood structures are very closely matched. Through extensive empirical study, it has been found that local optima always consist of some information similar to the global optimum, which means that certain variables are identical in both optima, i.e., general and optimal. There are many well proposed and established VNS schemes. Variable neighbourhood search involves a number of neighbourhoods at a time at every level as noted above. Sometimes a VNS scheme is undertaken in the frame of a broader VNS scheme and the neighbourhood structures involved in each scheme can be very different. The inclination, time management and quality proposed by the user play a big role in selecting the right neighbourhood classification for a scheme. If the VNS scheme is highly developed and evolving, it can encompass a change of neighbourhood structure at all iterations. All the factual information provided previously can be used together to solve a specific problem with changing neighbourhoods at each solution level (see Algorithm 2.3 below). This minimizes extreme diversity and intensification in solutions which are followed through to the end. The factual information gives 3 combinations of methods, namely, deterministic, stochastic and a combination of these two. For more details, the neighbourhood is defined as bellow.

Let S be the solution space and let a set $\mathcal{N}_k$ denote the finite set of pre-selected neighbourhood structures ($k = 1, ..., k_{max}$), and $\mathcal{N}_k(x)$ the set of solutions in the k^{th} neighbourhood of x. Neighbourhood structures $\mathcal{N}_k$ may be induced from one or more metric (or quasi-metric) functions in a form $\delta : S^2 \rightarrow \mathbb{R}$. Then

$$\mathcal{N}_k(x) = \{y \in S \quad | \quad \delta(x, y) \leq k\} \tag{2.1}$$

As a result of that, neighbourhoods of x are nested, i.e. $\mathcal{N}_k(x) \subset \mathcal{N}_{k+1}(x)$ for all x in the solution space S. For more details about calculating the neighbourhood structures see Section (3.3).

Algorithm 2.3: Neighbourhood change or Move or not function

```
Function NeighbourhoodChange $(x, x', k)$;
1 if $f(x') < f(x)$ then
2     $x \leftarrow x'$; $k \leftarrow 1$ // Make a move ;
  else
3     $k \leftarrow k + 1$ // Next neighbourhood ;
```

Function `NeighbourhoodChange()` compares the new value $f(x')$ with the incumbent value $f(x)$ obtained in the neighbourhood k (line 1). If an improvement is obtained, k is returned to its initial value and the new incumbent updated (line 2). Otherwise, the next neighbourhood is considered (line 3).

Variable Neighbourhood Descent. When every neighbourhood classification has been completely analyzed and studied, a Variable Neighbourhood Descent (VND) is formed. It has certain attributes which limit its research value because only a certain number of neighbourhoods can be studied under VND at any time. The solution obtained at the end is the best of all in the data available. The absence of a limiting criterion brings all the neighbourhoods at every level of the method, which gives us a number of solutions, out of which only the best and most valued solution is selected. This procedure is carried out repeatedly until the number of improvements start coming up as zero. This extensive search is very time consuming and extremely exhaustive; therefore, more parameters are introduced for limiting it, including the time limit and maximum but specified number of iterations. The ultimate result is a small, specified amount of information which is local for all the neighbourhood structures that have been studied. This makes the global optimum an attainable goal in the context of VND, unlike other methods which make use of only one neighbourhood. Although the time required for this makes the process of giving diversified solutions very slow, it also continue to increase the strength of the ongoing process. Thus Variable Neighbourhood Descent is applied on more local operations along with any other metaheuristic program. If a metaheuristic program is a modified form of the Variable Neighbourhood Search, then this gives us the General VND scheme (see Algorithm 2.4 below).

Algorithm 2.4: Steps of the basic VND

```
Function VND $(x, k'_{max})$;
1  repeat
2      $k \leftarrow 1$;
3      repeat
4          $x' \leftarrow arg\min_{y \in \mathcal{N}'_k(x)} f(x)$ // Find the best neighbour in $\mathcal{N}_k(x)$ ;
5          NeighbourhoodChange $(x, x', k)$ // Change neighbourhood ;
       until $k = k'_{max}$ ;
   until no improvement is obtained ;
```

Reduced Variable Neighbourhood Search. Reduced Variable Neighbourhood Search (RVNS) is another modified form of VNS, in which a random number and type of solutions are selected from specified neighbourhoods and no steps are taken to refine or improve those raw solutions by any local means. The boundaries formulated in RVNS for its procedure prove to be an optimization difficulty. These limitations include the startup process solution, the limited neighbourhood structure size and the unchecked neighbourhood structure size. Any solution which is found to be the most appropriate is termed the final result, without further intensive search in the limited knowledge and result. The boundaries of this variant which limit its optimum use are checked by certain stopping criteria of a maximum time allowed and the number of iterations allowed between any two progressions of the result. Unlike VND, RVNS provides diversity with solutions in a stochastic nature. It is very useful for large structures of data, unlike VND, which can be very costly (see Algorithm 2.5 on page 31).

Algorithm 2.5: Steps of the Reduced VNS

```
Function RVNS$(x, k_{max}, t_{max})$ ;
1  repeat
2      $k \leftarrow 1$;
3      repeat
4          $x' \leftarrow$ Shake$(x, k)$;
5          NeighbourhoodChange $(x, x', k)$ ;
       until $k = k_{max}$ ;
6      $t \leftarrow$ CpuTime()
   until $t > t_{max}$ ;
```

With the function Shake represented in line 4, a point x' is generated at random from the k^{th} neighbourhood of x, i.e., $x' \in \mathcal{N}_k(x)$. Its steps are given in Algorithm 2.6 on page (32), where it is assumed that points from the $\mathcal{N}_k(x)$ are $\{x^1, \ldots, x^{|\mathcal{N}_k(x)|}\}$.

Algorithm 2.6: Steps of the Shaking function

```
Function Shake($x, x', k$);
1  $w \leftarrow [1+\text{Rand}(0,1) \times |\mathcal{N}_k(x)|]$;
2  $x' \leftarrow x^w$
```

Basic Variable Neighbourhood Search. The Basic Variable Neighbourhood Search (BVNS) is another variant of the Variable Neighbourhood Search. It is an approach balanced between diversity and intensity in the solutions obtained. As was previously explained, Variable Neighbourhood Search is laborious to carry out because it finds a solution only in the current data structure, and while Reduced Variable Neighbourhood Search (RVNS) selects the solution randomly from the best neighbourhood structure, which can greatly reduce the quality of the solution obtained. This Basic Variable Neighbourhood Search (BVNS), selects the next optimal solution from the most suitable neighbourhood structure through an interesting process of choosing any component of the neighbourhood and putting it through some local method to refine and improve the chosen solution. This solution is then made the current candidate solution from the neighbourhood which had been observed in this particular iteration. This saves time by providing a good solution without exhausting one's resources by fully analyzing the neighbourhood structure (see Algorithm 2.7 on page 32).

Algorithm 2.7: Steps of the basic VNS

```
Function VNS($x, k_{max}, t_{max}$);
1  repeat
2      $k \leftarrow 1$;
3      repeat
4          $x' \leftarrow$ Shake($x, k$)              // Shaking ;
5          $x'' \leftarrow$ BestImprovement($x'$)   // Local search ;
6          NeighbourhoodChange($x, x'', k$) // Change neighbourhood ;
       until $k = k_{max}$ ;
7      $t \leftarrow$ CpuTime()
   until $t > t_{max}$ ;
```

General Variable Neighbourhood Search. A General Variable Neighbourhood Search

(GVNS) is formulated when a Variable Neighbourhood Descent (VND) is applied in the parameters of a basic Variable Neighbourhood Search (VNS). This scheme has been found highly successful, although only examples of this search can be found. The steps of the general VNS (GVNS) are given in Algorithm 2.8 on page (33).

Algorithm 2.8: Steps of the general VNS

```
Function GVNS ($x, k'_{max}, k_{max}, t_{max}$);
1 repeat
2     $k \leftarrow 1$;
3     repeat
4         $x' \leftarrow$ Shake($x, k$);
5         $x'' \leftarrow$ VND($x', k'_{max}$) ;
6         NeighbourhoodChange($x, x'', k$);
      until $k = k_{max}$ ;
7     $t \leftarrow$ CpuTime()
  until $t > t_{max}$ ;
```

Skewed Variable Neighbourhood Search. The Skewed Variable Neighbourhood Search (SVNS) allows a broad array of data to be explored with a much more flexible criterion of acceptance than before. This is a very important method for any case that involves a local optimum of a very broad search space. According to this method, we analyze broader neighbourhoods of such a space to get away from a particular optimum, using the criterion of acceptance and moving towards the general optimum of the space. But this exploration can be extremely exhausting and time-consuming. Even if the process is speeded up, reaching new neighbourhoods, no matter how small or large, always requires the process to be started in each new neighbourhood from the beginning. The SVNS method has the advantage of letting solutions move on to even worse ones than the previous. This idea is the basis of all diversification processes. The best result is formulated through empirical and learning processes. Its steps are presented in Algorithms 2.9, 2.10, and 2.11 on page (34).

The KeepBest(x, x') function in Algorithm 2.10 simply keeps the better between x and x'.

Variable Neighbourhood Decomposition Search. The Variable Neighbourhood Decomposition Search (VNDS) is a VNS scheme involving two levels which resolves the optimization

Algorithm 2.9: Steps of neighbourhood change for the skewed VNS

```
Function NeighbourhoodChangeS($x, x'', k, \alpha$);
1 if $f(x'') - \alpha\rho(x, x'') < f(x)$ then
2     $x \leftarrow x''; k \leftarrow 1$
  else
3     $k \leftarrow k + 1$
```

Algorithm 2.10: Steps of the Skewed VNS

```
Function SVNS ($x, k_{max}, t_{max}, \alpha$);
1 repeat
2     $k \leftarrow 1; x_{best} \leftarrow x$;
3     repeat
4         $x' \leftarrow$ Shake($x, k$) ;
5         $x'' \leftarrow$ FirstImprovement($x'$) ;
6         KeepBest ($x_{best}, x$);
7         NeighbourhoodChangeS($x, x'', k, \alpha$);
      until $k = k_{max}$ ;
8     $x \leftarrow x_{best}$;
9     $t \leftarrow$ CpuTime();
  until $t > t_{max}$ ;
```

Algorithm 2.11: Keep the better solution

```
Function KeepBest($x, x'$);
1 if $f(x') < f(x)$ then
      $x \leftarrow x'$ ;
```

difficulties which depend on its decomposition. The basic assumption in this form of VNS lies in the difficulties faced in the optimization processes, for a simple VNS is not appropriate for formulating good solutions in a short time. This variant promotes the basic idea of reducing the search process to a representative subset of the whole of the space and thus it is analyzed more efficiently and in less time than a simple VNS. At every progressive stage, the VNDS selects a sample subset for all the solutions at random and a local method is used to analyze this subset. Only those variables and solutions are selected that display the attributes attached to the main solution. VNS schemes other than this are also applied as local search procedures using this method. The local optimum is redefined at every stage in which an improvement is made to the solution and this involves all the solutions down to the last. The search is considerably strengthened, because it is filtered down to the right solution. The VND is the most reliable form of local search tool. The criterion for stopping the process does end this process at some point, but until then this process continues to repeat itself. As stated about the other variants, the criterion for stopping is characterized by a time limit, a certain number of iterations that can be performed in a series or in between improving solutions which helps us to stop the process when it reaches a stated limit. The VNDS method is greatly renowned, with increasing numbers of applications being discovered. Its steps are presented in Algorithm 2.12 below.

Algorithm 2.12: Steps of VNDS

```
Function VNDS ($x, k_{max}, t_{max}, t_d$);
1  repeat
2      $k \leftarrow 2$;
3      repeat
4          $x' \leftarrow$ Shake $(x, k)$; $y \leftarrow x' \setminus x$;
5          $y' \leftarrow$ VNS$(y, k, t_d)$; $x'' = (x' \setminus y) \cup y'$;
6          $x''' \leftarrow$ FirstImprovement$(x'')$;
7          NeighbourhoodChange$(x, x''', k)$;
       until $k = k_{max}$ ;
   until $t > t_{max}$ ;
```

Chapter 3

Heuristics for Harmonic Means Clustering

Harmonic means clustering is a variant of Minimum sum of squares clustering (sometimes called K-means clustering), designed to alleviate the dependance of the results on the initial choices of solution. In the harmonic means clustering problem, the sum of harmonic averages of the distances from the data points to all the cluster centroids is minimized. In this chapter, a variable neighbourhood search heuristic is proposed for improving it. This heuristic has been tested on numerous datasets from the literature. It appears that the results obtained on standard test instances compare favorably with recent ones from Tabu Search and Simulated Annealing heuristics.

3.1 Introduction

The method for forming natural groupings in data is called data clustering; it is a very important function of machine memory and the recognition of patterns. Clustering [48, 57, 5] is a scientific method which addresses the following very general problem: given the data on a set of entities, find clusters, or groups of these entities, which are both homogeneous and well-separated. Homogeneity means that the entities in the same cluster should resemble one another. Separation means that the entities in different clusters should differ from one another.

There are numerous ways to express homogeneity and/or separation by one or two criteria. In addition, various structures may be imposed upon the clusters, the two most common being hierarchy and partition. Choosing a criterion and constraints defines a clustering problem. If this is done explicitly and rigorously, it takes the form of a mathematical program [39]. Many solution methods exist for most clustering problems. In rare cases, they are exact algorithms which provide proven optimal solutions [64, 6].

Cluster analysis deals with various types of data. However, partitioning in Euclidean space $\mathbb{R}^q$ is only concerned in this thesis. To explain the notation, let $X = \{x_1, \ldots, x_N\}$ be a set of objects or entities to be clustered ($x_i \in \mathbb{R}^q$) , and let C be a subset of X. Then $P_K = \{C_1, C_2, \ldots, C_K\}$ is a partition of X into K clusters if it satisfies: (i) $C_k \neq \emptyset; \quad k = 1, 2, \ldots, K$, (ii) $C_i \cap C_j = \emptyset; \quad i, j = 1, 2, \ldots, K; \quad i \neq j$, and (iii) $\bigcup\limits_{k=1}^{K} C_k = X$.

One of the most popular partitioning problems for points in Euclidean space is the minimum sum of squares clustering (MSSC) [4, 5, 40]. It considers simultaneously the criteria for homogeneity and separation. Minimizing these criteria amounts to replacing each cluster by its centroid while finding the partition which minimizes the sum-of-squares distances from the entities to the centroid of their cluster. A mathematical formulation of the MSSC problem is as follows (see for example [40]): consider a set $X = \{x_1, ..., x_i, ..., x_N\}, \quad x_i = (x_{1i}, ..., x_{qi})$ of N entities in Euclidean space $\mathbb{R}^q$. The MSSC problem is to find a partition of X into K disjoint subsets C_j such that the sum of squared distances from each entity x_i to the centroid c_j of its cluster C_j is the minimum.

Specifically, let P_K denote the set of all partitions of X into K sets. Let partition P be defined as $P = \{C_1, C_2, ..., C_K\}$. Then MSSC can be expressed as:

$$f_{MSSC}(P) = \min_{P \in P_K} \sum_{i=1}^{N} \min_{j=1,\ldots,K} \|x_i - c_j\|^2, \tag{3.1}$$

where the centroid of cluster j is given as $c_j = \frac{1}{|C_j|}\sum_{i \in C_j} x_i$.

The K-Harmonic Means (KHM) clustering problem is similar in several respects to the MSSC problem. Indeed, it considers partitions and minimizes a function of distances to cluster centroids, this last term being understood in a slightly different sense from the above. KHM minimizes the sum of harmonic averages of the distances between each entity and all centroids. It has been observed that the final solution of the MSSC problem obtained by many local search heuristics depends substantially on the initial choice of centroids. However, this does not appear to be the case for the KHM clustering problem [83]. To support this fact, some comparisons between KM and KHM are made to show the impact of initializations on the final solution in the next Chapter.

KHM uses a weight function which allows the same entities to belong to different clusters. A weight function w, recalled below, determines the repartition of the belonging that each entity has in each cluster. The other function used in the KHM algorithm is called the membership function m_{ij}, which assigns each entity or point x_i to a cluster c_j.

It will be seen that the KHM clustering problem is adequately improved by using the basic Variable Neighbourhood Search (VNS) heuristic. For scaling the quality of VNS, it is compared with those results obtained by multi start local search (MLS), Tabu Search (TS) [36] and Simulated Annealing (SA) [35] heuristics. The improvements of the results when the initial data are scaled or normalized are also shown. Finally, a VNS-based heuristic is tested on greater instances than previously used in the literature. For these purposes, instances from the Traveling Salesman Problem (TSP) library [73] are used.

This chapter is organized as follows. The next section presents a very brief review of the K-means algorithm. Then, details the KHM clustering problem and its local search algorithm. Section 3.3 on page 43, show how the suggested VNS heuristic improves upon the local search of KHM. Section 3.4 on page 46 contains computational results. Conclusions are drawn in section 3.5 on page 51.

3.2 K-Harmonic Means clustering problem (KHMCP)

Among numerous heuristics for MSSC, the best known and most often used is the K-means (KM) [25, 55]. From an initial set of centroids, K-means proceeds by reassigning the entities to their closest centroids and updating the cluster centroids until stability is reached. Its steps are given in Algorithm 3.1 below. The basic local search for KHM, described below,

Algorithm 3.1: K-Means algorithm (KM) for the MSSC problem

```
Function KM ($X, K, Maxit, N, C, z$)
1  Choose initial centroids $c_k$ ($k = 1, \ldots, K$)
2  $l \leftarrow 0$
3  repeat
4      $l \leftarrow l + 1$
5      for $i := 1, \ldots, N$ do
6          $m(x_i) \leftarrow \text{argmin}_{j\in\{1,\ldots,K\}}(\|x_i - c_j\|_2)^2$
7      $z = f_{MSSC}$ as in (3.1)
8      for $j := 1, \ldots, K$ do
9          Calculate centroid $c_j$
   until $m$ does not change or $l$ =$Maxit$
```

is quite close to KM. Zhang and his colleagues [83, 82] have established the K-Harmonic Means. KHM evolves from the optimization criteria which have been built on the concept of Harmonic mean. K-Harmonic Means (KHM) offers a more promising way of finding much better and quickly clustering solutions. It also surpasses the k-means (KM) as it will be shown later. The KHM objective function is calculated by using the harmonic average which in many situations gives the truest average. The harmonic average (HA) is always the least of three Pythagorean means (including Arithmetic average (AA) and Geometric average (GA)) for positive sets that contain non-equal values. However the AA is always the greatest. The HA tends (compared to the the AA) to reduce the impact of large outliers and enlarge the impact of small ones.

In some certain problems such as the speed average, the HA is the truest mean. It is very often that the AA is mistakenly used instead of the HA [20]. The harmonic average (HA) of

K numbers $a_j \in \mathbb{R} \quad (j = 1, ..., K)$ is defined as:

$$HA = \frac{K}{\sum_{j=1}^{K} \frac{1}{a_j}} \quad . \tag{3.2}$$

The HA is used to measure the distances between entities and centroids. In addition, the pth power of the Euclidean norm is used as a distance function. Hence, Equation 3.2 will be:

$$HA_i(K, p) = \frac{K}{\sum_{j=1}^{K} \frac{1}{||x_i - c_j||^p}}, \quad \forall \quad i = 1, ..., N \quad . \tag{3.3}$$

Then the objective function for KHM is:

$$f_{KHM}(K, p) = \min \sum_{i=1}^{N} HA_i(K, p) = \min \sum_{i=1}^{N} \frac{K}{\sum_{j=1}^{K} \frac{1}{||x_i - c_j||^p}} \quad . \tag{3.4}$$

The most popular iterative procedure for solving this problem is the K-Harmonic means local search [82], recalled in Algorithm 3.2. Let $d_{ij} = ||x_i - c_j||^p$, then the recursive updating [54] rule for each centroid can be derived easily by the partial derivative of f_{KHM} as displayed in 3.4 with respect to c_j equal to zero. i.e.,

$$\frac{\partial f_{KHM}}{\partial c_j} = 0, \forall j = 1, \ldots, K \tag{3.5}$$

$$\Rightarrow \frac{\partial f_{KHM}}{\partial c_j} = \sum_{i=1}^{N} \frac{-2K(x_i - c_j)}{(\sum_{j=1}^{K} \frac{1}{d_{ij}})^2 d_{ij}^2} = 0 \tag{3.6}$$

To get the new centroid, Formula 3.6 needs to be reshaped to obtain:

$$c_j^{(new)} = \frac{\sum_{i=1}^{N} \frac{x_i}{d_{ij}^2 (\sum_{j=1}^{K} \frac{1}{d_{ij}})^2}}{\sum_{i=1}^{N} \frac{1}{d_{ij}^2 (\sum_{j=1}^{K} \frac{1}{d_{ij}})^2}} \tag{3.7}$$

According to [37], the membership function and the weight function which KHM uses are defined as follows:

$$m_{KHM}(c_j/x_i) = \frac{\|x_i - c_j\|^{-p-2}}{\sum_{j=1}^{K} \|x_i - c_j\|^{-p-2}}, \quad \forall \quad i = 1, ..., N, \quad \forall \quad j = 1, ..., K \quad , \tag{3.8}$$

$$w_{KHM}(x_i) = \frac{\sum_{j=1}^{K} \|x_i - c_j\|^{-p-2}}{(\sum_{j=1}^{K} \|x_i - c_j\|^{-p})^2}, \quad \forall \quad i = 1, ..., N \quad , \tag{3.9}$$

Thus, from Equations 3.8 and 3.9, the centroids equation 3.7 can be rewritten in the following formula [37]

$$c_j^{(new)} = \frac{\sum_{i=1}^{N} m_{KHM}(c_j/x_i) \cdot w_{KHM}(x_i) \cdot x_i}{\sum_{i=1}^{N} m_{KHM}(c_j/x_i) \cdot w_{KHM}(x_i)}, \quad \forall \quad j = 1, ..., K \quad . \tag{3.10}$$

The local search algorithm starts by generating K-centroids chosen at random among the given entities $x_i \quad (i = 1, ..., N)$. From Equations 3.8, 3.9, and 3.10 new centroids are obtained. This process is repeated until the difference between centroids in two consecutive iterations is less than ε (a small number) or a maximum number of iterations is reached (see Algorithm 3.2 below). The KHM FORTRAN code details is explained in Appendix A.

Algorithm 3.2: The local search algorithm for KHM problem

```
Function KHM ($X, K, Maxit, \varepsilon$,N,C,z)
1  $C^{(new)} = \{c_1, c_2, \dots, c_K\}$      // $K$ centroids are chosen from $X$
2  $i \leftarrow 0$                          // i-iteration counter
3  repeat
4      $i \leftarrow i + 1$
5      $C \leftarrow C^{(new)}$
6      $z \leftarrow f_{KHM}(C)$ as in 3.4
7      Calculate $m$ as in 3.8 and $w$ as in 3.9 for all entities
8      Find new centroids $c_j^{(new)}$, $j = 1, \dots, K$ as in 3.10
   until $(\|c_j^{(new)} - c_j\| \leq \varepsilon, \ \forall j = 1, ..., K)$ or $i = Maxit$
```

3.2.1 Multi-Start KHM

To keep the best value of the objective function obtained after several iterations, we design a multistart (MLS) algorithm, in which the local search algorithm can easily be embedded. Thus, the (KHM) are applied several times, keeping the best solution (best local minimum) found so far. For some data sets MLS obtain very good results. Details are shown in Algorithm 3.3 below.

Algorithm 3.3: The multi-start local search for KHM clustering (MLS)

```
Function MLS ($X, K, C, Maxit, \varepsilon$)
1  $z_{opt} \leftarrow 10.e^{20}$
2  $i \leftarrow 0$
3  repeat
4      $i \leftarrow i + 1$
5      Generate solution $X$ at random
6      $z \leftarrow$ KHM ($X, K, C, Maxit, \varepsilon$)
7      if $z \leq z_{opt}$ then
8          $z_{opt} \leftarrow z$
   until $i = Maxit$
```

3.2.2 Tabu Search

A Tabu Search based heuristic for solving KHM (TabuKHM for short) is proposed in [36]. For this purpose two kinds of move are used: (i) a random swap of a current centroid with an existing facility and (ii) a so-called logical swap which consists of merging two close clusters and splitting them in two again. The clusters to be merged are selected according to their utility, as defined in [67]. A cluster whose utility is low (< 1) is merged with one with high utility (> 1). In addition, both moves may be rejected if the generated random number $r < s$, where $r \in (0, 1)$ and s is a parameter set by the analyst. The Tabu List is updated in the usual way. The results of such a probabilistic Tabu Search are reported and compared with a VNS based heuristic in the computational results section of this paper.

3.2.3 Simulated Annealing

A Simulated Annealing (SA) based heuristic for solving KHM is proposed by the same authors in [35]. The same types of move as for TabuKHM are used. The difference is that the decision to move or not is made according to the rule defined by a cooling schedule function (as is usual in SA-based heuristics), and without use of a Tabu List. The results obtained by SA are also reported in section 4 , as they are given in [35]. Both of these algorithms are explained in the sections 2.3.1 and 2.3.2.

3.3 VNS for solving KHM

Variable neighbourhood search (VNS) is a metaheuristic designed for solving combinatorial and global optimization problems. The basic idea is to proceed to a systematic change of neighbourhood within a local search algorithm [60, 41, 42, 16]. The set of neighbourhoods is usually induced from a metric function introduced into the solution space. The algorithm centres the search around the same solution until another solution better than the incumbent is found and then recentres the search, or jumps there.

Let a set $\mathcal{N}_k$ denote the set of neighbourhood structures ($k = 1, ..., k_{max}$), and $\mathcal{N}_k(x)$ the set of solutions in the k^{th} neighbourhood of x. To calculate these neighbourhoods, according to Definition 2.1 in page 29, it implies that:

$$\begin{aligned} |\mathcal{N}_k(x)| &= \binom{K}{k} \;.\; \binom{N}{k} \\ &= \frac{K!}{k!(K-k)!} \;.\; \frac{N!}{k!(N-k)!} \\ &= \frac{K(K-1)(K-2)...(K-k-1)}{k!} \;.\; \frac{N(N-1)(N-2)...(N-k-1)}{k!} \\ &\approx K^k \;.\; N^k \end{aligned} \tag{3.11}$$

For example if $x \in S$ is a solution of the optimization problem and $x = \{a_1, a_2, ..., a_N\}$ then if

K is the number of clusters, the first neighbourhood of x is:

$$\begin{aligned}\mathcal{N}_1(x) \quad = \quad & \{\{1, a_2, ..., a_N\}, \{2, a_2, ..., a_N\}, ... \quad , \{K, a_2, ..., a_N\}, \\ & \{a_1, 1, a_3, ..., a_N\}, \{a_1, 2, a_3, ..., a_N\}, ... \quad , \{a_1, K, a_3, ..., a_N\}, \\ & \vdots \\ & \{a_1, a_2, ..., a_{N-1}, 1\}, \{a_1, a_2, ..., a_{N-1}, 2\}, ... \quad , \{a_1, a_2, ..., a_{N-1}, K\},\end{aligned}$$

It concludes that $|\mathcal{N}_1(x)| \approx K.N$. To produce $\mathcal{N}_2(x)$, the shaking step in Algorithm 3.4 gives 2 sets of new centroids. This will be made easily for the rest of neighbourhoods as it is shown in the previous example. Then the steps of the basic variable neighbourhood search (BVNS) are given at Algorithm 3.6.

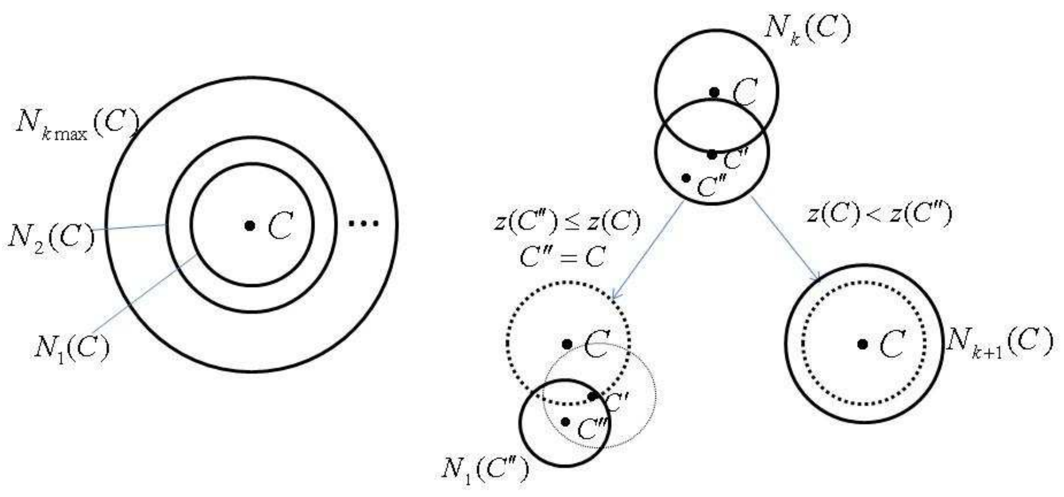


Figure 3.1: Basic scheme of variable neighbourhood search

VNS has already been applied for solving MSSC and Fuzzy MSSC [10, 40]. Beside the alternate neighbourhood structure, as used in the KM heuristic, two more neighbourhood structures were used: H-means and J-means. H-means can be applied as follows. Let $\{C_1, ..., C_K\}$ be an initial partition which is chosen randomly. An entity x_j that belongs to cluster C_l will be reallocated to different cluster $C_i(l \neq i)$. This process is called re-allocate or re-assign. On the other hand, J-means works as re-locate by relocating entities that do not coincide with a cluster centroid and making them the new centroids. Actually, J-means comes from jumping neighbourhoods of the current solution. Both H-means and J-means are used within a nested variable neighbourhood descent (VND) strategy. More precisely, at any point in a J-means neighbourhood, K-means and H-means are used one after the other iteratively, until no improvement is possible.

To apply the BVNS for solving KHM the shaking process is used, as given in Algorithm 3.4 on page 45. It simply selects the new K centroids randomly from the set of existing entities X where k is given a VNS neighbourhood parameter. Note that in fact it selects a random point from the J-means neighbourhood structure [40]. In Algorithm 3.4, r represents a number with uniform distribution from (0,1). Therefore, $r1$ denotes the index of a chosen centroid and $r2$ denotes the index of random entity.

Algorithm 3.4: Shaking step

```
Function Shaking ($X, k, C$)
1  $j \leftarrow 0$                          // initializing iteration counter
2  repeat
3      $j \leftarrow j + 1$
4      $r1 \leftarrow \lfloor (K-j+1)*r \rfloor$     // a cluster is chosen at random
5      $r2 \leftarrow \lfloor (N-j+1)*r \rfloor$     // an entity is chosen at random
6      for $i := 1, \ldots, q$ do
7          $c(r1, i) \leftarrow x(r2, i)$
   until $j = k$
```

After shaking, random centroids are obtained to start the KHM local search (KHMLS). The previous centroids will be replaced by the new one if the solution is improved. Otherwise, another solution is generated at random from $\mathcal{N}_{k+1}$. In other words, $k + 1$ new centroids will be selected from among the existing entities. The search continues this inner loop until a certain predefined number k_{max} of neighbourhoods is reached (see Figure 3.1). It has been observed that making k_{max} larger than K would not be productive. In fact, exchanging the K centroids, as in J-means neighbourhood, will produce a solution farthest from the current one, with respect to the J-means neighbourhood. Therefore it is set in Algorithm 3.6 (on page 46) that $k_{max} \leftarrow K$. To know how the decision for choosing the centroids in each iteration is made, Algorithm 3.5 is designed and this is used in the main code.

The pseudo-code of the basic VNS is given in Algorithm 3.6 (on page 46). The outer loop

Algorithm 3.5: Neighbourhood change or move or not function

```
Function NeighbourhoodChange ($C, C', k$)
1  if $z(C') < z(C)$ then
2      $C \leftarrow C'$; $k \leftarrow 1$                    // make a move
   else
3      $k \leftarrow k + 1$                                 // next centroid
```

of the VNS is performed until the running time reaches t_{max} (a parameter) seconds. t_{max} is equal to 100 independent calls of KHM local search given in Algorithm 3.2. Note that the same stopping condition was used in [10, 40]. For more details about the basic VNS and other VNS methods see [41, 42].

Algorithm 3.6: Steps of the basic VNS

```
Function VNS($X, K, k_{max}, t_{max}, C$)
1  repeat
2      $k \leftarrow 1$                                   // the neighbourhood index
3      repeat
4          $C' \leftarrow$ Shake($X, k, C$)                  // Shaking
5          $C'' \leftarrow$ KHM ($X, K, C', Maxit, \varepsilon$)     // Local search
6          NeighbourhoodsChange($C, C'', k$) // Change centroid
       until $k = k_{max}$
7      $t \leftarrow$ CpuTime()
   until $t > t_{max}$
```

Since the basic VNS is easy to formulate in applications, other VNS extensions have not been used. The other reason is that the basic VNS is a powerful algorithm and has the best solutions in most of the experiments although it is terminated by MLS time.

3.4 Computational results

Computer. All experiments were performed on a personal computer Intel(R) Core(TM)2 with 3.24GB of RAM and a speed of 2.40GHz. All the methods were coded on Lahey/Fujitsu FORTRAN 95.

Test instances. the following test instances are chosen: (i) *Iris* which has 150 entities in

4-dimensions with $K = 3$; (ii) *Glass* which has 214 entities in 9-dimensional space with $K = 2$; (iii) *Wine* which has 178 entities in 13-dimensions with $K = 3$, and (iv) *Breast-cancer* which has 699 entities in 10-dimensions with $K = 2$. For more details about them see [12]. Also, two datasets are used, obtained from [73] and [12]. They are called (v) dataset 1, which has 1060 entities in 2-dimensions and (vi) dataset 2, which has 2310 entities in 19-dimensions.

Parameters. The value $\varepsilon = 0.01$ is used in all the algorithms. However, because of the sensitivity of ε in some rare situations (like *Wine* dataset if the power $p = 3.5$), it might be changed to different values as it appears later. For that reason, other conversion criteria are used by solutions instead of ε (i.e. the difference between 2 solutions is identical) to check the results which may be concluded by coincidence. This new method is applied for *Iris* dataset. See Table 3.2. In Algorithm 3.2 and Algorithm 3.3, the $Maxit = 180$. In Algorithm 3.6, the t_{max} is equal to the time that KHM spends on 100 independent calls and $k_{max} = K$. In this way, a user-friendly VNS heuristic will be obtained, since the single parameter is t_{max}. For dataset1 and dataset2, the power of the KHM objective function is $p = 2$ for each number of clusters K.

The following tables compare basic VNS algorithm with a previous work using the same data sets.

Comparison with Tabu Search. In the experiments the three data sets were used in [36]: *Iris*, *Glass* and *Wine*. In Table 3.1 Multistart local search (MLS) and VNS are created and compared with TabuKHM from the literature [36] on these data sets. The name of the instance and the value of K are reported in the first column of Table 3.1, the value of p in the second, and objective function values obtained by three methods in the next three columns. In columns 6, 7 and 8, the MSSC objective function value is calculated with the partition obtained by the corresponding KHM method. Columns 9 and 10 report the time when the best solutions given in the table are reached by MLS and VNS respectively. The CPU time for TabuKHM is not reported in [36]. In the last column the computing time spent on $Maxit = 100$ independent calls of LS is given. That time used is t_{max} for the VNS heuristic.

		KHM objective			MSSC objective					
D	p	TabuKHM	MLS	VNS	TabuKHM	MLS	VNS	t_{MLS}	t_{VNS}	t_{Max}
	1.5	182.293	182.071	182.066	81.343	80.159	79.254	0.078	0.016	0.203
	2	181.728	181.519	181.518	79.454	79.254	79.026	0.031	0.031	0.078
Iris	2.3	182.252	182.064	182.064	79.061	78.856	78.856	0.125	0.016	0.125
K=3	2.5	183.037	182.866	182.865	79.062	78.856	78.851	0.109	0.016	0.140
	3	186.827	186.699	182.699	79.549	78.851	78.851	0.016	0.031	0.109
	3.5	193.637	193.481	193.476	80.400	78.851	78.851	1.078	0.203	1.234
	1.5	642.9	642.877	642.874	851.209	826.556	826.556	0.109	0.250	0.281
	2	1112.8	1112.771	1112.769	828.540	820.782	820.782	0.031	0.078	0.109
Glass	2.3	1616.3	1616.254	1616.252	824.323	820.782	820.526	0.094	0.078	0.156
K=2	2.5	2105.1	2105.144	2105.143	825.382	820.028	819.629	0.109	0.078	0.109
	3	4247.9	4247.938	4247.938	846.686	821.312	821.312	0.031	0.016	0.094
	3.5	8870.5	8870.487	8870.473	918.330	831.553	831.553	0.094	0.000	0.172
	1.5	4.0756e5	399360.781	399360.781	2396.8e3	2371841.75	2371841.75	0.094	0.250	1.063
	2	5.3926e6	5388246.00	5388245.50	2402.7e3	2379535.25	2379535.25	0.156	0.063	0.406
Wine	2.3	26.216e6	26216142.0	26216138.0	2438.2e3	2412870.25	2412870.25	0.156	0.313	0.500
K=3	2.5	75.84e6	75840168.0	75840152.0	2489.1e3	2416444.75	2416444.75	0.109	0.563	0.672
	3	1058.8e6	1.05884640e9	1.05884621e9	2687.4e3	2643674.00	2643674.00	0.125	0.203	0.484
	3.5	14340e6	1.43466772e10	1.43464008e10	2733.7e3	2658181.75	2658181.75	1.438	0.031	3.688

Table 3.1: *Comparison of three heuristics*

		KHM objective			MSSC objective					
D	p	TabuKHM	MLS	VNS	TabuKHM	MLS	VNS	t_{MLS}	t_{VNS}	t_{Max}
	1.5	182.293	182.093	182.066	81.343	80.160	79.254	0.043	0.000	0.203
	2	181.728	181.525	181.518	79.454	79.254	79.026	0.053	0.003	0.090
Iris	2.3	182.252	182.102	182.064	79.061	78.859	78.856	0.098	0.012	0.146
K=3	2.5	183.037	182.866	182.865	79.062	78.858	78.851	0.165	0.012	0.218
	3	186.827	186.701	182.699	79.549	78.853	78.851	0.243	0.016	0.289
	3.5	193.637	193.482	193.476	80.400	78.851	78.851	1.056	0.089	1.170

Table 3.2: *Comparison of three heuristics for Iris dataset using solutions conversion criteria*

It appears that the previous TabuKHM results are improved for all values of p in the *Iris* data set. For the *Glass* dataset, the results are almost the same for all three methods. Indeed, the problem with $K = 2$ seems to be that easy and optimal solutions are probably obtained by all methods. Regarding the *Wine* data set, the results are improved by VNS, except for $p = 3.5$. It is noticed that in this case, the KHM local search never converged in 100 restarts when the usual value $\varepsilon = 0.01$ is used. It is found that this fact is explained by the sensitivity of parameter ε when the power p is large, i.e., the denominators in formulas 3.8 and 3.9 become too small. The results reported in Table 3.1 (for $p = 3.5$) are obtained by setting $\varepsilon = 10$. In this way the instability caused by dividing by a number close to 0 is avoided.

Some interesting observations can be deduced from columns of Table 3.1 which report the values of the MSSC objective (values of the best partitions obtained by the heuristics and KHM are calculated using the MSSC objective): (i) For the *Iris* dataset, the quality of the

MSSC solution increases with p; the optimal solution of 78.8514 [64] is reached with larger values of p ($p = 2.5, 3$ and $p = 3.5$). (ii) For the *Glass* dataset, the quality of the MSSC solution first increases and then declines when p increases; (iii) For the *Wine* dataset the quality declines with p. This intriguing result appears to be worth further analysis.

In [36], the TabuKHM has also been tested after the normalization of all the entities in the data sets, i.e.,

$$x_i^* = \frac{x_i - \min\limits_{i=1,\dots,N}\{x_i\}}{\max\limits_{i=1,\dots,N}\{x_i\} - \min\limits_{i=1,\dots,N}\{x_i\}}, \quad \forall i = 1, \dots, N. \tag{3.12}$$

MLS and VNS are also applied to these normalized data sets. In Table 3.3 comparative results for $p = 2.3$ are given. It appears that the VNS provides solutions with the smallest objective function values. An exception is the dataset *Glass*, where the objective function for the Tabu Search heuristic is twice as small as those obtained by MLS and VNS. Such a difference exceeds by far what one would expect. In order to check this result, I asked the corresponding author of [36] to provide me with a full description of the solution. He declined to do so.

Datasets	TabuKHM	MLS	VNS	t_{MLS}	t_{VNS}	t_{Max}
Iris	7.012	7.004	6.982	0.047	0.016	0.094
Glass	16.629	34.158	34.158	0.031	0.000	0.172
Wine	49.501	49.022	48.990	0.156	0.016	0.194

Table 3.3: *Comparison of results with Tabu Search when the datasets are normalized and* $p = 2.3$

Comparison with Simulated Annealing. The same authors used *Iris*, *Wine* and *Breast-cancer* test instances for improving the KHM local search by using a simulated annealing approach in [35]. They called their heuristic SAKHM. In their computational results section, they give results for $p = 3.5$ only on the original and normalized data sets. In contrast with [36], they also report on the CPU times used in the search. Their results are only used in the following tables.

In Tables 3.4 and 3.5, comparisons between methods are given on original and normalized data sets in turn. In the first case, the final solution is recalculated using the MSSC objective.

It appears that VNS and MLS provide solutions of better quality than SAKHM does, and do so in significantly smaller CPU times. The reason appears to be that the step in SA which merges and splits current clusters according to their utility (based on the distortion and utility approaches from [67]) is probably not very efficient.

Datasets	SAKHMC	MLS	VNS	t_{MLS}	t_{VNS}	t_{Max}	t_{SAKHMC}
Iris	80.32	78.851	78.851	1.078	0.203	1.234	15.25
Wine	2720.0e3	2658181.75	2658181.75	1.438	0.031	3.688	20.38
B-cancer	20.92e3	20091.9473	20091.9473	0.641	0.250	0.671	77.78

Table 3.4: *Comparison of results with Simulated Annealing search when* $p = 3.5$ *based on the MSSC objective function*

Datasets	SAKHMC	MLS	VNS	t_{MLS}	t_{VNS}	t_{Max}	t_{SAKHMC}
Iris	7.11	6.990	6.990	0.109	0.156	0.547	15.48
Wine	49.95	48.989	48.989	0.031	0.078	0.172	20.52
B-cancer	258.36	255.532	255.532	0.531	0.422	0.578	71.58

Table 3.5: *Comparison of results with Simulated Annealing when the datasets are normalized and* $p = 3.5$

Comparison between MLS and VNS on large datasets. In previous tables VNS shows slightly better performance than MLS. In the last set of experiments, and to check how these two heuristics compare in large test instances, two datasets were used. Dataset 1, called the *Drilling Problem*, has 1060 entities in 2-dimensions [73]. Dataset 2 is called the *Image Segmentation* with 2310 entities in 19-dimensions [12]. The results are given in Tables 3.6 and 3.7.

K	MLS	VNS	Improvements	t_{MLS}	t_{VNS}	t_{Max}
10	8.88987136e9	8.88986829e9	0.00%	8.906	14.250	22.344
20	6.99046605e9	6.99042970e9	0.00%	31.000	34.578	61.219
30	6.07008205e9	6.02638234e9	0.72%	75.047	3.734	89.188
40	5.45612134e9	5.43608781e9	0.37%	111.172	84.578	120.438
50	5.08445184e9	4.97067776e9	2.24%	136.781	14.016	148.203

Table 3.6: *Comparison on Dataset 1*: $n = 1060,\ q = p = 2$

It appears that, for a small number of clusters, both methods are similar. For larger K, i.e., for more difficult problems where there are many local minima, VNS obtains better results.

K	MLS	VNS	Improvements	t_{MLS}	t_{VNS}	t_{Max}
10	36186608.0	36186488.0	0.00%	15.016	47.922	55.922
20	32377326.0	32329884.0	0.15%	17.813	68.484	139.203
30	31412122.0	30867270.0	0.17%	71.672	177.656	213.969
40	30768640.0	30127940.0	2.08%	101.797	253.953	319.938
50	28922248.0	27687874.0	4.27%	264.203	320.625	390.219

Table 3.7: *Comparison on Dataset 2*: $n = 2310,\ q = 19,\ p = 2$.

3.5 Conclusion

This chapter proposes that a variable neighbourhood search based heuristic is used for solving the K-harmonic means clustering problem; it was initially introduced in [83, 82]. The neighbourhoods consist of centroid to entity moves, similar to those used in the J-Means heuristic [40] for solving the Minimum sum of squares clustering. In a series of test instances often used in the literature, a considerably better performance is obtained using VNS than with two recent metaheuristic based methods: Tabu Search [36] and Simulated Annealing [35]. Moreover, the results for much larger test instances than previously used in the literature are presented. Therefore, this method may be considered as a new state-of-the-art heuristic for solving the K-harmonic means clustering problem.

Chapter 4

Degeneracy of harmonic means clustering

It is well known that some local search algorithms for K-clustering problems could stop at a solution with fewer clusters than the desired K. Such solutions are called degenerate. In this chapter, I first show that the K-Harmonic Means heuristic has this property, although it does not have the same initialization sensitivity as the K-Means heuristic (for solving the Minimum sum-of-squares clustering problem). I then found that two types of degenerate solutions can be found in the K-Harmonic Means heuristic and provide counter-examples of both. I also propose a simple method to remove degeneracy during the execution of the K-Harmonic Means algorithm (KHM) and use it within a recent variable neighbourhood search (VNS) based heuristic. Extensive computational analysis, performed on the usual test instances from the literature, shows significant improvement obtained with my simple degeneracy correcting method, used within both KHM and VNS.

4.1 Introduction

One of the most popular models for partitioning points in Euclidean space is the minimum sum-of-squares clustering (MSSC) model [4, 5, 40]. (see 3.1 on page 36). Then the MSSC can be expressed as follows:

$$f_{MSSC}(P) = \min_{P \in P_K} \sum_{i=1}^{N} \min_{j=1,\ldots,K} \|x_i - c_j\|^2, \tag{4.1}$$

where the centroid of cluster j is given as $c_j = \frac{1}{|C_j|} \sum_{i \in C_j} x_i$.

The most popular heuristics for solving minimum sum-of-squares clustering (MSSC) alternate two types of variable: for fixed centroids (location), the best assignment of entities (clusters) are found, and for a given K allocations (clusters), the best centroids are found. Such heuristics are known as Alternate (ALT) heuristics. Used for solving MSSC, the ALT heuristic is called K-Means (see Algorithm 3.1 on page 39).

Most alternate heuristics have an undesirable property known as degeneracy [59, 17]: one or more groups of entities (new facilities) become empty during its execution. In other words, the better solution in the next iteration of ALT may be found but with a lower number of clusters (new facilities). Clearly, such solutions may easily be improved by adding a new centroid (facility) at the location of any unoccupied existing facility. Papers which investigate the reason for the deterioration of solution quality obtained by ALT heuristics mostly pay attention to the choice of initial points. There are more than a dozen papers devoted to initialization of K-Means alone and it is still a subject of debate (see e.g. [68, 50, 71, 76]). Recent Harmonic means clustering [83] is designed to show that the solution quality of the ALT heuristic for the K-Harmonic Means clustering problem (KHMCP) depends less on the choice of its initial solution. This fact is empirically confirmed in the next section. The natural question is then whether K-Harmonic Means heuristic (KHM), the most popular heuristic for solving KHM problem, poses the problem of degeneracy as well.

In this chapter I show that the KHM method could also contain two types of degenerate solution: (i) Type-1, when the cluster centre has no entities allocated to it; (ii) Type-2, when two cluster centres coincide, or they are at a distance less than an arbitrary small number ε. I then suggest an efficient and fast method for removing empty clusters immediately when they occur within KHM heuristics. Such a procedure is induced into the VNS based heuristic [3], which represents a current state-of-the art heuristic for KHMCP. In order to understand degeneracy better, I performed an extensive computational analysis on test instances from the literature. It shows also that KHM contains degeneracies of a smaller degree than the K-Means heuristic does.

This chapter is organised as follows. In the next section, for completeness, I give pseudo-codes for the ALT procedures in solving the K-Harmonic Means clustering problem (KHMCP). In the same section I show empirically that KHM is indeed less sensitive on the initial solution. I also prove by constructing counter-examples that KHM could stop at the degenerate solutions of both Type-1 and Type-2. At the end of this section, I propose a method for removing degeneracy. In section 4.3 on page (65), I show the impact of removing degeneracy on variable neighbourhood search (VNS) and Multi-Starts of KHM (MLS). In section 4.4 on page (68), I perform extensive computational analysis. Section 4.5 on page (72) concludes the chapter.

4.2 Degeneracy of K-Harmonic Means clustering

K-Harmonic Means clustering problem (KHMCP). To make it easier to the reader, I repeat some notation mentioned in previous chapter. In the KHMCP the sum of harmonic averages of the distances between each entity and all centroids is minimized:

$$f_{KHM}(p) = \min \sum_{i=1}^{N} HA_i(K, p) \tag{4.2}$$

where:

$$HA_i(K, p) = \frac{K}{\sum_{j=1}^{K} \frac{1}{||x_i - c_j||^p}}, \quad \forall \quad i = 1, ..., N. \tag{4.3}$$

The parameter p is a power of the Euclidean norm which is used as a distance function.

***K*-Harmonic Means (KHM) algorithm.** The most popular heuristic for solving KHMCP is of the alternate type, which will be referred to as the K-Harmonic Means (KHM) [83, 37]. For the sake of completeness, I recall here the steps of the KHM heuristic. The set of variables is naturally divided into a set of locations of cluster centres and a set of membership (allocation) variables of each entity. KHM uses a weight function which allows the same entities to belong to different clusters. A weight function w_i, recalled below, determines the partition of the belonging which each entity has in each cluster. In contrast to the K-Means algorithm (KM) (see Algorithm 3.1), which gives equal weight (i.e. w_i=1) to all data , the KHM algorithm varies the weights at each step. The other function used in the KHM algorithm is called the membership function m_{ij} which assigns each entity or point x_i to a cluster c_j. This function should satisfy the following:

(i) $\sum_{j=1}^{K} m_{ij}=1 \quad \forall \quad i = 1, ..., N;$ (ii) $0 \le m_{ij} \le 1 \quad \forall \quad i = 1, ..., N, \quad \forall \quad j = 1, ..., K.$

The membership function and the weight function which it uses are defined as follows:

$$m_{KHM}(x_i/c_j) = \frac{\|x_i - c_j\|^{-p-2}}{\sum_{l=1}^{K} \|x_i - c_l\|^{-p-2}}, \quad \forall \quad i = 1, ..., N, \quad \forall \quad j = 1, ..., K \quad , \tag{4.4}$$

$$w_{KHM}(x_i) = \frac{\sum_{j=1}^{K} \|x_i - c_j\|^{-p-2}}{\left(\sum_{j=1}^{K} \|x_i - c_j\|^{-p}\right)^2}, \quad \forall \quad i = 1, ..., N \quad , \tag{4.5}$$

where the centroids are given by the formula [83, 82, 37]:

$$c_j^{(new)} = \frac{\sum_{i=1}^{N} m_{KHM}(c_j/x_i) \cdot w_{KHM}(x_i) \cdot (x_i)}{\sum_{i=1}^{N} m_{KHM}(c_j/x_i) \cdot w_{KHM}(x_i)}, \quad \forall \quad j = 1, ..., K \quad . \tag{4.6}$$

The local search algorithm KHM starts by generating K-centroids chosen at random from among the given entities $x_i \quad (i = 1, ..., N)$. From Equations (4.4), (4.5), and (4.6) the new centroids are obtained. This process is repeated until the difference between the centroids in two consecutive iterations is less than ε (a small number) or a maximum number of iterations is reached (see Algorithm 4.1).

Algorithm 4.1: The local search algorithm KHM for KHMCP

```
Function KHM $(X, K, Maxit, \varepsilon, N, C, z)$
1  $C^{(new)} = \{c_1, c_2, \ldots, c_K\}$      // $K$ centroids are chosen from $X$
2  $\ell \leftarrow 0$                          // $\ell$-iteration counter
3  repeat
4      $\ell \leftarrow \ell + 1$; $C \leftarrow C^{(new)}$
5      $z \leftarrow f_{KHM}(C)$ as in (4.2)
6      for $i = 1, ..., N$ do
7          for $j = 1, ..., K$ do
8              Calculate $m(c_j/x_i)$ as in (4.4)
9          Calculate $w(x_i)$ as in (4.5)
10     for $i = 1, ..., n$ do
11         for $j = 1, ..., K$ do
12             Find new centroids $c_j^{(new)}$, as in (4.6)
   until $(\|c_j^{(new)} - c_j\| \leq \varepsilon,\ \forall j = 1, ..., K)$ or $\ell = Maxit$
```

Sensitivity on the initial solution. As noted above, the KHMCP is introduced to avoid the sensitivity of choosing the initial centroids of the MSSC [83, 82, 54]. In order to check this, computational analysis is performed on several well-known test instances from the literature (more detailed description of these test instances is given in section 4.4). Table 4.1 shows the differences between the worst and best objective function values obtained with 100 restarts of KM and KHM heuristics in turn. Since the objective functions of these two problems are different, crisp partitions obtained by KHM are taken and found corresponding MSSC objective function values. In this way, it is easier to compare the influence of the initial solutions on the final solution of KM and KHM.

The first and the second columns of Table 4.1 display the number of entities and the corresponding dimension of the data set, respectively. The desired number of clusters is shown in column 3. Columns 5 and 6 show the worst and best values among the 100 restarts respectively. The last column gives the difference between the worst (largest) and the best (smallest) values, such as were obtained and give the % difference between 2 algorithms calculated as:

$$\frac{KM - KHM}{KM}.100 \tag{4.7}$$

DATASET	DIM	M	ALG	WORST-SOL	BEST-SOL	DIFF	DEV %
Ruspini (75)	2	3	KM KHM	50298.04 12415.12	10126.72 10126.72	40171.32 2288.39	94.30
Iris (150)	4	3	KM KHM	145.53 78.85	78.85 78.85	66.68 0.00	100
Wine (178)	13	3	KM KHM	2633555.33 2371841.59	2370689.69 2371841.59	262865.64 0.00	100
Glass (214)	9	2	KM KHM	1240.11 820.03	819.63 819.63	420.48 0.40	99.90
B-Cancer (699)	10	50	KM KHM	7700.88 7298.29	6112.12 5954.09	1588.76 1344.20	15.39
		100	KM KHM	5853.25 5028.09	4348.77 4348.92	1504.48 679.17	54.86
TSP (1060)	2	50	KM KHM	349545617.68 293226666.88	275703293.57 257897808.70	73842324.11 35328858.18	52.16
		100	KM KHM	157827133.11 122563406.21	111301083.09 102361445.84	46526050.02 20201960.37	56.58
I-Segmentation (2310)	19	50	KM KHM	4182208.78 3182598.13	2819337.21 2294420.45	1362871.57 888177.68	34.83
		100	KM KHM	2908213.19 1947788.50	1839231.27 1340153.25	1068981.92 607635.25	43.16
TSP (3038)	2	50	KM KHM	113402496.67 105470392.38	99913944.85 100278655.58	13488551.83 5191736.80	61.51
		100	KM KHM	58159312.92 50614550.76	50568302.39 48540001.30	7591010.53 2074549.46	72.67

Table 4.1: *MSSC objective functions for* KM *and* KHM *partitions obtained in 100 restarts*

Table 4.1 confirms that the final solution of KHM is not as sensitive as the KM on the choice of the initial solution, since the differences between worst and best solutions obtained by KHM are much smaller than the differences obtained by the KM heuristics. Note also that in some cases better objective function values are obtained with KHM, despite the fact that MSSC problems are considered (see, e.g., the TSP-1060 dataset).

4.2.1 Degeneracy of KHM

In this subsection I show, by counter-example that the solution obtained by KHM could also be degenerate. I first notice that there are 2 types of degeneracy. We can say that the solution of the clustering problem is *degenerate Type-1*, if there are one or more cluster centres which have no entities allocated to them. We can say that the solution of the clustering problem is *degenerate Type-2*, if there exist at least two cluster centres which are identical. Following these definitions, it is clear that a degenerate solution of Type-2 is also a degenerate of Type-1, but the converse does not hold. I also define the degree of degeneracy [17]: We can say that a degenerate solution has a degree of degeneracy equal to d if the number of empty clusters in the solution is equal to d.

Type-1 degeneracy of KHM. I first illustrate *degenerate Type-1* on the following well-known Ruspini data set [74] (entities are 75 points in the plane, as given in Figure 4.1). In this experiment, I attempt to start with bad initialization clustering to check for degeneracy. I show that a degenerate solution of Type-1 occurs even in the first iteration of the KHM algorithm if $K = 4$. In fact, if the initial cluster centres are located at customer locations 75, 63, 65 and 61 (see Figure 4.2a), then after the allocation step, the objective function of such a proper solution is 669408.938. Entities are divided into 4 clusters, as follows: 63 entities $\{1, 2, \ldots, 59, 71, 72, 74, 75\}$ are closest to entity 75; 4 entities $\{60, 63, 66, 73\}$ are closest to entity 63; 7 entities $\{62, 64, 65, 67, 68, 69, 70\}$ are allocated to cluster centre 65. The last entity is 61, which contains itself. The next step shows the degeneracy in cluster 4 (see Figure 4.2b). It is interesting to note that Type-1 degeneracy could appear in the KHM algorithm and then be fixed by itself, i.e., without applying additional rules. After only one step of location and allocation, it appears that the degeneracy is removed by itself. However, the objective function is almost 3 times smaller: 252499.813.

Degeneracy of degree 2 also exists if $K = 5$. It is shown in Figure (4.2d) where I suggest the same initial solution as in Figure 4.2a but entity 62 is added as the centroid for the fifth cluster. Thus, the degenerate solution is already obtained in the first iteration (see Figure (4.2e). However, it is removed from the solution as before, in the next step of KHM. Therefore, type 1 degeneracy may be automatically corrected during the execution of `KHM`. But for many other datasets, the degenerate solutions are significantly affected at the end of the local search, as explained in section (4.2.2).

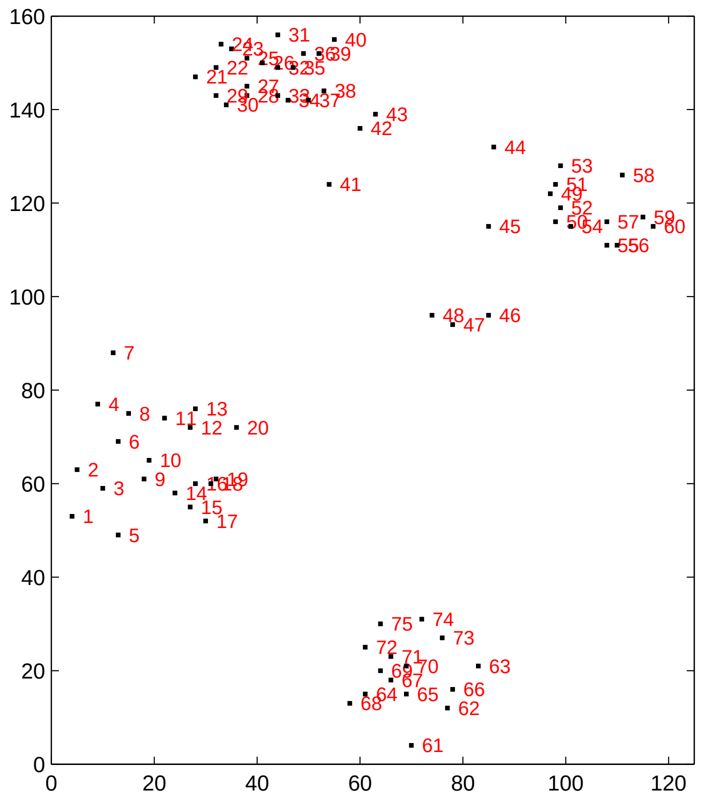
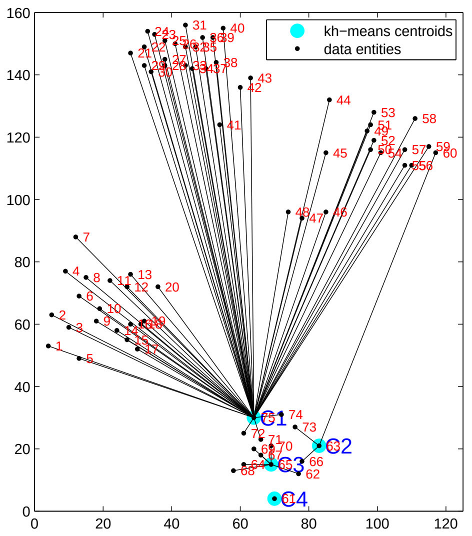
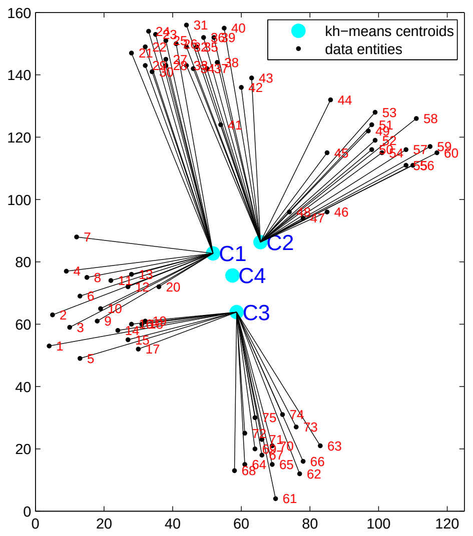


Figure 4.1: Ruspini dataset.

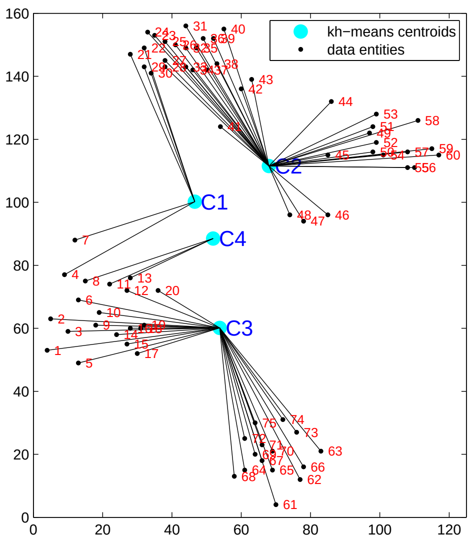
Although the `KHM` heuristic improves Type-1 degeneracy automatically, I show in section 4.4 that the quality of the final solution exceeds this if the degeneracy is removed immediately as it appears. Moreover, the number of iterations of the original `KHM` is greater (see Table 4.2).



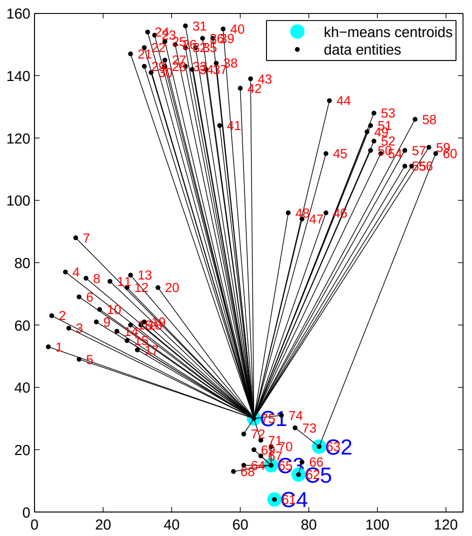
(a) Initial solution and $K = 4$.



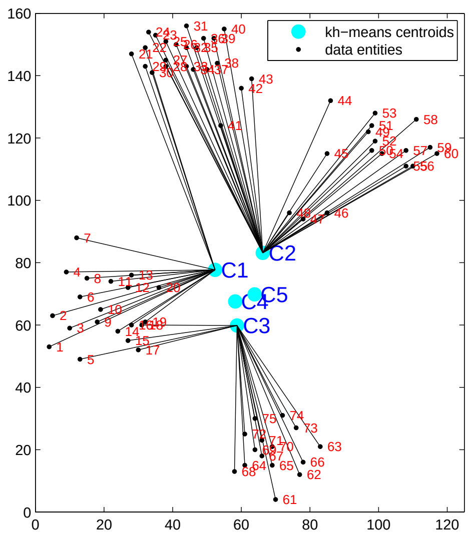
(b) 1st iteration ; $K = 4$; degeneracy degree=1.



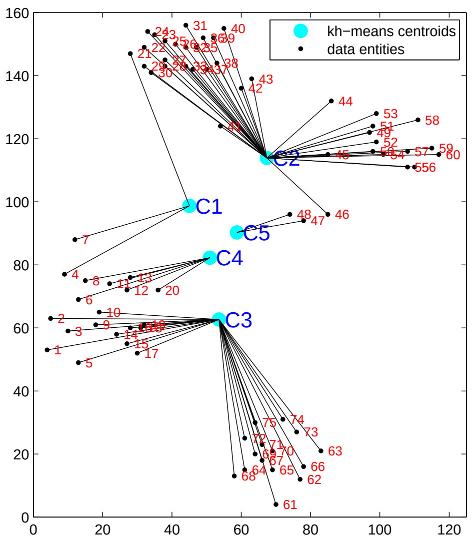
(c) 2nd iteration ; $K = 4$.



(d) Initial solution and $K = 5$.



(e) 1st iteration ; $K = 5$; degeneracy degree=2.



(f) 2nd iteration ; $K = 5$.

Figure 4.2: `KHM` clustering degeneracy for the *Ruspini* dataset.

Type-2 degeneracy of KHM. The following example proves the degeneracy of Type-2 for a KHM local search. The entities $x(i, \ell)$ and the initial solution $c(j, \ell)$ are as follows:

$$X = \begin{pmatrix} 0.5 & 0 \\ 1 & 1 \\ 1 & 5 \\ 1 & -5 \\ 1.5 & 0 \end{pmatrix}; \qquad C = \begin{pmatrix} 1 & 5 \\ 1 & 1 \\ 1.5 & 0 \\ 0.5 & 0 \end{pmatrix}.$$

The initial solution of this step is shown in Figure 4.3a. The next step is to calculate the objective function, as in (4.2), to get new centroids as in (4.6) and to calculate the membership and weight matrices as in (4.4) and (4.5). I choose $p = 2$ and $\varepsilon = 0.01$. The objective function is:

$$\begin{aligned} f_{KHM}(P) = \sum_{i=1}^{N} HA_i(K,P) &= \sum_{i=1}^{N} \frac{4}{\|x_i - c_1\|^{-2} + \|x_i - c_2\|^{-2} + \|x_i - c_3\|^{-2} + \|x_i - c_4\|^{-2}} \\ &= \frac{4}{10001.84} + \frac{4}{10001.66} + \frac{4}{10000.14} + \frac{4}{0.12} + \frac{4}{10001.84} \\ &= 34.1938 \end{aligned}$$

By simple calculations, we get the membership matrix:

$$m_{KHM}(c_j/x_i) = \begin{pmatrix} 0.0000 & 0.0000 & 1.0000 & 0.0249 & 0.0000 \\ 0.0000 & 1.0000 & 0.0000 & 0.1925 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.3913 & 1.0000 \\ 1.0000 & 0.0000 & 0.0000 & 0.3913 & 0.0000 \end{pmatrix}.$$

The fact that $m_{34} = m_{44} = 0.3913$ will cause future degeneracy. From this matrix, in fact, we can obtain crisp clustering matrix, as follows:

$$M_1 = max_j(m_{KHM}(c_j/x_i)) = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}; \qquad \text{or} \qquad M_2 = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

By repeating the same steps in the next iteration we get:

$$w_{KHM}(x_i) = \begin{pmatrix} 0.9996 & 0.9997 & 1.0000 & 0.2929 & 0.9996 \end{pmatrix}^T.$$

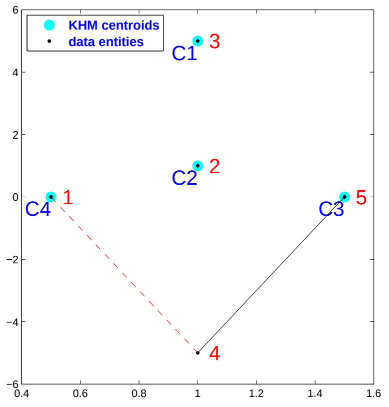
Now the new centroids, calculated from (4.6), are

$$C = \begin{pmatrix} 1.0000 & 4.9275 \\ 1.0000 & 0.6797 \\ 1.4486 & -0.5143 \\ 0.5514 & -0.5143 \end{pmatrix}.$$

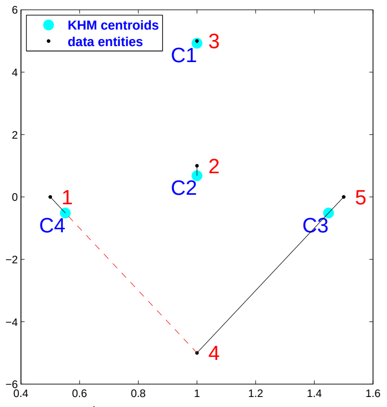
The results are shown in Figure (4.3b). We see in Figures (4.3b and 4.3c) that all five entities are clustered in 4 groups, as desired. But in in Figure (4.3d), two centroids are almost joined in one cluster. In the rest of the figures (4.3e, 4.3f), we can see clearly how they become the same. This step implies that the degeneracy in this example is considered to be of type 2. The final solution is:

$$C(final) = \begin{pmatrix} 1.0000 & 4.9953 \\ 1.0000 & 0.3291 \\ 1.0000 & -4.9891 \\ 1.0000 & -4.9891 \end{pmatrix}.$$

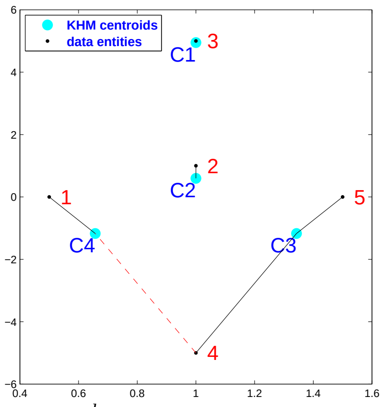
Note that entity 4 belongs to clusters 3 and 4 equally in the initial solution, as well as in all 5 iterations. At the end, cluster centroids 3 and 4 become identical, producing a degeneracy Type-2 solution.



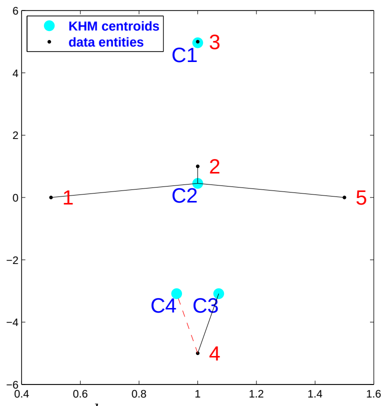
(a) Initial solution; objective function $f = 34.1938$.



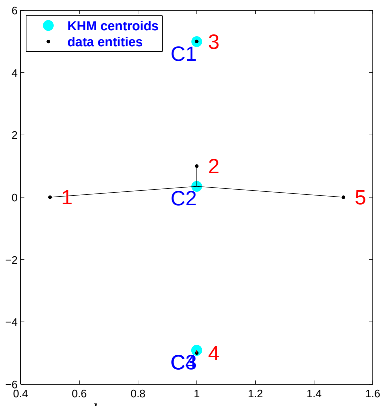
(b) 1st iteration; $f = 30.3828$.



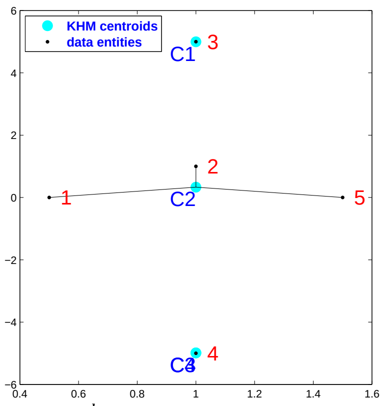
(c) 2nd iteration; $f = 25.9296$.



(d) 3rd iteration; $f = 11.2037$.



(e) 4th iteration; $f = 4.4748$.



(f) 5th iteration; $f = 4.4587$.

Figure 4.3: `KHM` clustering degeneracy for dataset-2.

4.2.2 Removing degeneracy (`KHM+`)

There are many efficient ways to remove degeneracy from the solution. Such procedures are found for example, in Cooper's ALT type algorithm for solving the Multi-Source Weber problem in [59, 17]. In certain datasets (for example, in the *B-Cancer data set 699*), the degeneracy remains in all iterations of `KHM`, i.e., it does not automatically vanish as in the example in Figure 4.2. This guides me to design an algorithm for removing degeneracy immediately as it appears, in order to avoid it in the next step. My pseudo-code is given in Algorithm 4.2. If a degeneracy of degree d occurs, my algorithm randomly selects d new centroids among existing entities. Such new solution is obviously not degenerate, since all K centroids have at least one entity which is allocated to them.

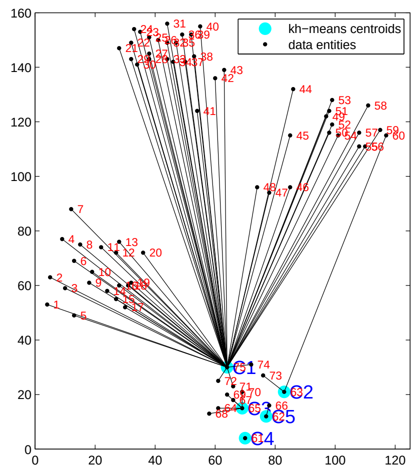
In other words, the coordinates of any centroid c_j without an entity are replaced by entity x_i taken at random. I also tested some different strategies for choosing the entity to be taken as a new centroid. However, it appears that the most efficient is random selection although the solution qualities are not significantly different. I found that the computing time for any deterministic search is long and does not usually improve the quality of the final solution.

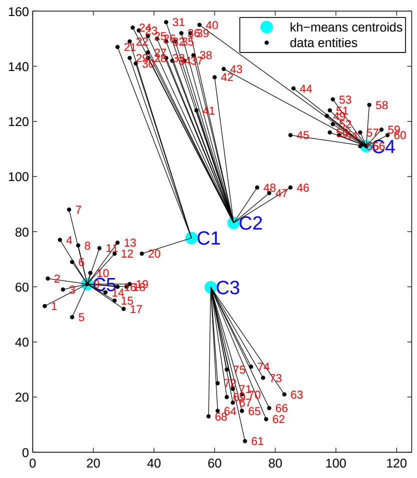
Algorithm 4.2: KHM+ local search with removing degeneracy

```
Function KHM+ ($X, K, Maxit, \varepsilon, N, C, z$)
1  $C^{(new)} = \{c_1, c_2, \ldots, c_K\}$   // $K$ centroids are chosen from $X$ at random
2  $i \leftarrow 0$                          // i-iteration counter
3  repeat
4      $i \leftarrow i + 1$; $C \leftarrow C^{(new)}$
5      $z \leftarrow f_{KHM}(C)$ as in (4.2)
6      Calculate $m$ as in (4.4) and $w$ as in (4.5) for all entities
7      Find new centroids $c_j^{(new)}$, $j = 1, \ldots, K$ as in (4.6)
8      Indicate indices $b_\ell$ of degenerate solutions ($\ell = 1, \ldots, g$)
9      if ($g > 0$) then
10         for $\ell := 1, \ldots, g$ do
11             $t \leftarrow b_\ell$
12             $h = 1 + n * RND$   // choose an entity $h$ at random
13             for $\beta := 1, \ldots, q$ do
                   $c_{t\beta} \leftarrow x_{h\beta}$
   until ($||c_j^{(new)} - c_j|| \leq \varepsilon$, $\forall j = 1, ..., K$ or $i = Maxit$)
```

By applying Algorithm 4.2, I can simply remove the degeneracy in the previous counter example for $k = 5$ in section (4.3). Figure 4.4 shows the solutions obtained by my KHM+. Although the solutions obtained by KHM and KHM+ are both proper after the second iteration, it appears that the objective function value of the former is more than twice as large (compare $f_{KHM} = 212,390$ with $f_{KHM+} = 97,519$).

In other experiments, the degeneracy may appear again in other iterations. So, the way to insert a random solution instead of degenerate ones once it appears reduces the time of justifying the initial solution.


(a) Initial solution; $f = 685329.3$.


(b) 1st iteration; $f = 440700.4$.

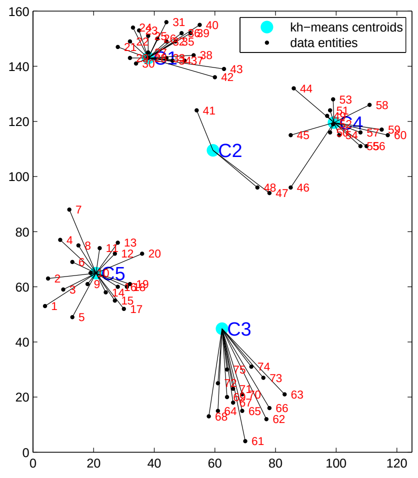

(c) 2nd iteration; $f = 97519.8$.

Figure 4.4: KHM clustering for the *Ruspini* dataset after removing degeneracy.

To make a precise comparison between KHM and KHM+, I use the same initial solutions for both algorithms. Table 4.2 contains a comparison of the two local searches on the *Ruspini* data and different values of cluster numbers (m). In column 4 of Table 4.2, I give the % difference between 2 algorithms calculated as:

$$\frac{f_{KHM} - f_{KHM+}}{f_{KHM+}}.100 \tag{4.8}$$

In column 5, I report number of iterations used and in column 6 the type of degeneracy that occurred.

4.3 VNS for KHM

Variable neighbourhood search (VNS) is a metaheuristic for solving combinatorial and global optimization problems whose basic idea is a systematic change of neighbourhood both within a descent phase to find a local optimum and in a perturbation phase to get out of the corresponding valley. The efficiency of VNS is based on three simple facts: (i) A local minimum

K	mth	obj	dev %	maxit	type	maxdeg	time
4	KHM	42980.7852	0.00	10	1	1	0.062
4	KHM+	42980.7812		9			0.000
5	KHM	41442.8750	0.00	23	1	2	0.016
5	KHM+	41442.8711		13			0.016
6	KHM	38989.2109	0.00	45	1	2	0.016
6	KHM+	38989.2109		28			0.016
7	KHM	40957.8125	2.58	59	1	2	0.016
7	KHM+	39928.8477		60			0.031
8	KHM	35056.9453	1.43	42	1	3	0.016
8	KHM+	34562.2109		40			0.016
9	KHM	32716.4531	0.00	34	1	4	0.031
9	KHM+	32716.4512		41			0.031
10	KHM	32406.1074	10.13	42	2	6	0.047
10	KHM+	29426.3652		34			0.031
11	KHM	30778.1641	4.23	41	2	7	0.016
11	KHM+	29527.8652		65			0.047
12	KHM	30869.2480	0.39	41	2	8	0.016
12	KHM+	30748.0254		80			0.047
13	KHM	31482.8633	7.97	41	2	7	0.047
13	KHM+	29160.1875		107			0.062
14	KHM	36413.9570	3.86	59	2	8	0.078
14	KHM+	35059.1758		30			0.031
15	KHM	37569.1562	3.50	54	2	9	0.078
15	KHM+	36298.4766		41			0.031

Table 4.2: *Comparison between methods* `KHM` *and* `KHM+` *on the Ruspini dataset*

with respect to (w.r.t.) in one neighborhood structure is not necessarily the same for another; (ii) A global minimum is a local minimum w.r.t. all possible neighborhood structures; (iii) For many problems, the local minima w.r.t. one or several neighbourhoods are relatively close to each other. The VNS metaheuristic is well-established in the literature. For an overview of the method and numerous applications, the reader is referred to [60], [41], and for the most recent survey, to [42].

For solving KHMCP, the VNS based heuristic (`VNS-KHM`) has already been proposed in [3]. For the sake of completeness, I repeat its steps in Algorithm 4.3.

In my `VNS-KHM+` the initial solution is obtained by selecting K centroids among the existing entities at random. The method terminates when a given running time t_{max} is reached. The inner loop iterates until there is no better solution in the last neighborhood (k_{max}) of the incumbent solution C. The inner loop consists of 3 steps: Shaking; Local search and Neighbourhood change. The only difference between my new VNS based heuristic suggested here

Algorithm 4.3: Steps of the basic VNS+

```
Function VNS+($X, K, k_{max}, t_{max}, C$)
1  repeat
2      $k \leftarrow 1$                                  // the neighbourhood index
3      repeat
4          $C' \leftarrow$ Shake($X, k, C$)                   // Shaking
5          $C'' \leftarrow$ KHM+ ($X, K, C', Maxit, \varepsilon$)     // Local search
6          NeighbourhoodsChange($C, C'', k$)  // Change centroid
       until $k = k_{max}$
7      $t \leftarrow$ CpuTime()
   until $t > t_{max}$
```

(VNS-KHM+) and the VNS-KHM as in [3] is that KHM+ local search, given in Algorithm 4.2, is used in the new method instead of the KHM used in the old VNS. Details regarding the functions of Shake and NeighbourhoodChange may be found in [3]. For the sake of completeness, here I give only their pseudo-codes. The main purpose of the Shaking step is

Algorithm 4.4: Shaking step

```
Function Shaking ($X, k, C$)
1  $j \leftarrow 0$                          // initializing iteration counter
2  repeat
3      $j \leftarrow j + 1$
4      $r1 \leftarrow \lfloor (m-j+1)*r \rfloor$       // a cluster is chosen at random
5      $r2 \leftarrow \lfloor (n-j+1)*r \rfloor$       // an entity is chosen at random
6      for $i := 1, \dots, q$ do
7          $c(r1, i) \leftarrow x(r2, i)$
   until $j = k$
```

to diversify the incumbent solution C. Neighbourhood k ($k = 1, \dots, k_{max}$) consists of random centroid-to-entity swaps. Such a random solution is the initial one for the KHM+ local search.

Algorithm 4.5: Neighbourhood change or move or not function

```
Function NeighbourhoodChange ($C, C', k$)
1  if $z(C') < z(C)$ then
2      $C \leftarrow C'; k \leftarrow 1$                    // make a move
   else
3      $k \leftarrow k + 1$                          // next centroid
```

Function `NeighborhoodChange()` compares the new value $z(C')$ with the incumbent value $z(C)$ obtained in the neighbourhood k (line 1). If an improvement is obtained, k is returned to its initial value and the new incumbent is updated (line 2). Otherwise, the next neighbourhood is considered (line 3).

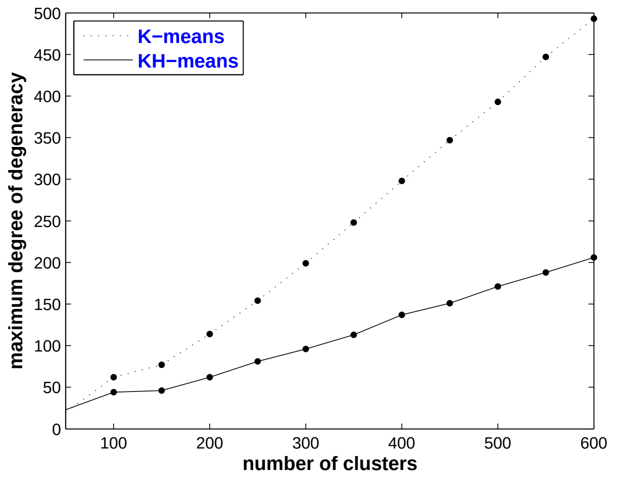
4.4 Computational Results

Computer. All experiments were performed on a personal computer Intel(R) Core(TM)2 with 0.98GB of RAM and a speed of 2.40GHz. All my methods were coded on Lahey/Fujitsu FORTRAN 95. For plotting, I use MATLAB 7.6.

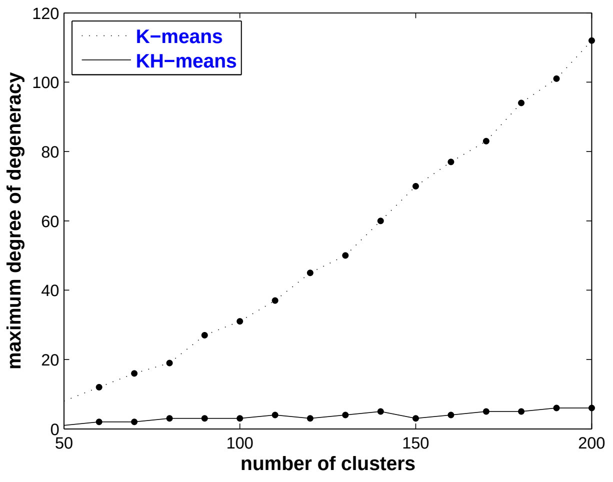
Test instances. I choose the following test instances: (i) *Ruspini* which has 75 entities in 2-dimensions [74]; (ii) *Iris* which has 150 entities in 4-dimensions; (iii) *Wine* which has 178 entities in 13-dimensions; (iv) *Glass* which has 214 entities in 9-dimensional space; (v) *Breast-cancer* which has 699 entities in 10-dimensions, and (vi) *Image Segmentation* with 2310 entities in 19-dimensions. For more details about them, see [12].

Parameters. I choose $\varepsilon = 0.01$ in all my algorithms. In Algorithm 4.1, the $Maxit = 180$. For all datasets, I put the power of `KHM` objective function as $p = 2$ for each number of clusters K.

Maximum degree of degeneracy. As mentioned above, the `KHM` algorithm has a smaller degree of degeneracy than `KM` (for solving Minimum sum-of-squares clustering). In Figure 4.5, I show the maximum degrees of degeneracy obtained during the execution of these two heuristics. Comparative results on two well-known datasets from the literature are presented: (i) *Breast-cancer* and (ii) *Image Segmentation*. The gap between the two algorithms is very clear: the maximum degree of degeneracy is much larger for `KM` than for `KHM`. However, it is interesting to note that the maximum degree of degeneracy is the empirically linear function of the cluster number K. Tables of these results and more details about KHM degeneracy for these datasets are explained in Appendices B.1 and B.2.



(a) Dataset:*Breast Cancer-699*



(b) Dataset:*Image Segmentation-2310*

Figure 4.5: Comparison between the degeneracy degrees of *K*-Means and KHM local searches after 100 starts.

Comparison between KHM and KHM+. In the following tables I present a comparison between the objective function values obtained with KHM and KHM+. The first column indicates the number of desired clusters *(K)*. The second column indicates the method used *(mth)*: in the original KHM and my KHM+. Column 3 *(obj)* gives the corresponding objective function values. Column 4 shows the percentage improvement *(dev %)* obtained by KHM+. The number of local search iterations *(maxit)* is displayed in Columns 5. The Type *(type)* and max degree *(maxdeg)* of degeneracy are displayed in columns 6 and 7 respectively. The last column shows the computing time *(time)* (in seconds) for each method.

K	method	obj	% dev	maxit	type	maxdeg	time
50	KHM	1010980.81	0.16	153	1	1	1.25
50	KHM+	1009370.12		138			0.84
60	KHM	772767.75	50.28	119	1	1	0.93
60	KHM+	514221.22		93			0.73
70	KHM	1214848.75	57.34	93	1	1	0.86
70	KHM+	772127.44		73			0.78

(a) Dataset: *Wine-178*

K	method	obj	% dev	maxit	type	maxdeg	time
180	KHM	136.40	26.54	4	2	1	0.09
180	KHM+	107.79		4			0.21
190	KHM	89.61	55.90	4	2	1	0.12
190	KHM+	57.48		5			0.37
200	KHM	37.74	21.60	5	2	1	0.09
200	KHM+	31.04		5			0.60

(b) Dataset: *Glass-214*

K	method	obj	% dev	maxit	type	maxdeg	time
100	KHM	29219.30	0.54	125	2	42	4.06
100	KHM+	29063.82		68			2.57
150	KHM	31468.25	9.24	48	2	39	2.35
150	KHM+	28806.04		7			0.57
200	KHM	30548.25	28.76	3	2	50	0.20
200	KHM+	23725.55		3			0.62
250	KHM	26197.01	37.50	3	2	62	0.25
250	KHM+	19052.13		2			0.85
300	KHM	23265.06	57.82	2	2	78	0.20
300	KHM+	14741.80		2			1.09

(c) Dataset: *Breast Cancer-699*

K	method	obj	% dev	maxit	type	maxdeg	time
100	KHM	32480866	0.20	104	2	1	14.75
100	KHM+	32416846		131			16.67
200	KHM	31760192	0.02	157	2	1	38.18
200	KHM+	31754658		150			40.87
300	KHM	30295272	0.02	105	2	3	37.68
300	KHM+	30288258		91			38.03
400	KHM	29596908	0.97	105	2	6	49.70
400	KHM+	29313292		102			48.68
500	KHM	28287296	0.42	50	2	10	29.28
500	KHM+	28169312		43			26.18

(d) Dataset: *Image Segmentation-2310*

Table 4.3: *Comparison between* KHM *and* KHM+ *based on one run.*

Based on the comparative results between KHM and KHM+ given in Table 4.3, the following observations can be obtained:

(i) By using KHM+, the solution qualities are improved up to 58%, in a lower number of iterations and smaller computing times, on average.

(ii) The degeneracy type is correlated with the instance. In other words, there is no instance with both types of degeneracy: *Wine-178* exhibits only type 1 and the other instances only type 2 degeneracy.

(iii) The number of clusters without entity (the maximum degree of degeneracy) could be more than 40% of the total number of clusters *K* (see *the Breast Cancer-699* instance).

Comparison between VNS-KHM and VNS-KHM+. The next table presents the influence of KHM+ when applied within 2 metaheuristics: Multi-start local search (MLS) and VNS. Heuristics which use KHM+ as a local search within MLS and VNS I denote as MLS+ and

VNS+ respectively. Table 4.4 presents the comparative results obtained by these 4 methods are presented. For each dataset I first run KHM and KHM+ 10 times to get the maximum time allowed for VNS and VNS+ (t_{max}). Those values are given in the last column of Table 4.4.

dataset	K	obj		% dev	obj		% dev	tmls	tvns	tmax
Wine (178)	50	MLS	619330.5630	8.75	VNS	466665.6250	0.78	0.02	0.04	0.17
		MLS+	569476.3750		VNS+	463067.2190		0.02	0.15	0.16
	60	MLS	571392.2500	11.12	VNS	435012.0310	9.16	0.00	0.13	0.13
		MLS+	514221.2190		VNS+	398527.0000		0.00	0.03	0.47
	70	MLS	463024.7500	3.28	VNS	353198.4380	3.81	0.02	0.17	0.29
		MLS+	448310.0940		VNS+	340229.7810		0.02	0.21	0.31
Glass (214)	180	MLS	106.0072	30.03	VNS	28.8215	4.24	0.05	0.14	0.17
		MLS+	81.5225		VNS+	27.6500		0.02	0.05	0.12
	190	MLS	48.2871	3.97	VNS	19.9370	1.29	0.09	0.14	0.14
		MLS+	46.4445		VNS+	19.6822		0.46	0.45	0.56
	200	MLS	29.4370	3.11	VNS	8.5587	3.98	0.08	0.71	0.75
		MLS+	28.5487		VNS+	8.2308		0.12	1.16	1.16
Breast Cancer (699)	100	MLS	28901.5645	0.67	VNS	27519.3906	0.26	0.08	0.23	0.23
		MLS+	28708.2246		VNS+	27449.3926		0.19	0.21	0.21
	150	MLS	27848.8496	2.51	VNS	24057.1543	0.15	0.09	0.40	0.40
		MLS+	27168.3398		VNS+	24021.5273		0.22	0.31	0.34
	200	MLS	27974.0879	22.40	VNS	20640.6973	10.33	0.43	0.40	0.48
		MLS+	22854.9551		VNS+	18707.5488		0.50	0.50	0.50
	250	MLS	26197.0117	44.04	VNS	18345.2031	21.15	0.06	0.55	0.65
		MLS+	18187.2559		VNS+	15142.7334		0.62	0.62	0.62
	300	MLS	23265.0645	71.16	VNS	16581.6523	47.98	0.08	0.67	0.70
		MLS+	13592.4814		VNS+	11205.0742		0.70	0.54	0.70
Image Segmentation (2310)	100	MLS	32480866	7.21	VNS	24805398	0.14	0.09	0.26	0.98
		MLS+	30295774		VNS+	24770124		0.76	0.83	0.95
	150	MLS	25970568	0.04	VNS	21838422	1.36	1.92	1.55	2.14
		MLS+	25960054		VNS+	21545372		0.25	0.41	2.06
	200	MLS	24675180	3.89	VNS	18951112	0.61	2.18	2.78	3.64
		MLS+	23750836		VNS+	18836514		1.25	1.01	4.25
	250	MLS	24025238	1.05	VNS	17222892	0.06	2.17	2.75	4.42
		MLS+	23776396		VNS+	17212400		2.12	2.54	4.43
	300	MLS	21686348	3.82	VNS	16094108	0.57	2.17	2.67	5.34
		MLS+	20888608		VNS+	16002548		1.03	1.98	5.18

Table 4.4: *Comparison between* `KHM-VNS` *and* `KHM-VNS+`.

It appears that:

(i) Clearly the best results for each instance tested are obtained by `VNS+` heuristic. Moreover, those results are obtained in less CPU time than results obtained by `VNS`.

(ii) VNS is always better than `MLS+`, exept for the two *Breast cancer* instances (for $k = 250$ and $K = 300$).

(iii) MLS+ improves the solution quality of MLS significantly. Thus, four methods can easily be ranked as follows: `VNS+, VNS, MLS+, MLS`.

(iv) I also observed that the better results obtained by VNS+ are reported even when the final solution obtained by `VNS` is not degenerate. This means that removing degeneracy immediately when it appears during the KHM iteration is better idea than to wait possible correction in future iterations.

(v) Regarding CPU time, `MLS+` and `VNS+` are slightly faster on average than `MLS` and `VNS` respectively.

4.5 Conclusion

In this chapter I consider the K-Harmonic Means clustering problem (KHMCP) and alternate type of heuristic (ALT) to solve it. I show that the K-Harmonic Means (KHM) clustering heuristic for solving KHMCP poses the property of degeneracy, i.e., the property that some clusters could remain empty (without entities) during the execution or at the code. I distinguish two types of degenerate solutions and provide an efficient procedure which removes degeneracy immediately when it appears in iterations. Moreover, this new routine is used as a local search within a recent variable neighbourhood search (VNS-KHM) which represents the current state-of-the-art heuristics for solving KHMCP. The extensive computational analysis on the usual data sets from the literature confirms that degeneracy could seriously damage the solution qualities of both `KHM` and `VNS-KHM`.

Chapter 5

Conclusion

It has been seen that K-Harmonic Means (KHM) clustering algorithm plays a very good role in clustering and heuristic applications. KHM has a soft membership function that measures the probability of every entity in the dataset being allocated to a cluster. Also, the weight function increases the weight the entities which are furthest away from each cluster. In comparison with K-Means (KM), it was also shown in details how KHM is not sensitive with initialization.

KHM is applied by using Variable Neighbourhood Search (VNS). The code is tested with known datasets and compared with some recent methods such as Tabu Search and Simulated Annealing. It is proved that VNS-KHM surpasses other methods. Some experiments give some good observations, like the changing values of objective functions based on the power of KHM. Also, the speed of getting the solution in VNS-KHM is very significant.

Despite of these advantages, KHM suffers from degeneracy as it is proven by counter examples. But it has less degree of degeneracy than KM. It is shown that removing the degeneracy immediately aids the solution and reduces the abundant degenerate iterations. The basic VNS-KHM and Multi-start algorithms are produced after removing the degeneracy and this leads to very good improvements.

In the mean time, the KHM code which is displayed in Appendix A might be adopted by a very big company in Hong Kong. They have a depot of containers in the port. They use the Radio Frequency Identification (RFID) technique. These containers should be arranged in levels up to 7. The problem is to arrange these containers to be clustered correctly to reduce the cost of the vehicle inside the depot which carry the required containers within unwanted ones.

One of the competitive projects is that intrusion detecting problem. The KDD cup 1999 [1] is the dataset used for this contest. These are a data set of more than 4 million entities with 41 dimensions collected from military network environment. The task is to classify the bad instances from the labeled data file. This dataset became very popular for testing the strength of clustering methods.

Future research may include: (i) the development of a general statement regarding degeneracy in alternate iterative procedures; (ii) the design of different methods for correcting degenerate solutions for ALT methods; (iii)an investigation of the relation between initial solution methods of K-Means and KHM with degeneracy, i.e., whether the proper initialization method could avoid degeneracy altogether?

Bibliography

[1] *Kdd cup 1999*, http://kdd.ics.uci.edu/databases/kddcup99.html, [Online; accessed March 3, 2009]. 74

[2] M.S. Aldenderfer and R.K. Blashfield, *Cluster Analysis*, vol. 7, Quantitative Applications in the Social Sciences, no. 44, CA: Sage Publications, California, 1984. 2

[3] A. Alguwaizani, P. Hansen, N. Mladenović, and E. Ngai, *Variable neighborhood search for harmonic means clustering*, Applied Mathematical Modelling (2010). 54, 66, 67

[4] D. Aloise, A. Deshpande, P. Hansen, and P. Popat, *NP-hardness of Euclidean sum-of-squares clustering*, Machine Learning **75** (2009), no. 2, 245–248. 37, 53

[5] D. Aloise and P. Hansen, *Clustering, in: D.r sheir (ed.)*, Handbook of Discrete and Combinatorial Mathemaics, CRC Press, 2009. 3, 9, 36, 37, 53

[6] D. Aloise, P. Hansen, and L. Liberti, *An improved column generation algorithm for minimum sum-of-squares clustering*, Mathematical Programming **123** (2010), no. 2. 3, 37

[7] F. Arito and G. Leguizamón, *Incorporating tabu search principles into aco algorithms*, Hybrid Metaheuristics **5818** (2009), 130–140. 24

[8] G. H. Ball, *Classification Analysis*, Technical report, Stanford Research Institute, California, November 1971. 2

[9] R.J. Barlow, *Statistics: a guide to the use of statistical methods in the physical sciences*, John Wiley & Sons Inc, 1989. 6

[10] N. Belacel, P. Hansen, and N. Mladenović, *Fuzzy J-means: a new heuristic for fuzzy clustering*, Pattern Recognition **35** (2002), no. 10, 2193–2200. 44, 46

[11] J. Ben Atkinson, *A greedy look-ahead heuristic for combinatorial optimization: an application to vehicle scheduling with time windows*, Journal of the Operational Research Society **45** (1994), no. 6, 673–684. 16

[12] C.L. Blake and C.J. Merz, *UCI repository of machine learning databases*,

http://archive.ics.uci.edu/ml/datasets.html, 1998, [Online; accessed July 19, 2008]. 47, 50, 68

[13] C. Blum and A. Roli, *Metaheuristics in combinatorial optimization: Overview and conceptual comparison*, ACM Computing Surveys (CSUR) **35** (2003), no. 3, 268–308. 23

[14] S.P. Borgatti, *How to explain hierarchical clustering*, Connections **17** (1994), no. 2, 78–80. 8

[15] Everitt Brian, *Cluster Analysis*, 1 ed., Heinemann Educational Books, London, 1974. 2, 3

[16] J. Brimberg, P. Hansen, and N. Mladenović, *Attraction probabilities in variable neighborhood search*, 4OR: A Quarterly Journal of Operations Research **8** (2010), no. 2, 181–194. 43

[17] J. Brimberg and N. Mladenović, *Degeneracy in the multi-source Weber problem*, Mathematical Programming **85** (1999), no. 1, 213–220. 53, 58, 63

[18] E.K. Burke and G. Kendall, *Search methodologies: introductory tutorials in optimization and decision support techniques*, Springer Verlag, 2005. 15

[19] V. Černỳ, *Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm*, Journal of optimization theory and applications **45** (1985), no. 1, 41–51. 19

[20] Y. Chou, *Statistical analysis*, Holt, Rinehart and Winston, 1970. 39, 81

[21] R.M. Cormack, *A review of classification*, Journal of the Royal Statistical Society. Series A (General) **134** (1971), no. 3, 321–367. 6

[22] M. Dorigo, V. Maniezzo, and A. Colorni, *Ant system: optimization by a colony of cooperating agents*, IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics **26** (1996), no. 1, 29–41. 18

[23] M. Dorigo and T. Stützle, *Ant colony optimization*, MIT Press, Cambridge, 2004. 18

[24] A. Eiben, P. Raué, and Z. Ruttkay, *Genetic algorithms with multi-parent recombination*, Parallel Problem Solving from NaturePPSN III (1994), 78–87. 27

[25] E.W. Forgy, *Cluster analysis of multivariate data: efficiency versus interpretability of classifications*, Biometrics **21** (1965), 768–769. 39

[26] M.R. Garey, D.S. Johnson, and R. Sethi, *The complexity of flowshop and jobshop scheduling*, Mathematics of Operations Research (1976), 117–129. 14

[27] F. Glover, *Future paths for integer programming and links to artificial intelligence*, Computers & Operations Research **13** (1986), no. 5, 533–549. 17

[28] ______, *Tabu search and adaptive memory programming: Advances, applications and challenges*, Interfaces in Computer Science and Operations Research **1** (1996). 21

[29] F. Glover, J.P. Kelly, and M. Laguna, *Genetic algorithms and tabu search: hybrids for optimization*, Computers & Operations Research **22** (1995), no. 1, 111–134. 24

[30] F. Glover and G. Kochenberger, *Handbook of Metaheuristics, volume 57 of International Series in Operations Research & Management Science*, 2003. 24

[31] F. Glover and M. Laguna, *Tabu search.*, 1997. 23

[32] D.E. Goldberg, K. Deb, and B. Korb, *Don't worry, be messy*, Proceedings of the Fourth International Conference on Genetic Algorithms, vol. 51, Morgan Kaufmann Publishers, 1991, p. 24. 24

[33] B. Golden, L. Bodin, T. Doyle, and W. Stewart Jr, *Approximate traveling salesman algorithms*, Operations Research **28** (1980), no. 3, 694–711. 16

[34] AD Gordon, *A review of hierarchical classification*, Journal of the Royal Statistical Society. Series A (General) (1987), 119–137. 6

[35] Z. Güngör and A. Ünler, *K-harmonic means data clustering with simulated annealing heuristic*, Applied mathematics and computation **184** (2007), no. 2, 199–209. 38, 43, 49, 51

[36] Z. Güngör and A. Ünler, *K-Harmonic means data clustering with tabu-search method*, Applied Mathematical Modelling **32** (2008), no. 6, 1115–1125. 38, 42, 47, 49, 51

[37] G. Hamerly and C. Elkan, *Alternatives to the k-means algorithm that find better clusterings*, Proceedings of the eleventh international conference on Information and knowledge management, ACM, 2002, pp. 600–607. 40, 41, 55

[38] J.M. Hammersley and D.C. Handscomb, *Monte carlo methods*, Taylor & Francis, 1975. 19

[39] P. Hansen and B. Jaumard, *Cluster analysis and mathematical programming*, Mathematical programming **79** (1997), 191–215. 3, 37

[40] P. Hansen and N. Mladenović, *J-means: a new local search heuristic for minimum sum of squares clustering*, Pattern Recognition **34** (2001), no. 2, 405–413. 37, 44, 45, 46, 51, 53

[41] ______, *Variable neighborhood search: Principles and applications*, European journal of operational research **130** (2001), no. 3, 449–467. 43, 46, 66

[42] P. Hansen, N. Mladenović, and J. A. Moreno Pérez, *Variable neighbourhood search: methods and applications*, 4OR: A Quarterly Journal of Operations Research **6** (2008), no. 4, 319–360. 43, 46, 66

[43] D.S. Hochba, *Approximation algorithms for np-hard problems*, ACM SIGACT News **28** (1997), no. 2, 40–52. 14

[44] J.H. Holland, *Adaptation in natural and artificial systems*, Ann Arbor MI: University of Michigan Press (1975). 24

[45] ______, *Adaptation in natural and artificial systems*, MIT Press Cambridge, MA, USA, 1992. 24

[46] F. Höppner, F. Klawonn, R. Kruse, and T. Runkler, *Fuzzy cluster analysis: methods for classification, data analysis, and image recognition*, Wiley, 1999. 9

[47] A.K. Jain and R.C. Dubes, *Algorithms for clustering data*, Prentice-Hall, Inc., 1988. 10

[48] L. Kaufmann and P. Rousseeuw, *Finding groups in data: An introduction to cluster analysis*, John Wiley & Sons, New York, 1990. 3, 36

[49] J. Kennedy and R. Eberhart, *Particle swarm optimization*, Neural Networks, 1995. Proceedings., IEEE International Conference on, vol. 4, IEEE, 1995, pp. 1942–1948. 28

[50] S.S. Khan and A. Ahmad, *Cluster center initialization algorithm for K-means clustering*, Pattern Recognition Letters **25** (2004), no. 11, 1293–1302. 53

[51] S. Kirkpatrick, *Optimization by simulated annealing: Quantitative studies*, Journal of Statistical Physics **34** (1984), no. 5, 975–986. 19

[52] J.R. Koza, *Genetic programming*, (1992). 27

[53] J. Lazić, *New variable neighbourhood search based heuristics for 0-1 mixed integer programming and clustering*, Ph.D. thesis, Brunel University, 2010. 15, 17, 18

[54] Q. Li, N. Mitianoudis, and T. Stathaki, *Spatial kernel K-harmonic means clustering for multi-spectral image segmentation*, Image Processing, IET **1** (2007), no. 2, 156–167. 40, 56

[55] J. MacQueen, *Some methods for classification and analysis of multivariate observations*, Berkeley, CA: University of California Press **1** (1967), 281–297. 39

[56] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, E. Teller, et al., *Equation of state calculations by fast computing machines*, The journal of chemical physics **21** (1953), no. 6, 1087. 19

[57] B. Mirkin, *Clustering for data mining: a data recovery approach*, CRC Press, FL, 2005. 3, 36

[58] D.W. Mitchell, *More on spreads and non-arithmetic means*, The Mathematical

Gazette 88 (2004), 142–144. 82

[59] N. Mladenović and J. Brimberg, *A degeneracy property in continuous location-allocation problems*, Les Cahiers du GERAD **G-96-37** (1996). 53, 63

[60] N. Mladenović and P. Hansen, *Variable neighborhood search*, Computers and Operations Research **24** (1997), no. 11, 1097–1100. 29, 43, 66

[61] D.G. Morrison, *Measurement problems in cluster analysis*, Management Science **13** (1967), no. 12, 775–780. 2

[62] P. Moscato, *On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms*, Caltech Concurrent Computation Program, C3P Report **826** (1989). 28

[63] H. Mühlenbein and H.M. Voigt, *Gene pool recombination in genetic algorithms*, Metaheuristics: Theory and applications (1996), 53–62. 27

[64] B. Jaumard O. du Merle, P. Hansen and N. Mladenović, *An interior point algorithm for minimum sum of squares clustering*, SIAM Journal on Scientific Computing **21** (1999), 1485 – 1505. 3, 37, 49

[65] I. Ono and S. Kobayashi, *A real-coded genetic algorithm for function optimization using unimodal normal distribution crossover*, Journal of Japanese Society for Artificial Intelligence **14** (1997), no. 6, 246–253. 27

[66] C.H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Inc. (1982). 14

[67] G. Patané and M. Russo, *The enhanced LBG algorithm*, Neural Networks **14** (2001), no. 9, 1219–1237. 42, 50

[68] J.M. Pena, J.A. Lozano, and P. Larranaga, *An empirical comparison of four initialization methods for the K-Means algorithm*, Pattern recognition letters **20** (1999), no. 10, 1027–1040. 53

[69] G. Pólya, *How to solve it: A new aspect of mathematical method*, Princeton University Press Princeton, NJ, 1945. 17

[70] V. Rayward, I. Osman, C. Reeves, and G. Smith, *Modern heuristic search methods*, John Wiley & Sons, Chichester, England, 1996. 15

[71] S.J. Redmond and C. Heneghan, *A method for initialising the K-means clustering algorithm using kd-trees*, Pattern Recognition Letters **28** (2007), no. 8, 965–973. 53

[72] C.R. Reeves and J.E. Rowe, *Genetic algorithms: principles and perspectives: a guide to GA theory*, Kluwer Academic Publishers, 2003. 18

[73] G. Reinelt, *Tsp-lib - a traveling salesman library*, ORSA Journal Computing **3** (1991), 376–384. 38, 47, 50

[74] E.H. Ruspini, *Numerical methods for fuzzy clustering*, Information Sciences **2** (1970), 319–350. 58, 68

[75] J. Scoltock, *A survey of the literature of cluster analysis*, The Computer Journal **25** (1982), no. 1, 130. 1

[76] T. Su and J. Dy, *A deterministic method for initializing K-means clustering*, International Conference on Tools with Artificial Intelligence, IEEE, 2005, pp. 784–786. 53

[77] Thomson, *ISI Web of Knowledge*, `http://pcs.isiknowledge.com/analyze/ra.cgi`, 2004, [Online; accessed January 12, 2011]. 1

[78] V.V. Vazirani, *Approximation Algorithms*, Springer, 2004. 15

[79] S. Vo, I.H. Osman, and C. Roucairol, *Meta-heuristics: Advances and trends in local search paradigms for optimization*, Kluwer Academic Publishers Norwell, MA, USA, 1999. 17

[80] J.H. Ward, *Hierarchical grouping to optimize an objective function*, Journal of the American statistical association **58** (1963), no. 301, 236–244. 6

[81] R. Xu and D. Wunsch, *Clustering*, IEEE Press, 2009. 2, 3, 9

[82] B. Zhang, *Generalized k-harmonic means - boosting in unsupervised learning*, Technical report, HPL-2000-137, Hewlett-Packard Laboratories, 2000. 11, 39, 40, 51, 55, 56

[83] B. Zhang, M. Hsu, and U. Dayal, *K-harmonic means - a data clustering algorithm*, Technical report, HPL-1999-124, Hewlett-Packard Laboratories, 1999. 11, 38, 39, 51, 53, 55, 56

Chapter 6

Harmonic Mean vs. Arithmetic Mean

Suppose we have the set $X = \{x_1, \ldots, x_n\}$ then the arithmetic Mean (AA) for X is:

$$AA = \frac{1}{n}\sum_{i=1}^{n} x_i \quad . \tag{6.1}$$

The harmonic average (HA) is always the least of three Pythagorean means (including Arithmetic average (AA) and Geometric average (GA)) for positive sets that contain non-equal values. However the AA is always the greatest. The HA tends (compared to the the AA) to reduce the impact of large outliers and enlarge the impact of small ones.

In some certain problems such as the speed average, the HA is the truest mean. It is very often that the AA is mistakenly used instead of the HA [20].

For example, suppose that a person drove an automobile on a highway. He passed five exits. Between each two exits he has been driving at average speed of 70 miles/hr, 75 miles/hr, 60 miles/hr and 65 miles/hr. The exits are equally distanced of 10 miles. What is the average speed between the first exit and the last exit?

So, the HA for this problem is:

$$HA = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}} = \frac{4}{\frac{1}{70} + \frac{1}{75} + \frac{1}{60} + \frac{1}{65}} = 67.03499079 \quad miles/hr.$$

However by using the AA formula above 6.1,

$$AA = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{1}{4}(70 + 75 + 60 + 65) = 67.5 \quad miles/hr.$$

But the total time for this journey is:

$$\frac{10}{70}+\frac{10}{75}+\frac{10}{60}+\frac{10}{65}=0.596703 \quad hr.$$

If the HA is considered for this journey, then the total time is

$$\frac{40}{67.03499079}=0.596703 \quad hr.$$

which is compatible with the right time. However if the AA is considered, then the total time for this journey is:

$$\frac{40}{67.5}=0.592593 \quad hr.$$

which is slightly different from the original time.

This example shows that the harmonic average is more accurate in many particular applications than the arithmetic average. For more details about other matters of means, the reader may refer to [58].

Appendix A

Fortran Code for KHM Local Search

```
      program KHarmonicMeans
*     ###########################################################
*     #         max number of entities = 3100                    #
*     #         maximum dimension of data = 50                   #
*     #         maximum number of clusters = 100                 #
*     #         eps - input precision (eps)                      #
*     #         p - power parameter for Harmonic means           #
*     #         maxit - maximum number of iterations             #
*     #         mk - membership matrix                           #
*     #         wi - weight function                             #
*     #         n - number of entities                           #
*     #         n1 - dimension of data                           #
*     #         m - number of clusters                           #
*     ###########################################################
*     ------------------------  Declarations
      real      x(3100,50),c(100,50),mk(3100,100),wi(3100),ck(100,50)
      logical*1 mm(3100,100)
      real*8    seed
*     ------------------------  Read the input data
*     ###########################################################
*     #        iun is the data file should be saved as fort.150   #
*     #        the first line must contain 6 parameters:          #
*     #        n,m,n1,eps,p,maxit  as shown above.                #
*     #        Here I present the IRIS data set as an example     #
*     ###########################################################
      seed=12
      iun=150
      read(iun,*)n,m,n1,eps,p,maxit
      do i=1,n
         read(iun,*)(x(i,j),j=1,n1)
      enddo
      call RndInit(m,n,n1,x,c)
*     ------------------------ Local search KHM
      zopt=10.e20;
      call khm(x,c,eps,p,maxit,n,m,n1,mk,mm,wi,ck,zopt)
*     ------------------------  Print results
*     ###########################################################
*     #        This gives the final result of KHM.                #
*     ###########################################################
      write(*,*)' Objective function value is ', zopt
      write(*,*)' Cluster centroid coordinates:'
      write(*,*)' ------------------------------'
      do j=1,m
```

```
         write(*,*)j,'.',(ck(j,i),i=1,n1)
      enddo
      stop
      end
*     ------------------------ Subroutine of KHM
      subroutine khm(x,c,eps,p,maxit,n,m,n1,mk,mm,wi,ck,zopt)
      real      x(3100,50),c(100,50),u(3100,100),s3(3100),s4(3100)
      real      mk(3100,100),ck(100,50),c1(3100,100),c2(3100,100)
      real      s5(3100),wi(3100)
      logical*1 mm(3100,100)
      integer   h,nbe(100),opt(3100)
*     ------------------------
      zopt=10.e21
      do h=1,maxit
      write(*,*)' Iteration: ',h
          z=0.
*         ------------------------  One iteration
          do i=1,n
             s3(i)=0.
             s4(i)=0.
             do j=1,m
                did=0.
                do k=1,n1
                   dd=(x(i,k)-c(j,k))*(x(i,k)-c(j,k))
                   did=did+dd
                enddo
                did=sqrt(did)
                if(did.eq.0.)did=eps
                u(i,j)=1/(did**(2+p))
                s3(i)=s3(i)+u(i,j)
                s4(i)=s4(i)+1/(did**p)
             enddo
             do j=1,m
                u(i,j)=u(i,j)/s3(i)
             enddo
             s5(i)=s4(i)*s4(i)
             wi(i)=s3(i)/s5(i)
             z=z+m/s4(i)
          enddo
*         ------------------------  Copy
          do i=1,n
             do j=1,m
                mk(i,j)=u(i,j)
             enddo
          enddo
*         ------------------------  Crisp clustering
          do i=1,n
             dmax=-1.e20
             do j=1,m
                mm(i,j)=.false.
                if(mk(i,j).gt.dmax)then
                   dmax=mk(i,j)
                   jst=j
                endif
             enddo
             mm(i,jst)=.true.
          enddo
          do k=1,n
          enddo
*         ------------------------  New centroids
          do j=1,m
             do i=1,n
                do k=1,n1
                   c1(i,k)= mk(i,j)*wi(i)*x(i,k)
                enddo
                c2(i,j)= mk(i,j)*wi(i)
             enddo
             sum2=0.
             do i=1,n
```

```
                 sum2=sum2+c2(i,j)
              enddo
              do k=1,n1
                 sum1=0.
                 do i=1,n
                    sum1=sum1+c1(i,k)
                 enddo
*                ck(j,k)=sum(c1(:,k))/sum(c2(:,j))
                 ck(j,k)=sum1/sum2
              enddo
*     ############################################################
*     #        To print the new centroids in each iteration.     #
*     ############################################################
              write(*,*)j,'.',(ck(j,i),i=1,n1)
           enddo
           dnorm=-1.e20
           do i=1,m
              dif=0.
              do j=1,n1
                 dif=dif+(c(i,j)-ck(i,j))*(c(i,j)-ck(i,j))
              enddo
              if(dif.gt.dnorm)dnorm=dif
           enddo
           dnorm=sqrt(dnorm)
*          write(*,*)'dnorm=',dnorm
           if(dnorm.lt.eps.or.Abs(z-z1).lt.eps/100.)then
              zopt=z
              return
           endif
*     ############################################################
*     #        To print the objective function in each iteration. #
*     ############################################################
           write(*,*)' Objective function value in iteration is ',z
           do i=1,m
              do j=1,n1
                 c(i,j)=ck(i,j)
              enddo
           enddo
           z1=z
      enddo
      zopt=z
      return
      end
      subroutine RndInit(m,n,n1,x,c)
*     ############################################################
*     #        Generation of initial centroids out of            #
*     #        existing entities.                                #
*     ############################################################
      real      x(3100,50),c(100,50)
      integer   p(3100)
*     ------------------------
      do j=1,n
         p(j)=j
      enddo
      do i=1,m
         call Exch(i,i7,n,p)
         do j=1,n1
            c(i,j)=x(i7,j)
         enddo
      enddo
      return
      end
*---------------------------------------------------
      subroutine Exch(i,i7,n,p)
      real*8     seed
      integer    p(3100)
*     -------------------------------------
      i2=n-i+1
      i1=1+i2*Ran(seed)
      i1=1+i2*seed
      i7=p(i1)
      p(i1)=p(i2)
      p(i2)=i7
      return
      end
*     ############################################################
*     #        Designed and created by:                          #
*     #          Abdulrahman Alguwaizani                         #
*     ############################################################
```

Appendix B

Details of KHM Degeneracy

The following tables may be interested for the reader, as they show the details of calculations of the charts which are presented in Chapter 4 (see Figure 4.5). In the next sections I show all results for the degeneracy of *Breast-cancer* Dataset and *Image Segmentation-2310* datasets after applying `KHM` 100 times.

B.1 Multi-Start of KHM for *Image Segmentation-2310* Dataset

In the following table, the first column denotes the number of clusters. The second column indicates the max degree of degeneracy for 100 multi-starts. However, the proper iterations are presented in column 3. The last column is designed to show number of degenerate iterations (from these 100) of each degree. For instance, when the dataset is clustered to 70, there are 87 proper iterations (out of 100) and 13 degenerate iterations, 12 of them are of degree 1, and 1 of degree 2.

K	max degree	proper iterations	degree					
			1	2	3	4	5	6
50	1	89	11					
60	2	88	11	1				
70	2	87	12	1				
80	3	74	17	8	1			
90	3	70	25	4	1			
100	3	60	31	7	2			
110	4	54	31	12	2	1		
120	3	53	30	14	3			
130	4	53	29	13	4	1		
140	5	37	40	14	6	2	1	
150	3	26	46	22	6			
160	4	30	39	24	5	2		
170	5	21	28	26	19	5	1	
180	5	18	30	25	13	10	4	
190	6	15	41	27	12	4	0	1
200	6	15	27	35	14	7	1	1

Table B.1: *Degeneracy degrees of KHM after 100 starts for dataset:* Image Segmentation-2310.

B.2 Multi-Start of KHM for *Breast-cancer* Dataset

In the following table, the first column denotes the number of clusters. The second and third columns indicate the max and min degree of degeneracy for 100 multi-starts. The last column shows number of degenerate iterations of each degree in details. For instance, when the dataset is clustered to 400, there is one iteration has 102 empty clusters.

K	**max degree**	**min degree**	**# of degenerate iterations of each degree up to max degree**															
100	47	15	degree	1-14	15	16-17	18	19	20	21	22	23	24	25	26	27	28	29
			# iter	0	1	0	1	2	0	3	1	2	2	5	5	9	4	6
			degree	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
			# iter	2	4	3	6	5	8	7	6	5	4	1	0	4	1	2
			degree	45	46	**47**												
			# iter	0	0	1												
200	59	35	degree	1-34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
			# iter	0	2	1	1	1	1	2	1	8	4	10	6	6	4	6
			degree	49	50	51	52	53	54	55	56	57	58	**59**				
			# iter	7	9	5	6	8	1	1	5	3	0	2				
300	94	69	degree	1-68	69	70	71	72	73	74	75	76	77	78	79	80	81	82
			# iter	0	1	1	0	2	2	4	4	7	5	9	7	8	4	5
			degree	83	84	85	86	87	88	89	90	91	92	93	**94**			
			# iter	7	5	5	5	4	4	0	2	3	1	4	1			
400	136	102	degree	1-101	102	103	104	105	106	107	108	109	110	111	112	113	114	115
			# iter	0	1	0	0	1	0	3	2	0	2	0	1	4	6	3
			degree	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130
			# iter	6	4	9	9	4	10	7	2	5	4	3	4	1	2	3
			degree	131	132	133	134	135	**136**									
			# iter	0	2	0	0	1	1									
500	172	145	degree	1-144	145	146	147	148	149	150	151	152	153	154	155	156	157	158
			# iter	0	1	0	2	2	2	9	5	6	4	4	5	1	13	5
			degree	159	160	161	162	163	164	165	166	167	168	169	170	171	**172**	
			# iter	6	6	4	5	6	6	3	0	1	1	0	1	1	1	
600	209	183	degree	1-182	183	184	185	186	187	188	189	190	191	192	193	194	195	196
			# iter	0	1	0	0	0	1	1	0	4	8	4	6	5	7	8
			degree	197	198	199	200	201	202	203	204	205	206	207	208	**209**		
			# iter	9	10	12	4	5	5	3	3	2	0	1	0	1		

Table B.2: *Degeneracy degrees of KHM after 100 starts for dataset:* Breast Cancer-699.