IMPLEMENTATION OF A SPECIALIZED ALGORITHM FOR CLUSTERING USING MINIMUM ENCLOSING BALLS

A THESIS
SUBMITTED TO THE DEPARTMENT OF INDUSTRIAL ENGINEERING
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
Utku Guruşçu
July, 2010

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Emre Alper Yıldırım(Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Barbaros Tansel

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Uğur Doğrusöz

Approved for the Institute of Engineering and Science:

Prof. Dr. Levent Onural
Director of the Institute

ABSTRACT

IMPLEMENTATION OF A SPECIALIZED ALGORITHM FOR CLUSTERING USING MINIMUM ENCLOSING BALLS

Utku Guruşçu
M.S. in Industrial Engineering
Supervisor: Assoc. Prof. Dr. Emre Alper Yıldırım
July, 2010

Clustering is the process of organizing objects into groups whose members are similar in some ways. The main objective is to identify the underlying structures and patterns among the objects correctly. Therefore, a cluster is a collection of objects which are more similar to each other than to the objects belonging to other clusters.

The clustering problem has applications in wide-ranging areas including facility location, classification of massive data, and marketing. Many of these applications call for the solutions of the large-scale clustering problems.

The main problem of focus in this thesis is the computation of k spheres that enclose a given set of m vectors, which represent the set of objects, in such a way that the radius of the largest sphere or the sum of the radii of spheres is as small as possible. The solutions of these problems allow one to divide the set of objects into k groups based on the level of similarity among them.

Both of the aforementioned mathematical problems belong to the hardest class of optimization problems (i.e., they are NP-hard). Furthermore, as indicated by previous results in the literature, it is not only hard to find an optimal solution to these problems but also to find a good approximation to each one of them.

In this thesis, specialized algorithms have been designed and implemented by taking into account the special underlying structures of the studied problems. These algorithms are based on an efficient and systematic search of an optimal solution using a Branch-and-Bound framework. In the course of the algorithms, the problem of computing the smallest sphere that encloses a given set of vectors

appears as a sequence of subproblems that need to be solved. Our algorithms heavily rely on the recently developed efficient algorithms for this subproblem.

A software has been developed that can implement the proposed algorithms in order to use them in practice. A user-friendly interface has been designed for the software. Extensive computational results reveal that our algorithms are capable of solving large-scale instances of the problems efficiently. Since the architecture of the software has been designed in a flexible and modular fashion, it serves as a solid foundation for further studies in this area.

Keywords: geometric optimization problems, design of algorithms, approximation algorithms, large-scale optimization, clustering problems.

ÖZET

EN KÜÇÜK KÜRELERLE DEMETLEME PROBLEMİ İÇİN ÖZGÜN BİR ALGORİTMANIN GELİŞTİRİLMESİ

Utku Guruşçu
Endüstri Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Doç. Dr. Emre Alper Yıldırım

Temmuz, 2010

Nesnelerin belirli yakınlık kıstaslarına göre gruplara ayrılmaları sürecine literatürde "demetleme" (clustering) adı verilmektedir. Burada temel amaç, verilen nesne kümesindeki yapıyı ve örüntüleri (pattern) doğru bir şekilde tanımlayabilmektir. Dolayısıyla, kümeleme süreci sonucunda ortaya çıkacak olan gruplarda aranan nitelik, aynı gruba ait olan nesneler arasındaki yakınlık ilişkisinin farklı gruplara ait olan nesneler arasındakine göre daha yüksek olmasıdır.

Kümeleme probleminin tesis yerleşimi, büyük ölçekli verilerin tasnifi ve pazarlama gibi çok değişik alanlarda uygulamaları bulunmaktadır. Bu uygulamalarda büyük ölçekli kümeleme problemlerinin etkin çözümüne gereksinim duyulmaktadır.

Bu tez çerçevesinde kümeleme probleminde verilen nesneleri temsil eden ve yüksek boyutlu bir uzayda yer alan m tane vektörü kapsayan, yarıçapları toplamı veya en büyüğünün yarıçapı en küçük olan k tane kürenin hesaplanması problemleri ele alınmıştır. Bu problemlerin çözümleri sonucunda problemlerde verilen nesneler, birbirlerine olan yakınlık ilişkilerine göre k tane gruba ayrılmaktadır.

Sözü edilen matematiksel problemler, evrensel olarak en zor problemler sınıfında yer almaktadır (NP-zor). Literatürde, problemlerin sadece en iyi çözümlerini hesaplamanın değil, iyi bir yaklaşık çözümlerini hesaplamanın bile evrensel olarak zorluğu gösterilmiştir.

Bu tezde problemlerin özgün yapıları kullanılarak özel çözüm yöntemleri geliştirilmiştir. Bu çözüm yöntemleri, dal-sınır yöntemi kullanılarak en iyi çözümün sistemli ve etkin bir şekilde aranması üzerine kurgulanmıştır. Bu çözüm sürecinde verilen vektörleri kapsayan tek bir kürenin hesaplanması, sürekli

çözülmesi gereken bir alt problem olarak ortaya çıkmaktadır. Bu alt problemlerin çözümü için son zamanlarda geliştirilen etkin çözüm yöntemlerinden faydalanılmıştır.

Geliştirilen çözüm yöntemleri, bir yazılıma dönüştürülerek uygulamada kullanılmaları sağlanmıştır. Geniş çevrelerin kullanımını sağlayabilmek amacıyla yazılımda kullanılabilirlik artırılmıştır. Yapılan kapsamlı deneysel hesaplama çalışmaları sonucunda geliştirilen yöntemlerin büyük ölçekli problemleri etkin bir şekilde çözebildikleri ortaya çıkarılmıştır. Geliştirilen yazılım, diğer pek çok geometrik eniyileme problemlerine de uygulanabilecek şekilde esnek ve modüler bir yapıda tasarlandığı için gelecekteki benzeri akademik çalışmalar için önemli bir alt yapı teşkil etmektedir.

Anahtar sözcükler: geometrik eniyileme problemleri, çözüm yöntemi tasarımı, yaklaşık çözüm yöntemleri, büyük ölçekli eniyileme, kümeleme problemi.

To my parents . . .

Acknowledgement

I would like to sincerely thank to my supervisor and advisor, Assoc. Prof. Dr. Emre Alper Yıldırım, for his invaluable support, encouragement, guidance and useful suggestions throughout this work. With his support and advices, he has always been more than an advisor to me.

I am also grateful to Prof. Dr. Barbaros Tansel and Assoc. Prof. Dr. Uğur Doğrusöz for accepting to read and review this thesis and for their invaluable suggestions.

I am as ever, especially indebted to my family, Hatice and İrfan Guruşçu and Bilge, Ahmet and Onur Yüksel, for their endless love and support. I have always felt very lucky that I am their child, brother or uncle, and I will always do my best to keep them feeling the same for me.

I am especially grateful to my princess, Könül, for her love, encouragement, motivation and endless support which made this thesis possible. I can never thank her enough for being there for me anytime I needed.

I would like to offer my thanks to Esra Aybar for her everlasting patience and help, and her keen friendship during my graduate study. My sincere thanks go to my friends Adnan Tula, İhsan Yanıkoğlu, Safa Onur Bingöl, Merve Çelen, Duygu Tutal, Ezel Ezgi Budak, Ceyda Kırıkçı, Onur Özkök, Sibel Alev Alumur, Yahya Saleh, Hatice Çalık, Ece Zeliha Demirci, Zeynep Aydın, Esra Koca, Ahmed Burak Paç, Esma Koca, Emre Uzun, Burak Ayar, Ali Gökay Erön and all other friends for providing me such a friendly environment to work. I am also indebted to Gülnar, for her delicious snacks and sparing her valuable time for me. Furthermore, my special thanks goes to Sıtkı Gülten for his everlasting patience and help, and his keen friendship during my graduate study.

Finally, I wish to express my special thanks to TÜBİTAK for the scholarship provided throughout the thesis study.

Contents

1 Introduction and Literature Review **1**

1.1 Literature Review 5

2 Problem Definition and Notation **11**

3 The Algorithm **17**

3.1 The Branch-and-Bound Algorithm 17

3.2 Initial Approximate Solution 22

4 Implementation **24**

4.1 Software Package 24

4.2 Implementation Details 29

4.2.1 The Main Module 31

4.2.2 The Menu Module 34

4.2.3 The MEB Module 35

4.2.4 The Initial Control Module 36

4.2.5 The Initial Upper Bound Module 37
4.2.6 The Tree Traversal Module 38
4.2.7 The Node Module 46
4.2.8 The Output Module 48

5 Computational Results 50

5.1 Computational Setup 50
5.2 Algorithmic Setup 54
5.3 Experimental Results 56
5.4 Discussions 58
5.4.1 The Effect of the Problem Type 58
5.4.2 The Effect of the Radius Type 62
5.4.3 The Effect of the Cluster Type 64
5.4.4 The Effect of the Number of Vectors 67
5.4.5 The Effect of the Number of Dimensions 72
5.4.6 The Effect of the Distribution 77
5.4.7 The Effect of the MEB algorithm 80
5.4.8 The Effect of the Tree Traversal Algorithm 81
5.4.9 The General Discussion 84

6 Conclusion 87

A Numerical Results **94**

List of Figures

4.1 Dependencies of Modules 31

4.2 The Branch-and-Bound Tree (No Vector Assignment) 39

4.3 The Branch-and-Bound Tree (With Vector Assignment) 40

4.4 The Branch-and-Bound Tree (With min{*numberOfNonemptyClusters + 1*, k} Child Nodes) 41

5.1 The Effect of Problem Type on the Number of Nodes, $k = 2$ . . . 60

5.2 The Effect of Problem Type on the Running Time, $k = 2$ 60

5.3 The Effect of Problem Type on the Number of Nodes, $k = 3$ . . . 61

5.4 The Effect of Problem Type on the Running Time, $k = 3$ 61

5.5 The Effect of Radius Type on the Number of Nodes, $k = 2$ 62

5.6 The Effect of Radius Type on the Running Time, $k = 2$ 63

5.7 The Effect of Radius Type on the Number of Nodes, $k = 3$ 63

5.8 The Effect of Radius Type on the Running Time, $k = 3$ 64

5.9 The Effect of Cluster Type on the Number of Nodes, $k = 2$ 65

5.10 The Effect of Cluster Type on the Running Time, $k = 2$ 65

5.11 The Effect of Cluster Type on the Number of Nodes, $k = 3$ 66

5.12 The Effect of Cluster Type on the Running Time, $k = 3$ 67

5.13 The Effect of Number of Vectors on the Number of Nodes, $k = 2$. 68

5.14 The Effect of Number of Vectors on the Running Time, $k = 2$ . . 69

5.15 The Effect of Number of Vectors on the Number of Nodes, $k = 2$. 70

5.16 The Effect of Number of Vectors on the Running Time, $k = 2$ . . 70

5.17 The Effect of Number of Vectors on the Number of Nodes, $k = 3$. 71

5.18 The Effect of Number of Vectors on the Running Time, $k = 3$ . . 71

5.19 The Effect of Number of Vectors on the Number of Nodes, $k = 3$. 72

5.20 The Effect of Number of Vectors on the Running Time, $k = 3$ . . 72

5.21 The Effect of Number of Dimensions on the Number of Nodes, $k = 2$ 73

5.22 The Effect of Number of Dimensions on the Running Time, $k = 2$ 73

5.23 The Effect of Number of Dimensions on the Number of Nodes, $k = 2$ 74

5.24 The Effect of Number of Dimensions on the Running Time, $k = 2$ 74

5.25 The Effect of Number of Dimensions on the Number of Nodes, $k = 3$ 75

5.26 The Effect of Number of Dimensions on the Running Time, $k = 3$ 76

5.27 The Effect of Number of Dimensions on the Number of Nodes, $k = 3$ 76

5.28 The Effect of Number of Dimensions on the Running Time, $k = 3$ 77

5.29 The Effect of Distribution on the Number of Nodes, $k = 2$ 78

5.30 The Effect of Distribution on the Running Time, $k = 2$ 78

5.31 The Effect of Distribution on the Number of Nodes, $k = 3$ 79

5.32 The Effect of Distribution on the Running Time, $k = 3$ 79

5.33 The Effect of MEB algorithm on the Number of Nodes, $k = 2$ . . 80

5.34 The Effect of MEB algorithm on the Running Time, $k = 2$ 81

5.35 The Effect of Tree Traveral Algoritm on the Number of Nodes, $k = 2$ 82

5.36 The Effect of Tree Traveral Algoritm on the Running Time, $k = 2$ 82

5.37 Number of Nodes for the Specific Instances 84

5.38 Running Time for the Specific Instances 84

List of Tables

5.1 The Radius Types for $k = 2$ 52

5.2 The Radius Types for $k = 3$ 52

5.3 The Cluster Types for all Radius Types for $k = 2$ 53

5.4 The Cluster Types for Radius Types 1, 7, 10 for $k = 3$ 53

5.5 The Cluster Types for Radius Types 2, 3, 4, 6, 8, 9 for $k = 3$ . . . 53

5.6 The Cluster Types for Radius Type 5 for $k = 3$ 54

5.7 Maximum Memory Usages of 5 Specific Instances, $k = 2$ 83

A.1 Number of Examined Nodes in the Branch-and-Bound Tree, $k = 2$ 94

A.2 Running Time, $k = 2$ 97

A.3 Number of Examined Nodes for Branch-and-Bound Tree, $k = 3$. 100

A.4 Running Time, $k = 3$ 104

Chapter 1

Introduction and Literature Review

Clustering is the process of organizing objects into groups whose members are similar in some ways. The main objective is to identify the underlying structures and patterns among the objects correctly. Therefore, a cluster is a collection of objects which are more similar to each other than to the objects belonging to other clusters.

The clustering problem has applications in wide-ranging areas including information retrieval (M. Charikar *et al.* [7]), facility location (Z. Drezner (ed.) [13]) and data mining (R. Agrawal *et al.* [2]). For example, clustering of customers according to their shopping habits enables firms to develop more cost efficient and effective marketing techniques such as informing their customers only about the products they are interested in. For instance, large-scale enterprises may record all information about the transactions of their customers, such as age and postal code, along with the list of purchased items into their database, and then they may wish to use this information in order to devise medium term and long term marketing strategies. In addition, clustering of the products and the way they are presented to customers have a significant role on sales. The locations of goods in supermarkets or listing of products in e-commerce web sites are such examples

of clustering.

Other application areas of the clustering problem include the classification of plants or animals according to their genetic characteristics, clustering of books in the library according to their subjects, authors or editions, grouping of patients in hospitals according to their bloodtype. Therefore, designing and implementing specific and efficient algorithms for these problems has a significant importance.

The use of computers in decision support systems has increased with the development of information technology. Rapid developments in computer technology has brought new perspectives to operations research. While larger scale problems can be solved within a shorter amount of time, much larger scaled problems that need solutions have arisen. For example, in the above marketing case, more volumes of data can be stored in their database. Nevertheless, this type of database must be administered in a more systematic and efficient way in order to keep integrity. Generally, the increase in the dimension of new problems is much faster than the increase in the dimension of solvable problems. Therefore, efficient algorithms are essential for large-scale clustering problems.

One of the crucial components of the clustering problem is to accurately and meaningfully define the closeness criterion among objects. Different closeness criteria exist in the literature for certain types of clustering problems. In the marketing example above, two customers who live nearby and whose ages and shopping lists are similar to each other can be defined as "close" under all meaningful "closeness" criteria. First, parameters that will be used to relate the objects must be identified. Then, each parameter is represented as a dimension in a high-dimensional space. Therefore, each object can be represented as a vector in this resulting space.

The distance among the vectors determines the similarity of the objects. We use the Euclidean distance as a measure of similarity among the objects. For example, customer's age, postal code, and expenses for electronic goods can be considered as the parameters that will be used to cluster customers. Therefore, a three dimensional space is constructed and every customer is represented as

a three dimensional vector. The similarity among customers can then be identified by the distance among the corresponding vectors in this space. Vectors corresponding to similar customers are closer to each other whereas vectors corresponding to dissimilar customers are not.

After defining the closeness criterion, clustering of objects can be mathematically expressed as "the grouping of vectors, corresponding to objects, as clusters that satisfies certain closeness criteria in high dimensional space". The distance among the vectors within a certain cluster must be as small as possible.

Clusters can be defined in several ways. One common approach is to define a cluster by a simple geometric object that covers all the vectors within a cluster. Spheres, ellipsoids, and boxes are usually chosen as covering geometric objects since they are easy to represent.

Next, the number of groups (k) must also be specified. There are two approaches for determining k while enclosing a given set of vectors: (1) computing k geometric objects according to a predetermined objective function, (2) computing minimum number of geometric objects while ensuring a given enclosing criteria. Hence, k is a parameter of the problem for the first approach, and is a decision variable of the problem for the second approach. Thus, the decision maker predetermines the number of groups in the former approach, while the number of groups is determined only after the solution of the problem in the latter approach.

In this study, sphere is used as the enclosing geometric object. Moreover, we assume that the decision maker predetermines the number of clusters. Therefore, k is a parameter of the problem. In other words, the first approach is adopted. Therefore, the problems studied in this thesis can be defined as computing k spheres that enclose all the given vectors, while minimizing a certain objective function. These types of problems are called geometric optimization problems due to their geometric structure.

Only the first approach (k is a parameter) is covered in the context of this study since it can be used to obtain a solution for the second approach (k is a

decision variable) easily. If k is a parameter, the minimum number of clusters that satisfies certain clustering properties can be computed by solving the problem with carefully selected k values. For instance, a binary search over k would be sufficient for this approach. As a result, our algorithms can be used for the solution of the second approach together with a binary search.

Within this study, several factors are taken into consideration during the selection of geometric objects. In recent years, many researchers have developed efficient algorithms that can compute a sphere ($k = 1$) which encloses a given vector set (Yıldırım [46]; Ahipaşaoğlu and Yıldırım [3]). This problem is known as the 1-center or the minimum enclosing ball (MEB) problem. The proposed algorithms are able to solve large-scale MEB problems. Some of these algorithms have been tested and the computational results demonstrate their efficiency in practice. These studies and their results led us to select the sphere as the covering geometric object. The clustering problem for $k = 1$ is a special case of the optimization problems studied in this thesis. Since the algorithm for $k = 1$ is solved repeatedly in our approach, it provides a basis for developing algorithms for cases where k can take different values.

Finally, we use two distinct objective functions. Hence, the clustering problems studied in the scope of this thesis can be defined as the computation of k spheres that enclose a given set of vectors, which represents the set of objects, in a high dimensional space in such a way that the radius of the largest sphere or the sum of the radii of spheres is as small as possible. We develop an efficient algorithm based on a systematic and efficient search of an optimal solution using a branch-and-bound framework. Then, we implement our algorithm and develop a software package. As a result, a user friendly software package is developed for certain clustering problems. Encapsulation, abstraction, modularity, usability and flexibility are determined as the fundamental necessities of the software package. Finally, experimental studies reveal that the proposed algorithms are able to solve large-scale clustering problems in a reasonable amount of time.

The remainder of this thesis is organized as follows: A review of related literature is provided in the remainder of this chapter. Chapter 2 formally defines

and presents nonlinear mixed-integer formulations of the clustering problems. Chapter 3 is devoted to the approximation algorithms for finding approximate solutions to problems. A review of the implementation of the algorithms and software package is given in chapter 4. Numerical results are presented in Chapter 5. Chapter 6 concludes the thesis by giving an overall summary of the contribution to the existing literature and lists some possible future research directions.

1.1 Literature Review

In the scope of this thesis, we study the problem of computing k spheres in a high dimensional space that enclose a given set of vectors in such a way that the radius of the largest sphere or the sum of the radii of spheres is as small as possible.

In the literature, these problems are initially studied on networks. In this context, a function that corresponds to the distances among the nodes on a network is first defined. Then, the objective is to find the k facility locations on a network in such a way that the maximum distances among the demand points and their respective nearest facilities or the sum of the distances among the demand points and their respective nearest facilities is minimized. These problems are mostly suitable for site selections.

In the pioneer work of the Hakimi [21], the “one-center” and the “one-median” problems are initially formulated and solved on networks. The “one-center” problem aims to locate a single facility on a network in such a way that the maximum distance among the facility and the demand points on a network is minimized, while the “one-median” problem aims to locate again a single facility on a network while minimizing the sum of distances between the facility and the demand points on a network. He introduces the concepts of the “absolute center” and the “absolute median” of a weighted graph. These concepts are the generalizations of the “center” and the “median” of a graph, respectively. Two methods are proposed in the study where the first method is used for locating a switching center (facility) in a communication network optimally and the second method is used

for finding the most suitable location of a site such as hospital or police station in a highway system. The former method formulates and solves the "one-center" problem where the latter one formulates and solves the "one-median" problem on networks.

There are further studies in the literature for the "one-center" and the "one-median" problems on networks. Goldman [19] proposes simple algorithms for the one-median problem, where he locates a single central facility on two different types of simple networks in such a way that the distances among the central facility and the sources of the flow is minimized. S. L. Hakimi, E. F. Schmeichel, J. G. Pierce [24] provide some improvements in Hakimi's method for the "one-center" problem on networks.

The "one-center" and the "one-median" problems on networks are later generalized to the "k-center" and the "k-median" problems on networks by Hakimi [22]. This study proves that the "k-median" of a weighted graph includes at least one of the optimal k switching centers (facilities) of a network. Therefore, this result can reduce the "k-median" problem on networks to a finite search.

Then, several solution methods have been proposed for different k values where $k > 1$ [10, 17, 18, 23, 29, 35, 44, 45] in the literature. One of the most important solution methods, for the k-center problem, in the literature is proposed by Minieka [15]. He shows that there are only a finite number of potential switching centers in a graph which reduces the problem to a finite search while it is enough to solve only a finite series of set covering problems in order to find k centers on a network.

The aforementioned problems show that there is a finite number of alternatives (number of nodes) for determining the k centers or k medians on general networks. However, Kariv and Hakimi [33] prove that finding a k-median of a general network is NP-hard. Similarly, Hsu and Nemhauser [28] and Kariv and Hakimi [32] show that the k-center problem is also NP-hard on general networks. For a review of the studies about the k-center and the k-median problems in a network location literature, the reader is referred to the review papers of Tansel, Francis and Lowe [42, 43].

The k-center and the k-median problems are also studied in high dimensional continuous spaces. These problems aim to find k supply points from a given set of points anywhere in the plane, in such a way that the distance from a point to its respective nearest supply point or the sum of the distances from the points to their respective nearest supply points is as small as possible. Hence, switching centers (facilities) in network location literature correspond to the supply points in continuous spaces. Moreover, these supply points refer to the centers of the spheres in the aforementioned problems. Therefore, there are no constraints on the centers of the spheres. Megiddo and Supowit [38] prove that, even for the plane, both problems are NP-complete.

The results in the literature reveal that it is hard to develop theoretically efficient algorithms for the studied problems since no polynomial time algorithm has been developed for NP-hard problems yet. On the other hand, some efficient algorithms have been proposed for some of the special cases of both the “k-center” and the “k-median” problems in a plane for $k = 2$.

Drezner [12] presents a trivial $O(n^d+1)$-time algorithm for the solution of the planar 2-center problem where n is the number of demand points and d is the dimension of the space. He also develops an efficient algorithm for solving the planar 2-median problem with a maximum of 100 demand points. The efficiency of these algorithms are further improved by using different search techniques. Agarwal and Sharir [1] give an $O(n^2 logn)$-time algorithm for the planar 2- center problem by using the parametric searching method. Afterwards, Matousek [37] uses the randomization method to propose a simpler algorithm with a running time of $O(n^2 logn)$ again for the planar 2-center problem. The running time of the planar 2-center algorithms are also further improved by Hershberger [25] and Jaromzyl and Kowaluk [31], respectively.

The first subquadratic solution to the planar 2-center problem is provided by Sharir [40]. He developes an $O(nlog^9 n)$-time algorithm by integrating the parametric searching technique with various other techniques such as dynamic maintenance of planar configurations. This algorithm is improved to $O(nlog^2 n)$-time algorithm subsequently by Eppstein [16]. However, the running times of

these algorithms depend on the number of supply points, where the size of the problems grows exponentially as a function of k. Therefore, these algorithms can not be generalized, while maintaing the same efficiency, for cases where $k > 2$.

Approximation algorithms are the alternative solution methods that aim to find approximate solutions to various optimization problems rather than exact solutions. These algorithms are often designed and developed for NP-hard problems, while it is not proved that there can ever exist an efficient polynomial time exact algorithm for solving these problems. Therefore, approximation algorithms are often developed for this class of problems. For a given positive ϵ value, $(1+\epsilon)$-approximate solution can be defined as the following for the studied problems. If the optimal value of the problem is r, then the objective function value of the approximate solution will not be more than $(1+\epsilon) \times r$. The solution times of these algorithms are generally inversely proportional with the value of the ϵ.

There are efficient approximation algorithms in the literature for both the k-center and the k-median problems. Gonzalez [20] presents a 2-approximation algorithm ($\epsilon = 1$) for the k-center problem that requires $O(pn)$ computations, where p is the number of clusters and n is the number of points. However, this algorithm lacks to find the solution if the points do not satisfy the triangular inequality. Another 2-approximation algorithm for the k-center problem is proposed in Hochbaum ve Shmoys [26, 27]. They develop general purpose algorithms that works with the problems in wide-ranging areas such as location theory, routing and etc. Furthermore, they show that any algorithm proposed with a better approximation factor will imply that *P*=*NP* for several of these problems. Finally, Feder and Greene [27] prove that, for $n \geq 2$, it is impossible to find an optimal solution to k-center problem within an approximation factor around 1.822 unless $P = NP$.

There are also efficient approximation algorithms for the k-median problem in the literature. Charikar and Guha [8] propose a 6.66-approximation algorithm, first constant factor approximation algorithm, for the k-median problem. Then, Jain and Vazirani [30] present a 6-approximation algorithm for the metric

k-median problem. Shortly after, Charikar and Guha [9] improve Jain and Vazirani's algorithm and develop a 4-approximation algorithm for the metric k-median problem.

Furthermore, in the literature there exist algorithms that are both theoretically and practically efficient for the special cases of the 1-center and 1-median problems. Chrystal and Peirce [41, 11] propose the first known exact algorithm for the MEB problem in the plane. It computes the minimum enclosing ball of m points in the plane in $O(m^2)$ operations for the worst case. However, the number of operations needed to solve these problems grows exponentially as a function of the dimension. Later, Elzinga and Hearn [14] also consider the MEB problem that encloses a given set of points in a high dimensional space in n dimensions. They provide a solution procedure in which the memory requirement of the computer is independent of the number of points. However, the solution time of the procedure grows, approximately, linearly with the number of points. Therefore, a new concept, core set of size ϵ, have arisen in the literature for the minimum enclosing ball problems.

Let us have a set of vectors $S \subset \mathbb{R}^d$, where d is the dimension of the space, and a positive ϵ value ($\epsilon > 0$). An *ϵ-core-set* $P \subset S$ ensures that, if the smallest ball that encloses P is expanded by ϵ, then the resulting ball encloses S. In other words, if the radius of the smallest ball that encloses P is multiplied by $1 + \epsilon$, then the resulting ball contains S. Note that the size of the core set is independent of the number of points and the number of dimensions. The existence of an epsilon core set of size $O(1/\epsilon^2)$ for the minimum enclosing ball problem is first established by Badiou, Har-Peled and Indyk [5]. They propose a $(1+\epsilon)$-approximate algorithm that computes the minimum enclosing ball of a given set in $O(\text{mn}/\epsilon^2 + (1/\epsilon^{10})log(1/\epsilon))$ operations. The existence of an epsilon-core set of size $O(1/\epsilon)$ is found by Badiou and Clarkson [4] and Kumar, Mitchell and Yıldırım [34] independently. Panigraphy [39] constructs the best known complexity bounded algortihm for the fixed ϵ problem, which computes the approximation algorithm in $O(\text{mn}/\epsilon)$ operations.

Finally, Yıldırım [46] focuses on the minimum enclosing ball problem on relatively large scale instances. Two $(1+\epsilon)$ algorithms are developed for the aforementioned problem for a given positive ϵ value. The MEB for a given instance can be computed in $O(\text{mn}/\epsilon)$ operations with both of the algorithms. This result is the same as the best known complexity for fixed ϵ. The extensive computational results reveal that they can compute the algorithm with a smaller size of the core set than the worst case estimates. These agorithms are effective and simple to implement. Moreover, they have good worst case complexities and efficient in practice. These studies have provided a significant background for the solution methods developed in this thesis since the MEB problem arises as a sequence of subproblems in the solution of the studied problems.

Previous studies in the literature, for the geometric optimization problems studied in the scope of this thesis, can be classified mostly as theoratical studies and have not been implemented in practice. However, these types of problems arise in numerous important applications such as data analysis, data mining, image processing and facility location. Therefore, the design and implementation of efficient algorithms are essential for solving such problems. We cover an important gap in the literature by designing and also implementing specific and efficient algorithms for solving certain types of geometric optimization problems.

Chapter 2

Problem Definition and Notation

In this chapter, we give the formal definitions of our problems and introduce the parameters, variables, and the mathematical models that can be used to solve the problems.

We study the problem of computing k minimum enclosing spheres in a high dimensional space that enclose a given set of m vectors in such a way that the radius of the largest sphere (*min-max* problem) or the sum of the radii of spheres (*min-sum* problem) is as small as possible. While the former problem is the same as the k-center problem, the latter one can be considered as a version of k-median problem with a different objective function.

Let $S = \{p^1, p^2, ..., p^m\} \subset \mathbb{R}^n$ be the given vector set, where $p^1, p^2, ..., p^m$ are the vectors that correspond to objects, n is the dimension of the space, and m is the number of vectors. The problem can be viewed as the assignment of m vectors to k groups and the computation of the smallest enclosing ball of vectors in each cluster in such a way that the radius of the largest sphere or the sum of the radii of spheres is as small as possible. Each group of vectors corresponds to a cluster. Intra - group similarity is increased by trying to reduce the distances among the vectors within a cluster.

Note that, if the optimal assignment of m vectors to k groups is known in

advance, the smallest sphere that encloses each cluster can be computed efficiently by the existing minimum enclosing ball (MEB) algorithms. Then, the maximum radius or the sum of the radii of spheres will yield the optimal solution. However, this simple algorithm is not valid for our problems since we do not know the optimal assignment.

Moreover, both of the optimization problems can be solved by using a brute-force approach. First, we assign m vectors to k clusters in all possible ways. Then, we compute the minimum enclosing ball for each cluster. Next, we compute the objective function value for each possible clustering. Finally, we can select the optimal grouping which gives the smallest objective function value. This method is known as complete enumeration. The number of all possible clusterings is finite, which makes the problems solvable. On the other hand, we can arrange m vectors to k groups in k^m possible ways. Therefore, the number of all possible clusterings increases exponentially with the number of vectors. Note that, k^m can be an extremely large number even if k and m are relatively small. For instance, if $k = 2$ and $m = 50$, then k^m is around 1.13×10^{15}, and this number is beyond the computational limit of today's most advanced computers. As a result, the complete enumeration method is computationally feasible for only very small values of k and m. Therefore, it is clear that sophisticated solution methods are required for solving the studied problems efficiently. The studied problems can be formally modeled as a nonlinear mixed-integer programming model (NLMIP) as in Model 1 and Model 2.

min-max problem

Parameters

$$
\begin{aligned}
p^i &= \text{vector } i, i = 1, 2, ..., m \\
k &= \text{number of spheres} \\
M &= \text{big constant}
\end{aligned}
$$

Decision Variables

$$
\beta_{ij} = \begin{cases} 1, & \text{if the } i^{th} \text{ vector is assigned to the } j^{th} \text{ sphere} \\ 0, & \text{otherwise.} \end{cases}
$$

$i = 1, ..., m$, $j = 1, ..., k.$

c^j = center of the jth sphere , $j = 1, ..., k$
r = radius of the largest sphere

Having defined the parameters and the decision variables of the $min - max$ problem, we can can formulate the problem as the following nonlinear mixed-integer optimization model:

Model 1 (NLMIP 1):

$$
\text{Minimize} \quad r \tag{2.1}
$$

Subject to

$$
\sum_{j=1}^{k} \beta_{ij} = 1, \qquad i = 1, ..., m \tag{2.2}
$$

$$
\left\| p^i - c^j \right\| \leq r + (1 - \beta_{ij})M, \qquad i = 1, ..., m, j = 1, ..., k \tag{2.3}
$$

$$
\beta_{ij} \in \{0, 1\}, \quad i = 1, ..., m, j = 1, ..., k \tag{2.4}
$$

$$
c^j \in \mathbb{R}^n, \quad j = 1, ..., k \tag{2.5}
$$

$$
r \in \mathbb{R} \tag{2.6}
$$

min-sum problem

Parameters

$$\begin{aligned} p^i &= \text{vector } i, i = 1, 2, ..., m \\ k &= \text{number of spheres} \\ M &= \text{big constant} \end{aligned}$$

Decision Variables

$$\beta_{ij} = \begin{cases} 1, & \text{if the } i^{th} \text{ vector is assigned to the } j^{th} \text{ sphere} \\ 0, & \text{otherwise.} \end{cases}$$

$i = 1, ..., m$, $j = 1, ..., k$.

c^j = center of the jth sphere , $j = 1, ..., k$
r^j = radius of the jth sphere , $j = 1, ..., k$

Having defined the parameters and the decision variables of the $min - sum$ problem, we can can formulate the problem as the following nonlinear mixed-integer optimization model:

Model 2 (NLMIP 2):

$$\text{Minimize} \quad \sum_{j=1}^{k} r^j \tag{2.7}$$

Subject to

$$\sum_{j=1}^{k} \beta_{ij} = 1, \qquad i = 1, ..., m \tag{2.8}$$

$$\left\| p^i - c^j \right\| \le r^j + (1 - \beta_{ij})M, \qquad i = 1, ..., m, j = 1, ..., k \tag{2.9}$$

$$\beta_{ij} \in \{0, 1\}, \quad i = 1, ..., m, j = 1, ..., k \tag{2.10}$$

$$c^j \in \mathbb{R}^n, \quad j = 1, ..., k \tag{2.11}$$

$$r^j \in \mathbb{R}^n \quad j = 1, ..., k \tag{2.12}$$

In the above two mathematical models, the objective functions are very similar. In the first model, the objective is to minimize the maximum of the radii of spheres, whereas in the second model the objective is to minimize the sum of the radii of spheres.

The first constraint set is the same for both models. In this set, there is a constraint for each vector, which implies that there is a total of m constraints. We ensure that a vector is assigned to exactly one cluster and no vector remains unassigned.

The second constraint set is also similar in both models. It includes a constraint for each vector and each cluster, which implies that there is a total of $m \times k$ constraints. If p^i is assigned to the jth cluster, then β_{ij} is equal to one. Therefore, the constraint that corresponds to the (i, j) pair ensures that the distance between p^i and c^j ,which represents the center of the jth cluster, can be at most r for Model 1, and at most r^j for Model 2. Thus, p^i is enclosed by the unique sphere whose center is c^j and radius is r for Model 1, and r^j for Model 2. If p^i is not assigned to the cluster j, then β_{ij} is equal to 0. In this case, the right hand side of the (i, j) pair becomes a large number. Therefore, the constraint on the distance between the vector p^i and the center of jth cluster c^j becomes redundant. For both problems, M must be a big enough constant in order to satisfy this condition. For instance, M can be selected as the distance among the furthest vectors since this is an upper bound on r for Model 1 and r^j for Model 2.

The third and the fourth set of constraints for both models ensure that the assignment variable is a binary variable and the variables corresponding to the centers of the clusters are free.

The fifth set of constraints are different for both models, but both constraints ensure that the variables corresponding to the maximum radius in the $min - max$ problem and the variables related to the radii in the $min - sum$ problem are free. Nevertheless, one must pay attention that r or r^j can only take nonnegative values due to the first constraint set.

There are totally $km + kn + 1$ decision variables in Model 1 and $km + kn + k$ decision variables in Model 2. In addition to this, there are $m \times k$ coverage and m assignment constraints in both models.

Furthermore, the first constraint set depends on the distance between the vectors and the centers of the clusters. Therefore, these constraints are nonlinear. Moreover, β_{ij} are binary variables, hence there exists integer variables in both models. Therefore, both models are nonlinear mixed-integer programming models.

Note that, there exists commercial solvers for solving both of the problems. Most of them are licensed products such as DICOPT and MINLP, and etc. These solvers can only be used with other licensed products such as GAMS. We have solved our models using these solvers. However, these solvers are able to solve only very small scaled problems (15 vectors, 5 dimensions) in reasonable time. In other words, even small-scaled instances of the problems can not be solved with these commercial solvers. Therefore, we have concluded that these solvers can not exploit the specific structure of our problems. As a result, we have focused on designing and implementing specialized algorithms that are able to use the specific geometric structure of the problems.

In the recent years, mixed-integer nonlinear problems have arisen in a variety of applications (Leyyfer *et al.* [36]). Several methods exist for solving such problems. The branch-and-bound method is one of the algorithms that is used for solving various optimization problems. The aim of the method is to search for an optimal solution in a systematic and efficient way where the integrality constraints are initially relaxed, and then added to the model subsequently. We have developed and implemented a specialized branch-and-bound method that exploits the underlying problem structure and tries to solve the problems efficiently in this thesis.

Chapter 3

The Algorithm

As mentioned before, commercial solvers fail to solve the studied problems efficiently since they are general purpose solvers and are unable to exploit the specific geometric structure of the problems. In this chapter we first present a branch-and-bound algorithm that initially finds a good feasible solution and then solves each of the problems in a systematic and efficient way by making use of initial feasible solution. Then, we present the algorithm that computes the initial approximate feasible solution.

3.1 The Branch-and-Bound Algorithm

If the optimal assignment of m vectors to k clusters is known in advance, each of the problems can be solved easily by computing the smallest spheres that enclose the vectors in each cluster. While we do not know the optimal assignment in advance, we need to develop a systematic and efficient search method. The branch-and-bound method was identified to be the most suitable method for solving the studied problems.

Let $S = \{p^1, p^2, ..., p^m\} \subset \mathbb{R}^n$ be the given vector set, where $p^1, p^2, ..., p^m$ are the vectors that correspond to objects, n is the dimension of the space and

m is the number of vectors. r^j corresponds to the radius and c^j corresponds to the center of the jth sphere. Moreover, C_j represents the jth cluster. The branch-and-bound algorithm for solving each of the problems is as follows where the expression in paranthesis corresponds to the objective function value of the $min - sum$ problem and the expression outside the paranthesis corresponds to the objective function vaue of the $min - max$ problem:

Algorithm 1: The Branch-and-Bound Algorithm

Input: $S = \{p^1, p^2, ..., p^m\} \subset \mathbb{R}^n$, k, m, Initial Upper Bound, Best Clusters, Best Radii, Best Centers

1 **begin**
2 **initialize**
3 *BestRadii*←Best Radii;
4 *BestCenters*←Best Centers;
5 *BestClusters*←Best Clusters;
6 *UpperBound*←Initial Upper Bound;
7 $C_1 \leftarrow \{p^1\}$;
8 $C_j \leftarrow \emptyset, j = 2, ..., k$;
9 **For** $j = 1$ to k
10 Compute the MEB for cluster C_j, assign its center to c^j, and radius
11 to r^j;
12 **end for**
13 **if** $\sum_{j=1}^{k} r^j \geq UpperBound[max_{j=1,...k} r^j \geq UpperBound]$
14 Stop branching (Pruning);
15 **end if**
16 **if** any unassigned vector is enclosed by any sphere
17 Assign it to the enclosing sphere whose center is the closest to
18 corresponding vector;
19 **end if**
20 **if** all vectors are assigned to clusters
21 Stop branching ;
22 **if** $\sum_{j=1}^{k} r^j < UpperBound[max_{j=1,...k}\ r^j < UpperBound]$
23 *UpperBound*← $\sum_{j=1}^{k} r^j [UpperBound \leftarrow \max_{j=1,...k}\ r^j]$
24 *BestRadii*← $\{r^1, r^2, ..., r^k\}$
25 *BestCenters*← $\{c^1, c^2, ..., c^k\}$
26 *BestClusters*← $\{C_1, C_2, ..., C_k\}$
27 **end if**
28 **end if**
29 **if** there exists any unassigned vector (p^i)
30 **For** $j = 1$ to *numberOfNonemptyClusters+1*
31 $C_j \leftarrow C_j \cup \{p^i\}$;
32 Go to step 9;
33 **end for**
34 **end if**
35 **Return** *BestRadii*, *BestCenters*, *BestClusters*

We can explain the practical performance of the algorithm by a branch-and-bound tree. Every node of a tree corresponds to a partial grouping obtained so far. Each node has zero or more child nodes. Each of these children is obtained by assigning an unassigned vector to a different cluster. If all the vectors are assigned to clusters in a node, such a node is called a leaf node and leaf nodes do not have any children nodes. This structure has to be constructed carefully in order to provide both time and memory efficiency.

The algorithm aims to search systematically for the arrangement of m vectors to k groups using different objective functions. To begin with, the given vector set S, the number of spheres k, the initial upper bound, the best clusters, the best radii and the best centers are the input parameters of the algorithms. Without loss of generality, we can assign the first vector p^1 to the first cluster C_1 initially in order to break symmetry which will be detailed further in the following sections. *UpperBound* parameter constitutes an initial upper bound on the optimal value. *BestRadii*, *BestCenters* and *BestClusters* parameters constitutes the best radii, the best centers and the best clusters obtained so far. The values of these parameters are initially computed during the initial upper bound computation. We have used an efficient algorithm for computing the initial upper bound value which will be explained in detail in the next section.

Next, we compute the MEB for each cluster. The radius of each ball that corresponds to a cluster is assigned to r^j, and the center of each ball is assigned to c^j.

Following this, we check whether the radius of the smallest ball for the $min-max$ problem or the sum of the radii of the balls for the $min-sum$ problem is greater than the *UpperBound* value or not. If this condition is satisfied, we stop branching for this partial grouping (pruning) since it cannot be a part of the optimal solution. Therefore, we are able to prevent any partial groupings that start with wrong assignments and the number of potential nodes can be significantly decreased.

After computing the MEB for each cluster, we check whether any unassigned vector lies inside any balls. We assign a vector to only one cluster in each node.

Therefore, it is enough to control only the newly constructed ball whether it contains any unassigned vector since all other balls are controlled before. If there is any such vector, it will be assigned to the corresponding cluster. If there is an unassigned vector that is enclosed by more than one ball, it will be assigned to the ball whose center is closest to itself. We therefore aim to increase the number of assigned vectors in partial clusterings without changing the structure of the balls and so decrease the size of the branch-and-bound tree. This approach mainly may lead us to reach a leaf node as soon as possible.

As a result, if all the vectors are assigned to clusters, we reach a feasible solution for each of the problems. Therefore, we update the *UpperBound*, *BestRadii*, *BestCenters* and *BestClusters* parameters if the radius of the smallest ball for the $min - max$ problem or the sum of the radii of the balls for the $min - sum$ problem is less than the *UpperBound* value.

If there are still unassigned vectors at the end of the clustering process, we continue branching. As for the new entry, we aim to select the vector that will minimize the number of unassigned vectors in the subsequent steps. We use a max - min approach for this selection. After finding the closest vector to each of the cluster centers, we select the furthest vector from its repective nearest cluster center as the new entry vector. Then, the new entry vector will be assigned to *numberOfNonemptyClusters + 1* clusters in order to prevent symmetry in clusters. The whole algorithm is repeated in the next stage. At the end of the algorithm, m vectors are assigned to k clusters.

For each node of the tree, only one cluster's geometric structure changes since the new vector is added only to that cluster. Therefore, it suffices to solve exactly one MEB problem for each node of the tree. Hence, the use of an efficient algorithm for solving the MEB problem is crucial to improve the solution time of the problem. Efficient algorithms are used for computing the MEB (Yıldırım [46]; Ahipaşaoğlu and Yıldırım [3]).

3.2 Initial Approximate Solution

Notice that, decreasing the number of potential nodes in the branch-and-bound tree plays an important role in the efficiency of the algorithm. As mentioned before, we stop branching if the objective function value of a node exceeds the upper bound value. Therefore, obtaining a good initial upper bound value (feasible solution) enables us to prune a potentially larger number of nodes.

Prior to the development of the algorithm, we concentrated on finding an efficient algorithm for computing the initial upper bound value. For instance, the assignment of m vectors to k clusters randomly yields a feasible solution. However, this solution may coincide with the optimal solution, or it may be quite far from it.

Therefore, we have decided to find a more accurate approach for computing the initial upper bound value. We focused on ease of implementation and the approximation factor of the algorithm. Hence, an approximation algorithm, which is easy to implement is used for the $min - max$ problem (Gonzalez [20]; Hochbaum and Shmoys [26, 27]).

The algorithm starts with selecting an arbitrary vector from the given vector set S. The furthest vector from this randomly selected vector represents the center of the first cluster. Then, the furthest vector from the center of the first cluster represents the center of the second cluster. If k is predetermined as 2, we stop searching cluster centers. Otherwise, we compute the distances among all unassigned vectors and existing cluster centers. Then, we select the furthest vector from the respective nearest cluster center as the center of the following cluster. We repeat this approach until we find k centers for k clusters. The aim is to select the cluster centers as far apart as possible. After determining the cluster centers, every unassigned vector is assigned to the closest cluster center. As a result, m vectors are assigned to k clusters, so we obtain a feasible solution. The initial upper bound value can be at most 2 times the optimal value using this approach (Gonzalez [20]; Hochbaum and Shmoys [26, 27]).

Although this algorithm does not give a theoretically good approximation for the $min-sum$ problem, it gives a feasible solution for it. There are approximation algorithms for the $min-sum$ problem but they are considerably more difficult to implement (Charikar, M. and S. Guha [9]). So, we choose to use the same algorithm for computing an initial upper bound value for both of the problems even though it may not return a provably good feasible solution for the $min-sum$ problem.

Chapter 4

Implementation

One of the main goals of this study is to test the efficiency of proposed algorithms in practice by solving medium to large scale instances of the previously defined geometric optimization problems. To this end, we implement our algorithms, and develop a software package. This chapter is devoted to the resulting software package and implementation of the algorithms.

4.1 Software Package

First, we aim to find the most appropriate programming language for implementing the branch-and-bound algorithm. We identify specific selection criteria. To begin with, we wish to release the resulting software package for free use of the scientific world. Therefore, we try to make it compatible with most of the commercial and noncommercial operating systems. Prevalence of the programming language is another concern because we aim to have high participation rates in the further developement of the software package. In addition to this, we try to select a middle-level programming language that lies at the interface of high-level and low-level programming languages. Furthermore, we wish to deal with the implementation of "The Big Picture". Last but not least, efficient memory management capability and run time speed are defined as other criteria. As a result,

C++ seems to be the most appropriate programming language for developing the software package. Therefore, we determine C++ as the programming language.

Having decided the programing language, we identify the following software design metrics.

- **Flexibility:** The resulting software package has a decisive role in this research area. Therefore, a flexible structure is designed.
- **Usability:** The widespread use of the resulting software package is aimed, hence usability is increased.
- **Modularity:** Software is partitioned into separate and independent parts called modules in order to improve the sustainability.
- **Encapsulation:** Information hiding is provided by encapsulation in order to increase the robustness of the software package and limit the interdependencies of the components.
- **Abstraction:** The software package is partitioned to its most fundamental parts by abstraction. The abstract data types are modelled by classes in software.
- **Compliance of Technical Infrastructure of Software:** We aim to minimize the memory usage to increase the dimension of the solvable instances of our problems. Advanced data structures are used for keeping data in memory.

After stating the software design metrics, we develop initial pseudo-code for our algorithm. The pseudo-code is as follows :

- Input data
- Perform initial control
- Compute initial upper bound
- Solve problem
- Output results

The algorithms are implemented via the commercial product Microsoft Visual Studio 6.0 and non-commercial product UNIX Command Window simultaneously in order to obtain synchronization.

Next, we design the technical infrastructure of our software. First, we create independent and separate classes for different modules. We define all parameters and functions of classes in library files *(.h)*, while we code functions in method compiler files *(.cpp)*. Functions are defined as public and parameters are defined as private members of classes. Moreover, a main file is created in order to compile the whole program *(main.cpp)* and so provide integrity. We create the following files:

1. Library Files
    - mebClass.h
    - menuClass.h
    - nodeClass.h
    - outputClass.h
    - searchClass.h
    - upperBoundClass.h
    - initialControlClass.h

2. Method Compilers
    - mebFunctions.cpp
    - menuFunctions.cpp
    - nodeFuntions.cpp
    - outputFunctions.cpp
    - searchFunctions.cpp
    - upperBoundFunctions.cpp
    - initialControlFunctions.cpp
3. Main Compiler
    - main.cpp

We can summarize the general structure of these files as follows:

- Each class has distinct names.
- The parameters and the functions of the classes are defined in library files *(.h)*.
- The functions are coded in correponding method compiler files *(.cpp)*.
- Objects are used for accessing to the classes.
- Objects are created as pointers and these pointers are deleted when they are no longer needed.

We provide the integrity by creating own method compiler files for each library file. Moreover, classes have some functions that perform the same operations for each class. These functions and their intended usages can be summarized as follows:

- Constructors:

  - We use constructors for assigning initial values to some data members during object creation. Constructors are invoked whenever a new class object created.
    * Default Constructor:
      · Constructor with no arguments.
    * Parameterized constructor:
      · Constructor with arguments.

- Destructors:
  - Destructors are executed whenever an instance of the class deleted. We release the private resources by destructors.

- Set and Get functions:
  - We can get read or write access to private data members by setter and getter functions.

4.2 Implementation Details

Computational complexity theory analyzes the amount of resources that is needed to solve computational problems. Time and space complexitites are the most important measures of the computational complexity. Time complexity measures the number of steps required for solving an instance of a problem whereas the amount of the memory used for solving this instance is studied in the context of space complexity. Since all algorithms have space and time constraints, both complexities are crucial.

As mentioned before, we implement a specialized algorithm for clustering problems using minimum enclosing balls. Therefore, we have to consider our algorithms in terms of both the time and the space complexities. The number of nodes in our branch-and-bound tree increases exponentially as a function of k. Therefore, it is crucial to develop an efficient algorithm for decreasing the number of nodes in our branch-and-bound tree. For example, let us have a small-scale instance with m=*50*, n=*2* and k=*2*. In the worst, the branch-and-bound tree can have 1.13×10^{15} nodes. This number is beyond the computational limits of today's most advanced computers. Hence, we initially aim to develop an efficient algorithm to decrease the number of nodes in the branch-and-bound tree.

However, there are other important issues that must be handled during the implementation of these algorithms. One of these issues is the space constraint where memory must be used efficiently and effectively. If this is not achieved, the size of the solvable instances of problems decreases considerably. Therefore, it is important to use the most appropriate data types and data structures during the implementation in order to provide memory efficiency. We use the following data types and data structures:

- We use *int*, *float*, *double*, *char*, *unsigned*, *string* and *boolean* data types.
- We use arrays if we are working with a data of fixed size since arrays keep less memory than other containers.
- A vector is a kind of a container that is implemented as dynamic arrays. We

can handle storage automatically in vectors where they are used to access individual elements with their position index. We use vectors, if we are working with a data of dynamic size and do not delete any member of the data.

- A set is a type of a container that is used to store unique elements which are sorted in ascending order. We use sets, if were are working with a data which is sorted.

- Lists are used when we need to add or remove elements anywhere in that container. We use lists, if we are working with a data of dynamic size in which we can delete any member of the container in any time.

The memory allocated for data types, vectors, lists and sets are freed whenever the destructor of the class is invoked. On the other hand, we can free the memory of arrays whenever we want. All arrays, vectors, lists and sets can keep data variables in all kinds of data types.

We do not only implement our algorithms, but also develop a software package for solving certain clustering problems. Our software package is composed of separate and independent modules. All separate modules have different and specialized functionalities. These modules are represented by separate classes in order to maintain modularity. The dependencies of these modules are illustrated in Figure 4.1. Moreover, we try to use the most efficient data structures in these modules in order to decrease the amount of memory used.

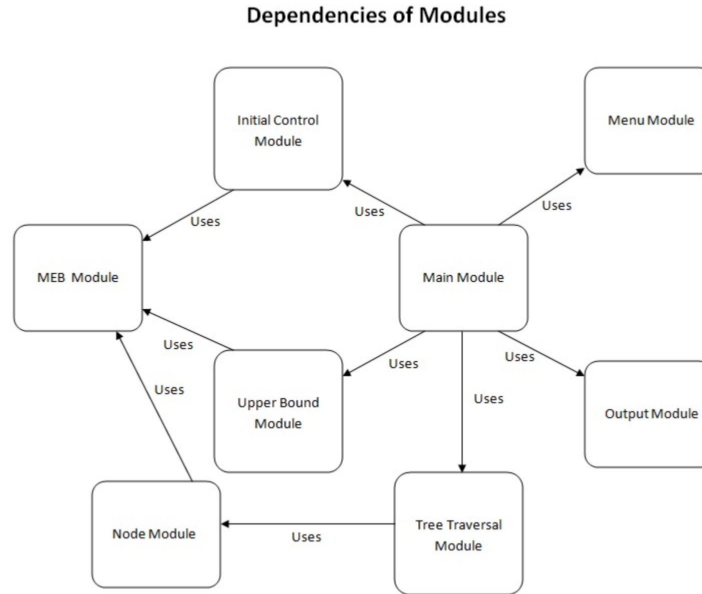

Figure 4.1: Dependencies of Modules

All the contents of these modules are summarized below:

4.2.1 The Main Module

The main module corresponds to the main compiler file (main function) in our software. Our program starts execution from the main function which organizes the rest of the program by invoking the classes that correspond to separate modules. We also provide integration and synchronization of the modules via this function. The algorithm of this module is as follows:

Algorithm 2: Algorithm of the Main Module

```
1  begin
2      initialize
3          Define parameters;
4          Open files for outputs;
5      Perform menu operations;
6      Start time;
7      if Initial condition is satisfied
8          Go to step 14;
9      end if
10     else
11         Compute initial upper bound;
12         Compute optimal solution by constructing the branch-and-bound tree;
13     end else
14     Stop time;
15     Compute run time;
16     Output results;
17     Restart or terminate the program;
18
```

We define all necessary parameters of the program in the main function. These parameters can be altered by the results coming from other modules. These parameters are;

- *Integer* type parameters:
  - *searchMethod:* Tree traversal algorithm type
    - *DFS:* Depth First Search
    - *BFS:* Breadth First Search
    - *BEST:* Best First Search
    - *RFS:* Random First Search
    - *HS:* Hybrid Search
  - *algorithmType:* MEB algorithm type
    - *Meb_u*
    - *Meb_u_elim*
    - *Meb_u_away*

    - *Meb_u_away_elim*
  - *problemType:* Problem type
    - *min-max*
    - *min-sum*
  - *numberOfClusters:* Number of clusters (k)
  - *dimension:* Number of dimensions (n)
  - *pointNumber:* Number of points (m)
  - *numberOfLBPrunes:* Number of prunes in the branch-and-bound tree
  - *numberOfExaminedNodes:* Number of examined nodes in the branch-and-bound tree
  - *numberOfReachedLeaves:* Number of reached leaves in the branch-and-bound tree
  - *numberOfOpenNodes:* Maximum number of active nodes in the branch-and-bound tree
- *Double* type parameters;
  - *tolerance:* Tolerance
  - *upperBound:* Best solution obtained so far
  - *timeDifference:* Running time of the program
  - *vm :* Maximum virtual memory usage
  - *rss :* Maximum resident set size usage
- *Double* type two dimensional arrays:
  - *userMatrix:* Vector set (S)
- *Integer* type vector of vectors:
  - *clusters:* Vectors in clusters

These parameters are defined prior to the execution of the algorithm and used during the run time. As a result, for each class (module) in the main function, we;

- Create the object,
- Return the results,
- Erase the object.

4.2.2 The Menu Module

In software engineering, a menu corresponds to a list of commands that is presented to users. Users can give instructions to the computer via menus.

We construct a menu for our software in order to provide convenient access to various operations. Users define all parameters of the program via menu class. We therefore designed a user-friendly interface for menu operations in order to maintain the clarity of the program.

Once the program starts, the user defines input parameters of the program to execute the code. Hence, the initial object is created for the menu class that corresponds to the menu module. Thus, users can perform the following operations via menu class.

- The user defines the following parameters respectively from both the menu screen or built-in files. These are constant parameters and cannot be altered during the execution.
  - k, m, n, S
  - Problem type
  - Tree traversal algorithm type
  - MEB algorithm type
  - Tolerance

If the user prefers to take S from the screen, there are two choices. The matrix can be constructed manually or randomly according to normal or uniform

distribution. After providing parameters, the user can not intervene with the program until it finds an optimal solution or manual termination.

Users may face user or code-related problems in all kinds of software. Software engineers have to deal with each types of these problems. Entering an integer by mistake instead of entering a character can be considered as a user-related problem. Conversely, trying to divide a number by zero is a code-related problem. When these problems occur, warning the user with an output message is a better programming practice than crashing the code or terminating the program. These problems that can arise during the execution of the program must be foreseen by programmers. This is one of the key aspects of software engineering.

Exception handling is a supervision mechanism that deals with all kinds of problems during the run time. We design a comprehensive exception handling mechanism for our software where both user and code originated errors are handled in detail. As a result, in case of problems, we prefer to give tangible error messages rather than terminating the program.

Finally, we assign necessary parameters to the variables in the main file. Hence, every separate class can use these parameters.

4.2.3 The MEB Module

The problem of computing the MEB of a given vector set arises as a subproblem in our problems. If the initial control is satisfied, we call the MEB algorithm either once or never. Otherwise, we call the MEB algorithm for *numberOfClusters* times during the initial upper bound computation and *numberOfNonemptyClusters* times for each node in the branch-and-bound tree. Hence, using an efficient algorithm for computing the MEB is one of the key aspects of our algorithms. As a result, the MEB module is the fundamental part of our software and the class related to this module is called the Algorithm class.

There are four separate and theoretically efficient algorithms in the literature

(Yıldırım [46]; Ahipaşaoğlu and Yıldırım [3]) for computing the MEB. These algorithms are also efficient in practice in terms of the time and space complexities. We implement these algorithms in the Algorithm class. Each of these algorithms is iterative in nature and tries to find the optimal center by moving the center at each iteration. They differ in terms of the possible movements of the center at each iteration.

We can summarize the differences of the algorithms as follows:

- *Meb_u*: The center of the cluster is moved towards the furthest vector in each step. (Yıldırım [46])
- *Meb_u_elim*: The center of the cluster is moved towards the furthest vector and potential vectors that are inside the interior of the MEB are removed from the vector set in each step. (Ahipaşaoğlu and Yıldırım [3])
- *Meb_u_away*: The Center of the cluster is moved towards the furthest vector or away from the closest vector in each step. (Yıldırım [46])
- *Meb_u_away_elim*: The center of the cluster is moved towards the furthest vector or away from the closest vector and potential vectors that can be inside the cluster are removed from the vector set in each step. (Ahipaşaoğlu and Yıldırım [3])

The Algorithm class takes n, m, tolerance and S as parameters.

4.2.4 The Initial Control Module

We identify two special cases of our problems in consideration. These cases are as follows;

- $k = 1$
- $|S| \leq k$.

We handle these special cases separately and efficiently in this module. The class takes m, n, tolerance, S, k and the algorithm type as parameters. The algorithm of the module is as follows:

Algorithm 3: Algorithm of the Initial Control Module

```
1 begin
2     if k = 1
3         Compute the MEB with the selected algorithm type;
4     end if
5     else if |S| ≤ k
6         Assign each vector to a separate cluster;
7         Assign 0 to radius of each cluster;
8     end else if
9
```

In the above cases, we can easily compute an optimal solution. Therefore, we do not need to do any further operations such as computing the initial upper bound or constructing the branch-and-bound tree.

4.2.5 The Initial Upper Bound Module

As mentioned before, we also concentrate on providing an efficient algorithm for computing an initial upper bound. This algorithm is implemented via the upper bound class. The class takes n, m, tolerance, S and the problem type as parameters. We can summarize the algorithm as follows:

Algorithm 4: Algorithm of the Initial Upper Bound Module

```
1  begin
2      Select an arbitrary vector;
3      Compute the furthest vector from the arbitrary vector and determine it as
4      the center of the first cluster;
5      Compute the furthest vector from the center of the first cluster and
6      determine it as the center of the second cluster;
7      if k ≥ 3
8          For i = 3 to k
9              Compute the distances among all unassigned vectors and cluster
10             centers;
11             Select the furthest vector from the respective nearest cluster center as
12             the center of the cluster i;
13         end for
14     end if
15     Assign each unassigned vector to the cluster with the nearest center;
16     Compute solution;
17     Keep this solution as the best solution obtained so far;
18
```

This approximate solution provides an initial upper bound value for the branch-and-bround algorithm.

4.2.6 The Tree Traversal Module

We use the branch-and-bound algorithm for finding optimal solutions to our optimization problems. It is a systematic enumeration method of all candidate solutions while discarding a large number of candidates by using upper and lower bounds. We implement our branch-and-bound algorithm via the search class.

We use a branch-and-bound tree for representing our algorithm. The branch-and-bound tree has a hierarchical structure with a set of linked nodes. Each node contains same data sructures and can have zero or more child nodes. The root node is the root of the tree. A parent node is a node that has at least one child. Moreover, each node has exactly one parent except the root node. Nodes that do not have any child nodes are called leaf nodes. The height of a node corresponds to the length of a path from that node to the deepest node of the tree reachable

from that node. Hence, the height of the tree corresponds to the height of the root node. Conversely, the level of a node is the length of a path from that node to the root node. Therefore, the level of each node is one more than the level of its parent node. An active node is a node that is created but not examined yet.

An arbitrary vector is assigned to the first cluster in the root node without loss of generality. We try to circumvent the symmetry of clusters in nodes using this approach. If we do not assign any nodes to any clusters in the root node, then there may exist some nodes in the tree that have potentially symmetric clusters. For example, let us given an instance such that $k = 3$. The following figure, Figure 4.2, represents the case in which we do not assign any vector to any cluster in the root node.

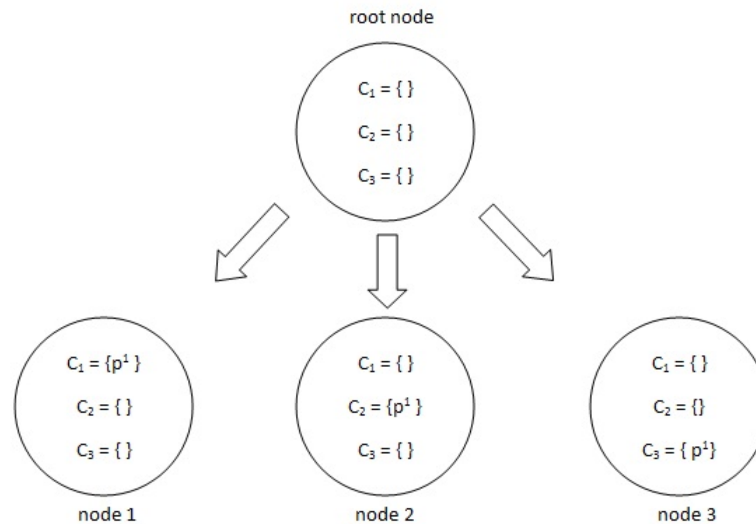

Figure 4.2: The Branch-and-Bound Tree (No Vector Assignment)

Notice that the clusters in node 1, node 2, and node 3 in Figure 4.2 are symmetric. Hence, all nodes give the same lower bound value, and we do not have to compute the lower bound value for each one of these nodes. On the other hand, if we assign a vector to an arbitrary cluster in the root node, the tree will be as follows:

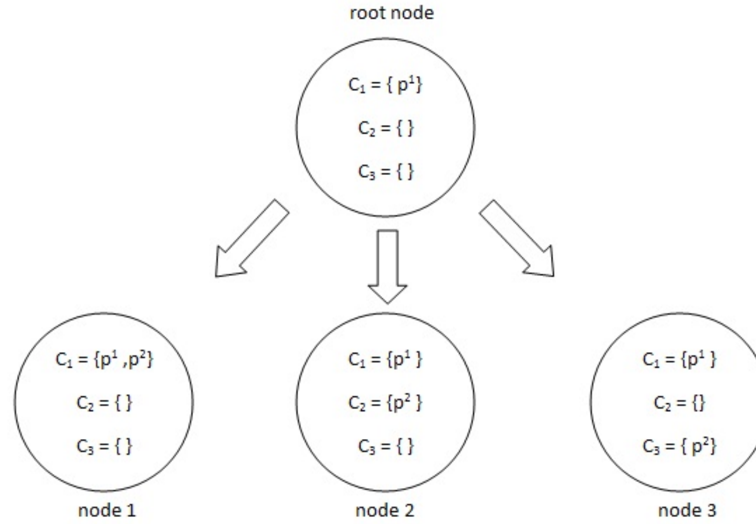

Figure 4.3: The Branch-and-Bound Tree (With Vector Assignment)

Therefore, the symmetry is tried to be broken with this approach. However, there is another potential source of symmetry if we create k child nodes for each node other than leaf nodes in the tree. We can see that the clusters in node 2, and node 3 in Figure 4.3 are still symmetric. Thus, we do not have to examine each of these nodes. In order to prevent symmetry, we create min{*numberOfNonemptyClusters + 1*, k} child nodes for each node other than the leaf nodes in the tree. Let us recall the above instance again. Suppose again that, the first vector is assigned to the first cluster in the root node, and we determine the second vector as the new entry vector for child nodes of the root node. Then, if the second vector is assigned to each child node, the tree will be constructed as in Figure 4.3. On the other hand, if we assign the second vector to each min{*numberOfNonemptyClusters + 1*, k} nodes, the tree will be as follows:

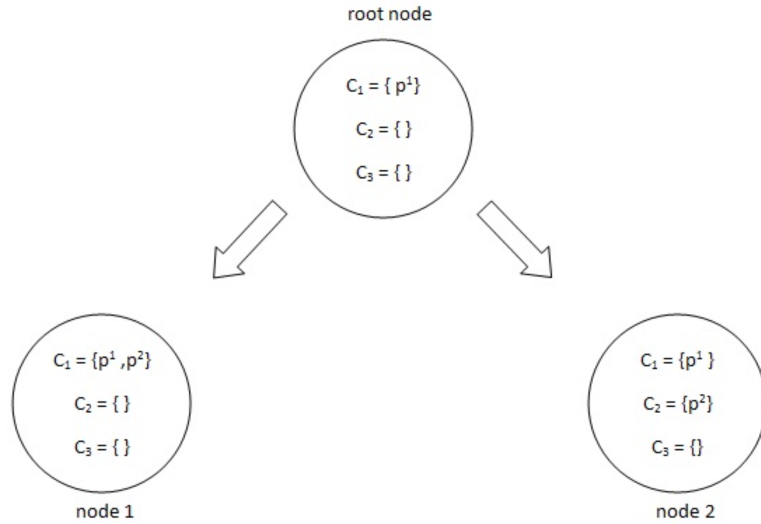

Figure 4.4: The Branch-and-Bound Tree (With min{*numberOfNonemptyClusters + 1*, *k*} Child Nodes)

Hence, we may break the symmetry of clusters in the nodes of the tree through the following reasons. The first vector is always assigned to the first cluster. Therefore, it can not be assigned to any other cluster for other nodes of the tree, and the first cluster of any node can not be appeared in any other clusters of the other nodes. Furthermore, each cluster in each descendant node of a given node is a superset of the corresponding cluster in the ancestor node. As a result, the cluster combination in a specific node can not be appeared in any other node. We aim to decrease the number of examined nodes, which is crucial for our algorithm, in the branch-and-bound tree using these two approaches.

In addition, we use the heap data structure to represent our tree. Heap is a kind of tree based data structure. It satisfies the following heap property. For instance, if Y is a child node of X, then $key(X) \geq key(Y)$ or $key(X) \leq key(Y)$. The former case is called a max-heap and the latter case is called a min-heap. Max-heap implies that the root node is an element with the greatest key whereas min-heap implies that the root node is an element with the smallest key.

In our tree structure, each node has its own key. The root node always has the smallest key, and all other nodes will always have greater keys. Therefore,

our tree-based data structure is a min-heap data structure.

One important question in implementing such an alogirthm is the way the tree is stored. We do not need to store the whole tree. It is enough to store only the active nodes (subproblems) that still need to be examined. Hence, a new question arises. "How much information do we need to keep for each active node?". We choose to keep the minimum required information for each active node while preventing the repetitions of certain calculations in each node.

The process of visiting (examining) each node in a tree data structure, called tree traversal, has a crucial role in the solution time of the problem. Trees are non-linear data structures that can be traversed in many ways. We implement five tree traversal methods. These are depth-first search (DFS), breadth-first search (BFS), best-first search (BEST), random-first search (RFS) and hybrid-search (HS). All of the methods have their own advantages and disadvantages.

In the DFS algorithm, we aim to find a good initial lower bound in order to prune the tree significantly. Hence, we descend as quickly as possible to find a good feasible solution. We traverse the root, the left sub-tree, and the right sub-tree respectively. With the DFS algorithm, the number of active nodes can be at most the height of the tree. Therefore, the DFS algorithm is efficient in terms of the worst case space complexity, whereas it may not be efficient in terms of the worst case time complexity since the optimal solution may be the rightmost leaf node of the tree. It is a practical tool in cases where you want to choose one possible solution through many different solutions.

The BFS algorithm is based on traversing the tree in level-order. One visits every node on the same level before going to a lower level in the tree. Its worst case time complexity is the same as the DFS algorithm. Moreover, it is not efficient in terms of the worst case space complexity since it examines the tree level by level and there may exist many active nodes during the tree traversal. One can use this algorithm when interested in all possible best solutions.

One can attempt to minimize the total number of nodes examined in the tree by choosing the active node with the best (smallest) upper bound. The tree

is traversed by exploring the most promising node chosen. Hence, we continue with better partial groupings in the BEST algorithm. Although this heuristic is efficient in terms of time complexity since it may decrease the total number of examined nodes, it is not efficient in terms of space complexity since there may exist many active nodes during the tree traversal.

Nodes are examined in a random manner in the RFS algorithm. The worst case time and space complexities of the method are the same as the BFS algorithm. We cannot predict the solution time of the method since it may use both the best and the worst paths to the optimal solution.

The DFS and the BEST algorithms are merged in order to develop a stronger tree traversal method. We call this method the HS algorithm. We aim to include the stronger parts of both algorithms and exclude the weaker parts of each of them. As mentioned before, the DFS algorithm is an efficient algorithm in terms of space complexity, and the BEST algorithm is a type of heuristic that tries to examine paths that are closer to an optimal solution. However, the BEST algorithm is not efficient in terms of space complexity. Hence, we adopt a compromise betweeen these two ideas.

The HS algorithm relies on switching between the DFS and the BEST algorithms when necessary. Our initial incentive was to start with the DFS algorithm to find an initial feasible solution. However, since we can find an initial feasible solution with the approximation algorithm as well, we concluded that using the DFS algorithm at start is not necessary. As a result, we start with the BEST algorithm which allows us to minimize the total number of examined nodes. Starting with this algorithm, however, may lead to memory problems especially for large-scale instances. For this reason, when a pre-defined upper memory limit is reached, we switch to the DFS algorithm. This allows us to overcome the memory problem and also to find potential better upper bounds. If we have not reached an optimal solution up to that point, we need to switch to the BEST algorithm again when the predefined lower memory limit is reached. This again allows us to minimize the total number of examined nodes and also to continue with better partial groupings. Switching between the two algorithms continues

until an optimal solution is found. With the HS algorithm, we not only have an efficient tree traversal algorithm but we also decrease the possibility of facing potential memory problems during the tree trversal.

The predefined upper and the predefined lower memory limits must be determined carefully. We defined specific selection criteria during the determination of the memory limits. To begin with, switching between the tree traversal algorithms has a time-cost that should be taken into consideration. Moreover, as mentioned before the number of active nodes can be at most the height of the tree in the DFS algorithm. Therefore, we can set a high upper memory limit. In addition, during the DFS algorithm the number of the active nodes decrease in the course of the time, while it increases during the BEST algorithm. Hence, lower memory limit must not be set very low. As a result, we define the upper memory limit as the %70 of the total memory and the lower bound limit as the %50 of the total memory.

We determine the keys of the nodes in the branch-and-bound tree according to the chosen tree traversal algorithm type. These keys are sorted in either ascending or descending order. Then, the node with the smallest or the largest key is examined according to the ordering criteria. The order of the node is called the priority of the node.

The search class takes m, n, tolerance, S, the algorithm type, the problem type, the search method, k, upper bound and best clusters as parameters. We can summarize the algorithm as follows:

Algorithm 5: Algorithm of the Tree Traversal Module

```
1  begin
2      Create the root node;
3      Perform node operations;
4      Create children nodes of the root node;
5      Add required information of the root node to the heap;
6      Delete the root node;
7      While There are nodes to examine
8          Select a node to examine (current node);
9          Perform node operations of the current node;
10         if Lower bound of the current node exceeds upper bound
11             Prune the current node;
12             Delete the current node;
13             Go to step 7;
14         end if
15         Create children nodes of the current node;
16         For i = 1 to numberOfChildrenOfTheCurrentNode
17             Perform node operations;
18             if The node is leaf
19                 Determine it as the leaf node;
20                 Do required updates;
21             end if
22             else
23                 if Lower bound of the node exceeds upper bound
24                     Prune the node;
25                 end if
26                 else
27                     Do required updates;
28                     Add required information of the node to the heap;
29                 end else
30             end else
31             Delete the child node;
32         end for
33     end while
34
```

We use additional advanced data structures in this module. For the DFS, BFS and BEST algorithms we use priority queues for storing the tree. It is similar to the heap structure, in which only the max heap element can be retrieved. The elements in priority queue are ordered according to a predefined ordering criterion. This criterion is different for each algorithm. In addition, we use vectors for storing the tree in the RFS and HS algorithms. The ordering criterion of the heap differs for both of the algorithms.

4.2.7 The Node Module

As mentioned before, each node in the branch-and-bound tree corresponds to a partial clustering obtained so far. During the tree traversal, we examine these nodes in order to find an optimal solution. We use the same processes for each node of the tree where all node operations are performed via the node class.

To begin with, we determine the amount of information we should keep for each node of the tree. We have two options. We can either keep a minimum amount of information, or we can keep all of the available information. In the former case, we have to repeat all certain calculations for each node, whereas we need a large amount of memory to keep all the available information in the latter case. Hence, we adopt a compromise between these two ideas.

For each node we determine the sufficient amount of information to store that will prevent any calculation repetitions in its descendant nodes. Therefore, we see that it is enough to keep the following parameters:

- Integer type parameters
  - Number of dimensions, number of points, the algorithm type, the problem type, number of clusters, level of the node, next entry
- Double type parameters
  - Tolerance, order of the node, radius of the node

- Vector of vectors
  - Indices of vectors in each cluster
- List of lists
  - Distances among unassigned points and cluster centers
- Vector
  - Radii of the clusters
- List
  - Indices of vectors that are not assigned to any clusters

We keep these values for each active node. When we delete a node, we delete all these parameters. Moreover, we use the most convenient data structures for keeping these values in order to save memory. We initially compute these values in the root node. Then, we transfer these values to the children nodes from the parent nodes. Therefore, we do not need to repeat the same calculations for each node. We use copy constructors for transferring information from parent nodes to children nodes. Copy constructors are used for creating a new object as a copy of an existing object. In addition, we do some further computations in children nodes such as computing the distances among the unassigned points and the center of the new constructed cluster, computing the new entry vector, computing the new constructed sphere, updating the indices of vectors in clusters and radii of clusters.

The class takes m, n, tolerance, S, order, unassigned points list, the algorithm type, the problem type, k, current clusters, distances of unassigned points to cluster centers as parameters.

We develop an efficient and modular algorithm for the Node module and we apply this algorithm to each of the nodes. Suppose we add a new entry vector (p^j) to cluster i (C_i) for each node where c^i corresponds to the center of the ith cluster: The algorithm then proceeds as follows;

Algorithm 6: Algorithm of the Node Module

```
1  begin
2      Select entering vector (p^j);
3          Random vector for the root node;
4          Vector inherited from the parent node for the other nodes;
5      Clear distances among all unassigned vectors and c^i;
6      Assign p^j to C_i;
7      Remove p^j from the unassigned vectors;
8      For Nonempty clusters
9          Clear distances among p^j and center of the corresponding cluster;
10     end for
11     Compute MEB for C_i;
12     Update radius value of C_i;
13     Compute distances among the unassigned points and c^i;
14     If There exist any vector inside C_i
15         Remove the vector from the unassigned vectors;
16         Add vector to C_i;
17         Clear the distances among the vector and centers of nonempty clusters;
18     end if
19     Compute the distances among all unassigned points and c^i;
20     Find the closest vector to each of the cluster center;
21     Determine the vector that is furthest from the respective nearest c^i
22     as the new entry vector
23
```

4.2.8 The Output Module

Output corresponds to the information that is produced by the computer programs and received by the users. Hence, outputs can vary from software to software. In our software, we output our results via the output class. The class provides users with the following capabilities:

- View all the results from the output screen.
  - Optimal solution
  - Initial upper bound
  - Optimal clustering
  - Number of reached leaves in the branch-and-bound tree

  - Number of examined nodes in the branch-and-bound tree
  - Number of prunes in the branch-and-bound tree
  - Maximum number of active nodes in the branch-and-bound tree
  - Number of vectors in clusters
  - RAM usage
  - Virtual Memory Usage
  - Running time

- Save all the results to the files.
- Terminate or restart the program.

We have mentioned the importance of the exception handling in previous sections. We develop a detailed exception handling for the output class like the menu class.

Chapter 5

Computational Results

In this chapter, we report the results of our extensive computational experiments. The computational experiments were carried out on a Pentium (R) Dual-Core CPU E5200 processor with a clock speed of 2.50 GHz and 4 GB RAM running under Linux OS version Ubuntu 9.04. The algorithms were implemented, executed and run in the `C++` environment using the Gcc version 4.3.3. (For this Chapter, number of points corresponds to the number of vectors (m)).

5.1 Computational Setup

We initially tested the efficiency of the proposed branch-and-bound algorithms for the aforementioned $min - max$ and $min - sum$ problems using $k = 2$ and $k = 3$.

For each value of k, different choices of n and m were used in order to assess the performance of the algorithms with respect to the sizes of the problems. The first data set was generated for $k = 2$ with sizes (n, m) chosen from all possible choices with $n \in \{25, 50, 100\}$ and $m \in \{100, 500, 1000\}$ and the second data set was generated for $k = 3$ with sizes (n, m) chosen from all possible choices with $n \in \{10, 25, 50\}$ and $m \in \{48, 96, 144\}$. Note that, n is doubled, and m is either

doubled or quintuplicated in both data sets. Therefore, the effect of doubling the number of dimensions and/or doubling or quintuplicating the number of vectors on both the hardness of the problems and the efficiency of the proposed algorithms could be measured with the above choices of n and m. In addition, we could also test the effect of k on both problems with the approximate sized instances where $n = 50$, $m = 100$ for $k = 2$ and $n = 50$, $m = 96$ for $k = 3$ with all the other parameters fixed. The accuracy parameter of the MEB algorithms, ϵ, is set to 10^{-3} for both data sets.

For each fixed value of n and m, two different distributions were used to generate random instances to factor into the effect of the distributions on the problems. These distributions are the uniform spherical distribution and the pseudo-normal spherical distribution. All the input vectors were generated within a sphere using these distributions. The uniform spherical distribution has vectors uniformly distributed in a given sphere where the pseudo-normal spherical distribution has vectors pseudo-normally distributed in a given sphere. The Matlab code developed by J.Burkardt [6] is used for generating these random data vectors.

Then, different radii and cluster types were constructed for each fixed values of k, (n, m) and each fixed distribution. For $k = 2$, ten different radii pairs were constructed and we call these radii pairs simply as radius types. For each radius type, the center of the first sphere lies at the origin and the center of the second sphere lies at the vector of all ones for each dimension of the space. Table 5.1 illustrates these radii types.

For $k = 3$, again, ten different radii pairs were constructed that are also called radius types. For each radius type, the center of the first sphere lies at the origin and the center of the second sphere is $(\sqrt{n}, 0, 0, ..., 0)$ while the center of the third sphere is $(\sqrt{n/2}, \sqrt{3n}/\sqrt{4(n-1)}, \sqrt{3n}/\sqrt{4(n-1)}, ..., \sqrt{3n}/\sqrt{4(n-1)})$. Table 5.2 illustrates these radii types.

Radii of the Spheres		
Radius Type	**First Sphere**	**Second Sphere**
1	$\sqrt{n/4}$	$\sqrt{n/4}$
2	$\sqrt{n/4}$	$\sqrt{n/2}$
3	$\sqrt{n/4}$	$\sqrt{3n/4}$
4	$\sqrt{n/4}$	$\sqrt{n}$
5	$\sqrt{n/2}$	$\sqrt{n/2}$
6	$\sqrt{n/2}$	$\sqrt{3n/4}$
7	$\sqrt{n/2}$	$\sqrt{n}$
8	$\sqrt{3n/4}$	$\sqrt{3n/4}$
9	$\sqrt{3n/4}$	$\sqrt{n}$
10	$\sqrt{n}$	$\sqrt{n}$

Table 5.1: The Radius Types for $k = 2$

Radii of the Spheres			
Radius Type	**First Sphere**	**Second Sphere**	**Third Sphere**
1	$\sqrt{n/3}$	$\sqrt{n/3}$	$\sqrt{n/3}$
2	$\sqrt{n/3}$	$\sqrt{n/3}$	$\sqrt{2n/3}$
3	$\sqrt{n/3}$	$\sqrt{n/3}$	$\sqrt{n}$
4	$\sqrt{n/3}$	$\sqrt{2n/3}$	$\sqrt{2n/3}$
5	$\sqrt{n/3}$	$\sqrt{2n/3}$	$\sqrt{n}$
6	$\sqrt{n/3}$	$\sqrt{n}$	$\sqrt{n}$
7	$\sqrt{2n/3}$	$\sqrt{2n/3}$	$\sqrt{2n/3}$
8	$\sqrt{2n/3}$	$\sqrt{2n/3}$	$\sqrt{n}$
9	$\sqrt{2n/3}$	$\sqrt{n}$	$\sqrt{n}$
10	$\sqrt{n}$	$\sqrt{n}$	$\sqrt{n}$

Table 5.2: The Radius Types for $k = 3$

As seen from both Table 5.1 and Table 5.2, we aimed to measure the efficiency of the proposed algorithms on vector sets chosen from both the disjoint and overlapping spheres with various sizes.

Without loss of generality, the radius of the second sphere is set to be greater than or equal to the radius of the first sphere for each radius type and for each k value. Moreover, for $k = 3$ the radius of the third sphere is again always greater than or equal to each of the radius of the first and the second spheres for each radius type. This allows us to break the symmetry of the spheres for different radius types since the index of the larger sphere is not critical.

Furthermore, the number of vectors in each sphere may have an impact on the efficiency of the proposed algorithms. Hence, three different cluster pairs were generated, with respect to the number of vectors in each sphere, for each fixed radius type for $k = 2$. Table 5.3 illustrates the cluster types for $k = 2$.

Number of Vectors in Cluster Types		
Cluster Type	**First Cluster**	**Second Cluster**
1	m/4	3m/4
2	m/2	m/2
3	3m/4	m/4

Table 5.3: The Cluster Types for all Radius Types for $k = 2$

In additon, for $k = 3$ different number of cluster pairs were generated for each fixed radius type regarding the number of vectors in each sphere. In radius types 1, 7 and 10, it is sufficient to generate only two cluster types because the radii of the spheres are identical. Table 5.4 illustrates the number of vectors in each spheres for these radius types.

Number of Vectors in Cluster Types			
Cluster Type	**First Cluster**	**Second Cluster**	**Third Cluster**
1	m/3	m/3	m/3
2	m/2	m/3	m/6

Table 5.4: The Cluster Types for Radius Types 1, 7, 10 for $k = 3$

There are four cluster types in each of the radius types 2, 3, 4, 6, 8 and 9 since two spheres have identical radii and the symetric spheres that arise as a result of the identical radii can be eliminated. The number of vectors in each cluster is illustrated in Table 5.5.

Number of Vectors in Cluster Types			
Cluster Type	**First Cluster**	**Second Cluster**	**Third Cluster**
1	m/3	m/3	m/3
2	m/6	m/3	m/2
3	m/3	m/6	m/2
4	m/3	m/2	m/6

Table 5.5: The Cluster Types for Radius Types 2, 3, 4, 6, 8, 9 for $k = 3$

One needs to generate seven cluster types in radius type 5, which are illustrated in Table 5.6, since each of the spheres has a distinct radius.

Number of Vectors in Cluster Types			
Cluster Type	**First Cluster**	**Second Cluster**	**Third Cluster**
1	m/6	m/3	m/2
2	m/6	m/2	m/3
3	m/3	m/2	m/6
4	m/3	m/6	m/2
5	m/3	m/3	m/3
6	m/2	m/3	m/6
7	m/2	m/6	m/3

Table 5.6: The Cluster Types for Radius Type 5 for $k = 3$

In addition to all, for each fixed k, (n, m), radius type, cluster type, and the distribution; five different problem instances were generated. The computational results are reported in terms of averages over these instances. This implies that there is a total of 2700 instances for $k = 2$. We run each of these instances four times to test the efficiency of the each tree traversal algorithms (the BEST and the DFS) along with the each MEB algorithms (*meb_u_away* and *meb_u_away_elim*). Furthermore, the upper memory limit for the BEST algorithm was reached for 100 of these instances. Therefore, these instances were rerun using the HS algorithm together with the *meb_u_away* and the *meb_u_away_elim* algorithms. Furthermore, there is a total of 3330 instances for $k = 3$. We run each of these instances once with the HS algorithm along with the *meb_u_away_elim* algorithm. As a result, we took 14330 runs in the scope of this thesis.

5.2 Algorithmic Setup

We designed and implemented five different tree traversal algorithms in the scope of this thesis. As mentioned before, each algorithm has its own advantages and disadvantages. We have already tested the efficiency of the BFS, the DFS, the RFS and the BEST algorithms along with all the MEB algorithms on considerably smaller scale instances in practice in two of our previous works (TUBITAK Project Report (2008) and the national YA/EM conference (2009)).

The results of the previous experiments revealed that the BFS and the RFS algorithms are inefficient in practice, as expected since even small-scaled instances

of the problems cannot be solved within a reasonable amount of time. The BFS algorithm faced memory problems in general and the RFS algorithm usually gave unpredictable performance. Therefore, we did not use the BFS and the RFS algorithms in the scope of this study.

For $k = 2$, we initially solved our problems with both the DFS and the BEST algorithms. Note that, as mentioned before the BEST and the HS algorithms exhibit the same performance if the memory usage does not exceed the predefined upper memory limit for a specific instance. Therefore, the HS algorithm was used for the instances in which the memory usage reached the upper memory limit.

We analyzed the results and observed that the HS algorithm works better than both the BEST and the DFS algorithms for $k = 2$ in terms of the running time. Moreover, this algorithm did not come across with any space problems since it switches to the DFS algorithm whenever a predefined upper memory limit is reached. These results will be presented in the following section. Therefore, while it was stated precisely that the HS algorithm is the most efficient tree traversal algorithm in practice in terms of the running time, we continued to solve the instances of $k = 3$ with only the HS algorithm.

The problem of computing the MEB of a given vector set arises as a subproblem in both of the aforementioned problems. In this study, we implemented four separate and theoretically efficient algorithms for computing the MEB that are also efficient in practice in terms of both time and memory complexities. However, we also tested their efficiency in practice in the studies mentioned above (TUBITAK Project, YA/EM Conference). As stated theoratically in the literature the algorithms in both algorithm pairs, *meb_u* and *meb_u_elim*, and *meb_u_away* and *meb_u_away_elim*, gave the same solutions to the same problems with different solution times. Furthermore, the solutions generated from both pairs and the memory usage of them do not differ much. On the other hand, the latter pair performed better than the former pair in terms of the solution time. As a result, we did not choose to use the *meb_u* and the *meb_u_elim* algorithms in the scope of this thesis.

For $k = 2$, we used the *meb_u_away* and the *meb_u_away_elim* algorithms

for computing the MEB of a given vector set. The results revealed, as in the literature, that the latter algorithm performed better than the former algorithm in 99% of the instances in terms of the running time with negligible difference in memory requirements. Therefore, we used only the *meb_u_away_elim* algorithm for $k = 3$ case.

5.3 Experimental Results

The software package outputs various results to the users. These results are the optimal solution of the problem, the initial upper bound, the optimal clusters, the number of reached leaves, the number of examined nodes, the number of prunes and the maximum number of active nodes in the branch-and-bound tree, the number of vectors in clusters, the maximum RAM usage, the maximum virtual memory usage and the running time.

The number of examined nodes in the branch-and-bound tree can be used to assess the hardness of the problems, while the running time corresponds to the speed of the algorithms. Therefore, these outputs are selected as the specific criteria for measuring the efficiency of the proposed branch-and-bound algorithms in practice. There is a relationship among these criteria. Note that, the running time of the proposed algorithms increases as the number of examined nodes in the branch-and-bound tree increases. However, the running time of the algorithms depends also on the sizes of the subproblems that are solved in the nodes of the tree. Therefore, examining fewer nodes in the branch-and-bound tree does not always imply that the algorithm finds a solution in a shorter amount of time.

However, other results generated by the software package such as the initial upper bound, the optimal clusters, the number of reached leaves, the number of prunes, the number of vectors in clusters, the maximum RAM usage, the maximum virtual memory usage were not be defined among the specific criteria mentioned above due to some significant reasons. First, the tree traversal algorithm type significantly effects the number of reached leaves and the maxiumu number

of active nodes along with the number of examined nodes in the branch-and-bound tree. Notice that, if one uses the DFS algorithm, the number of reached leaves in the branch-and-bound tree increases, while the maximum number of active nodes in the branch-and-bound tree decreases due to the structure of the algorithm. Conversely, if the BEST algorithm is used, then the maximum number of active nodes in the branch-and-bound tree increases, while the number of reached leaves in the branch-and-bound tree decreases. In addition, if the number of examined nodes in the branch-and-bound tree increases, then the number of potential prunes and the number of potential leaves in the branch-and-bound tree may also increase. Therefore, these result may be foreseen beforehand since they depend mostly on the tree traversal algorithm types. Apart from these the maximum number of active nodes in the branch-and-bound tree directly affects the maximum RAM and virtual memory usages of the algorithms along with the size of the problems. In other words, the maximum RAM and virtual memory usages may also be foreseen in advance. Therefore, we chose to use the memory usages of the tree traversal algorithms only for comparing them in terms of the memory complexity.

The computational results are reported in Tables A.1 through A.4. However, we could not include the results of all computational experiments in these tables due to space constraints. Therefore, we include the part of the results that represents the general structure of the computational results in Appendix A.

The results for $k = 2$ are illustrated in Tables A.1 through A.2. We fixed some parameters in these results in order to give the general structure of the computational results. To begin with, the problem type is fixed to the $min - max$ problem because it is harder than the $min - sum$ problem for this data set. Moreover, the tree traversal algorithm algorithm was fixed to the BEST algorithm and the MEB algorithm is fixed to the *meb_u_away_elim* algorithm while this combination of algorithms seems to be the most efficient algorithm combination for solving the problems. As a result, these tables illustrate the efficiency of the proposed algorithms on the harder problems with the best tree traveral and minimum enclosing ball algorithms. For $k = 2$, Table A.1 illustrates the results of the number of examined nodes in the branch-and-bound tree for each radius and

cluster type and Table A.2 gives the results of the running time of the algorithms again for each radius and cluster type.

The results for $k = 3$ are given in Tables A.3 through A.4. The problem type is again fixed to the $min - max$ problem as in the above data set. Moreover, the tree traversal and the MEB algorithms were fixed to the HS and the *meb_u_away_elim* algorithms, respectively, while only the HS algorithm was used as the tree traversal algorithm and the *meb_u_away_elim* algorithm was used as the MEB algorithm for this case. Table A.3 illustrate the results of number of examined nodes in the branch-and-bound tree for each radius type and each cluster type. Table A.4 corresponds to the results of the running time of the problems again for each radius and cluster type.

All of the tables are divided into four columns and four rows. The rows corrspond to the number of vectors while the columns present the number of dimensions of the space. The values in the tables give the number of examined nodes in the branch-and-bound tree or the running time of the algorithm. The empty cells correspond to the instances that were not solved to optimally with the algorithms due to time or space constraints.

5.4 Discussions

This section summarizes the results of the computational experiments. We initially present the effects each one of the parameters on the efficiency of the proposed branch-and-bound algorithms for $k = 2$ and $k = 3$, respectively. Then, an overall discussion will be given.

5.4.1 The Effect of the Problem Type

We studied two different problems in the scope of this thesis, namely the $min - max$ problem and the $min - sum$ problems. We discuss the performance of the proposed branch-and-bound algorithms on both problems in this section.

As mentioned before, we generated random data sets either from disjoint or overlapping spheres with various sizes. Both problems are very similar to each other for radius type 1 and for each value of k in which the data vectors were generated from the disjoint spheres that are far from each other. For this case, the disjoint clusters that the input data sets were generated are simply the optimal solutions of each of the problems. However, an unexpected and interesting result was appeared for other radius types of the $min - sum$ problem for this data set. For $k = 2$, the algorithm assigned exactly one vector to one cluster and assigned the rest of the vectors to the remaining cluster. Similarly for $k = 3$, the algorithm again assigned exactly one vector to each one of the two clusters and assigned the rest of the vectors to the remaining cluster. A cluster with exactly one vector has a radius value of zero, hence this solution seems to have minimized the sum of the radii of the spheres. On the other hand, the algorithm behaved differently for the $min - max$ problem while the objective is to minimize the maximum of the radii of spheres.

In Figure 5.1, we show the relationship between the problem type and the number of examined nodes in the branch-and-bound tree for $k = 2$. The next figure, Figure 5.2, shows the relation between the problem type and the running time in seconds again for $k = 2$. While showing the relations between the stated parameters in Figures 5.1 and 5.2, the radius type 1, cluster type 1, the BEST and the *meb_u_away_elim* algorithms, and the uniform spherical distributions were fixed. Each of the three lines in the figures corresponds to different sizes of data sets for (n, m) with numerical values given by (100, 100), (100, 500) and (100, 1000). The figures illustrate that the number of examined nodes in the branch-and-bound tree and the running time are close to one another for each problem type.

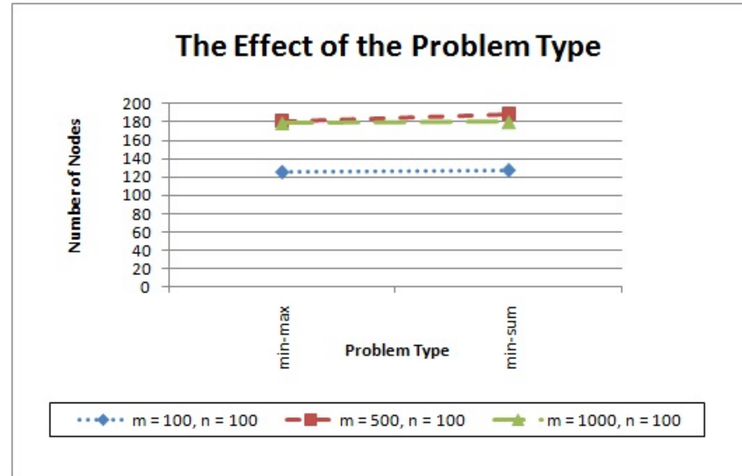

Figure 5.1: The Effect of Problem Type on the Number of Nodes, $k = 2$

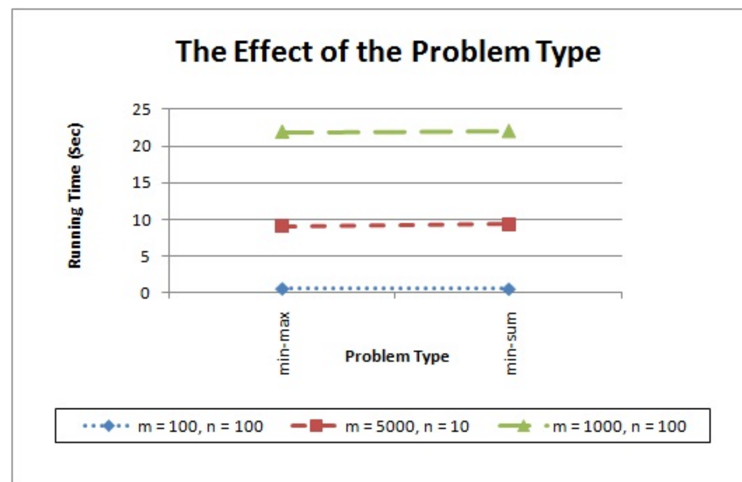

Figure 5.2: The Effect of Problem Type on the Running Time, $k = 2$

Figures 5.3 and 5.4 are organized similarly to Figures 5.1 and 5.2, respectively, showing the pattern of parameters for $k = 3$. In these figures, the radius type 3, cluster type 2, the HS and the *meb_u_away_elim* algorithms and the uniform spherical distribution are fixed. Each line corresponds to a different size of data set for (n, m) with numerical values given by (25, 96), (10, 144) and (25, 144). The number of examined nodes in the branch-and-bound tree and the running time are larger for the $min - max$ problem while the algorithms try to exchange the most suitable vectors between the clusters to minimize the maximum radii of spheres. On the other hand, the $min - sum$ problem tries to find exactly two

vectors that each will be assigned to single clusters in order to minimize the sum of the radii of spheres as stated above.

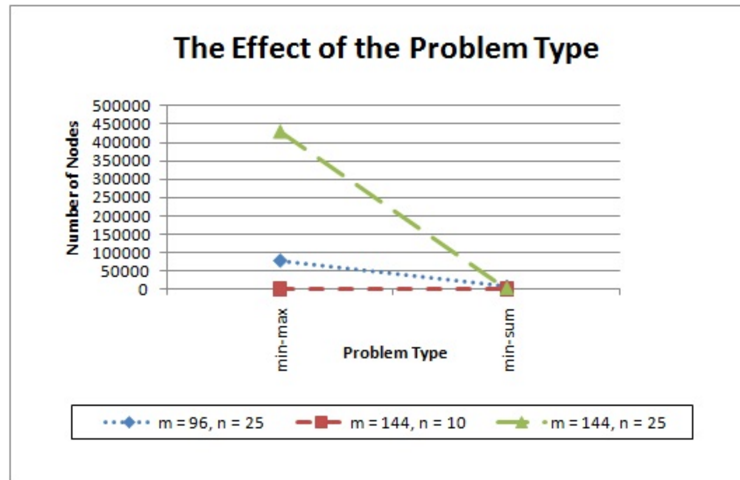

Figure 5.3: The Effect of Problem Type on the Number of Nodes, $k = 3$

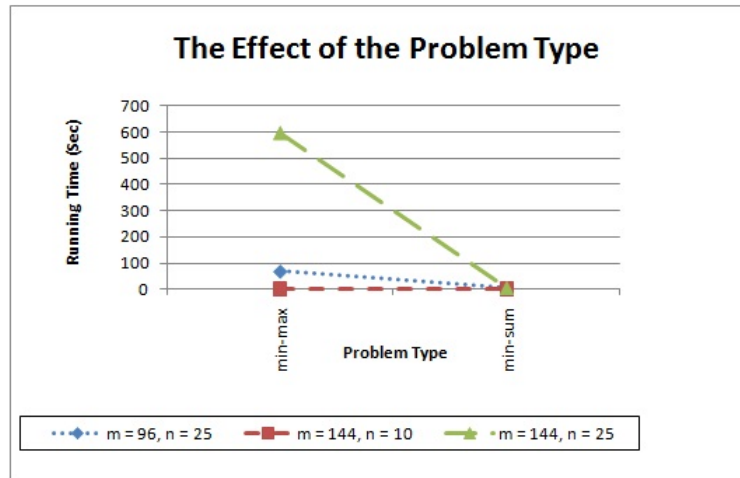

Figure 5.4: The Effect of Problem Type on the Running Time, $k = 3$

As a result, we can say that the algorithms perform similarly while solving the instances that belong to the radius type 1 for both problems and for each value of k. Furthermore, other radius types can be solved easily by the algorithms for the $min - sum$ problem and for each value of k, but the the algorithms get difficult to solve the problems for the radius type other than 1 for the $min - max$ problem. Therefore Figures 5.1 and 5.2 represent the radius type 1 for both problems and for each value of k. In addition, Figures 5.3 and 5.4 represent the radius types other than 1 for the $min - max$ problem for each value of k.

5.4.2 The Effect of the Radius Type

We constructed ten different radii configurations, called radius types, for each value of k. The radius types for $k = 2$ and $k = 3$ are illustrated in Tables 5.1 and 5.2, respectively. We study the effects of these radius types on the efficieny of the proposed algorithms in this section.

The relationship among the radius types and the number of examined nodes in the branch-and-bound tree for $k = 2$, are illustrated in Figure 5.5. The following figure, Figure 5.6, plots the relation between the radius types and the running time for $k = 2$. We fixed the $min-max$ problem, radius type 3, cluster type 1, the BEST and the $meb_u_away_elim$ algorithms and uniform spherical distribution for the illustration of the relations between the stated parameters in Figures 5.5 and 5.6. Each of the three lines in the figures shows different sizes of data sets for (n, m) with numerical values given by (25, 500), (25, 500) and (25, 1000). The figures report that both the number of nodes in the branch-and-bound tree and the running time vary widely with the radius types. It can be seen that the radius types 4, 7, 3, and 9 tend to yield more difficult instances than the other radius types.

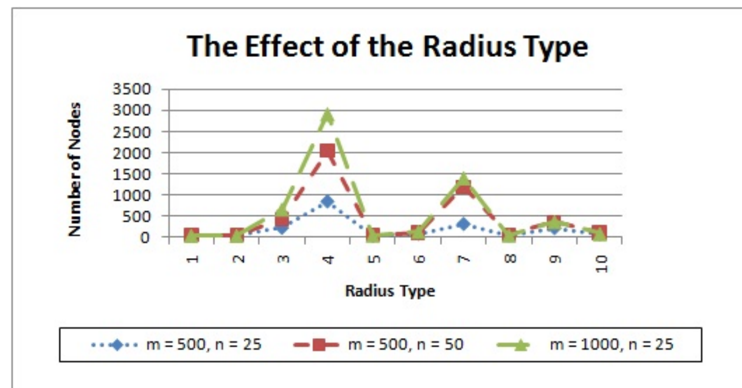

Figure 5.5: The Effect of Radius Type on the Number of Nodes, $k = 2$

The patterns of parameters (number of nodes and the running time) for $k = 3$ are shown in Figures 5.7 and 5.8, which are organized similarly to Figures 5.5 and 5.6, respectively. We used the $min - max$ problem, cluster type 1, the BEST and the $meb_u_away_elim$ algorithms, and the uniform spherical distribution in

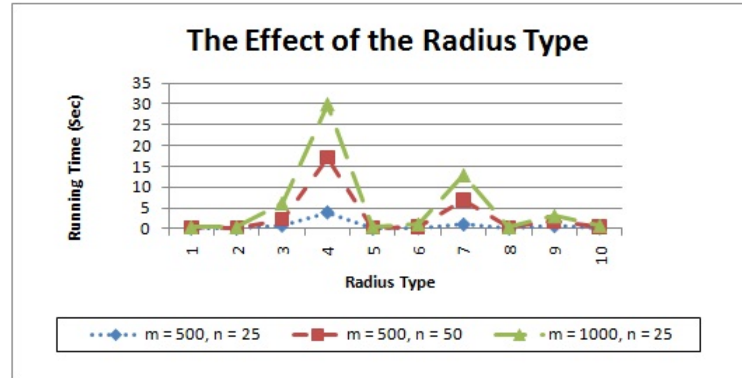

Figure 5.6: The Effect of Radius Type on the Running Time, $k = 2$

these figures. Each line corresponds to a different size of data set for (n, m) given by (50, 48), (10, 96) and (10, 144). We found results similar to those of $k = 2$. The number of nodes and the running time are different for each radius type. Radius type 5, 6, 3, 8, and 9 gave rise to the hardest instances.

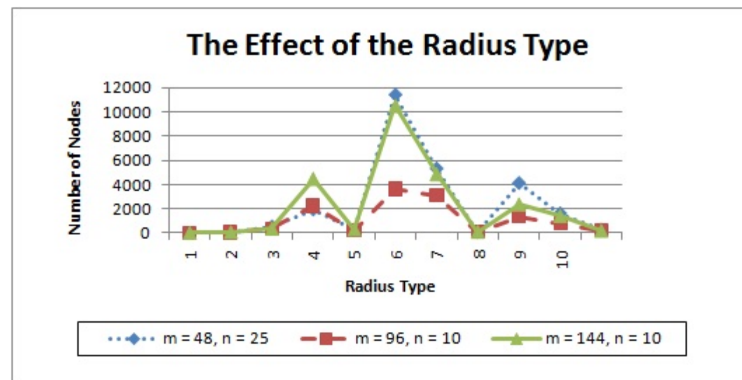

Figure 5.7: The Effect of Radius Type on the Number of Nodes, $k = 3$

We use the $min - max$ problem for showing the results for the radius types, while the $min - sum$ problem does have a small impact on the radius types other than the radius type 1.

Studying these results in detail, we can say that our algorithms can solve instances in which the data sets are generated from the disjoint spheres more efficiently than the instances in which the data sets are generated from overlapping spheres for both of the problems. As mentioned before, the algorithms try to select the cluster centers as far as possible. Therefore, they may eliminate many

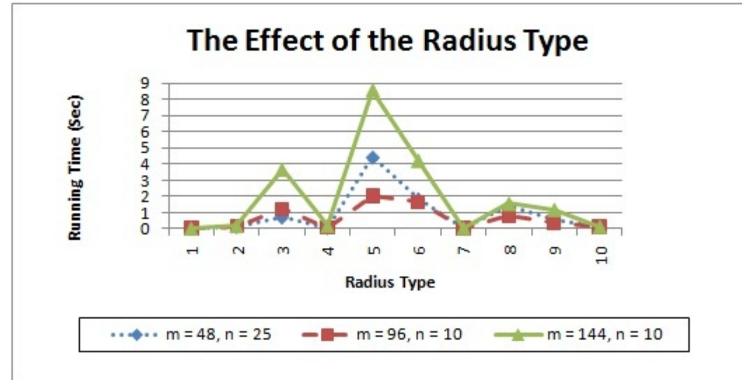

Figure 5.8: The Effect of Radius Type on the Running Time, $k = 3$

vectors since these vectors are identified to be inside the clusters (See Algorithm 6, Step 15). As a result, they might identify the underlying structures of the instances and solve the problems easily.

We can divide the overlapping spheres into two groups with respect to their sizes. The first group includes the spheres that have the same size while the second group consists of spheres with different sizes. If the spheres are not of the same size, then the size of the smaller sphere is forced to be enlarged, and the size of the larger sphere is also forced to be minimized with respect to the objective function of the $min - max$ problem. Therefore, some data vectors must be removed from the larger sphere and assigned to the smaller ones, which may complicate the problem. On the other hand, if the sizes of the spheres are similar, then there will not be as many vector exchanges between the spheres. Hence the instances in the former group might be easier than the instances in the latter group. In addition, as the difference among the sizes of the spheres increase, the vector exchange among the spheres will also increase and this may imply that the problems can become harder.

5.4.3 The Effect of the Cluster Type

As stated before called cluster types were generated with respect to the number of vectors in each sphere, for each radius type. We consider to the effects of these cluster types in this section.

For $k = 2$, Figure 5.9 corresponds to the relationship among the cluster types and the number of examined nodes in the branch-and-bound tree. The next figure, Figure 5.10, shows the relation between the cluster types and the running time for $k = 2$. In both figures the other parameters ($min - max$ problem, radius type 4, the BEST and the $meb_u_away_elim$ algorithms and the uniform spherical distribution) are fixed. Each of the three lines in the figures shows different sizes of data sets for (n, m) with numerical values given by (50, 100), (50, 500) and (50, 1000). The figures show that the cluster type effects both the number of examined nodes and the running time, in which these values are the largest for the first cluster type and the smallest for the third cluster type.

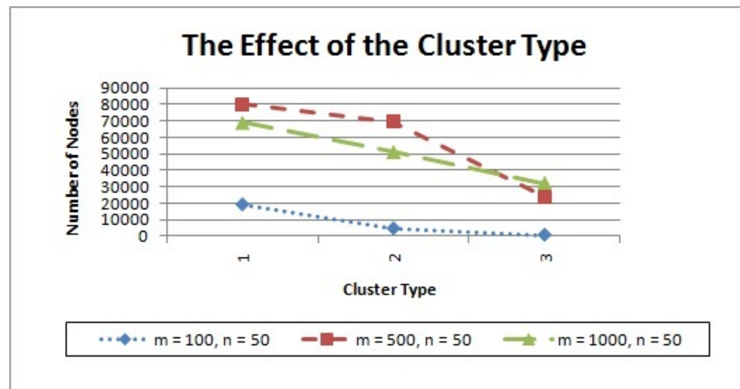

Figure 5.9: The Effect of Cluster Type on the Number of Nodes, $k = 2$

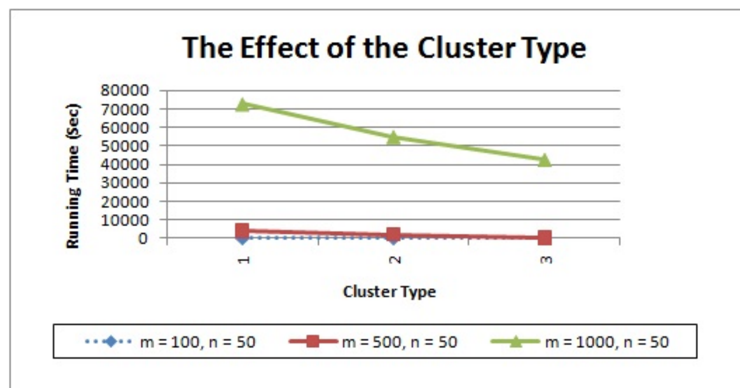

Figure 5.10: The Effect of Cluster Type on the Running Time, $k = 2$

Figures 5.11 and 5.12 are organized similarly to Figures 5.9 and 5.10, respectively, showing the patterns of parameters for $k = 3$. In these figures, the $min - max$ problem, radius type 5, the HS and the $meb_u_away_elim$ algorithms

and the uniform spherical distribution are kept constant. Each line corresponds to a data set with different size for (n, m) given by (25, 48), (10, 96) and (10, 144). As mentioned before, all the radius types have different number of cluster types. However, radius type 5 has 7 cluster types and it includes all the different cluster types. Therefore we included radius type 5. The results reveal that different cluster types have different effects on the algorithms. The number of examined nodes in the branch-and-bound tree and the running time significantly increase with the radius types 1, 2, 4, 5, and 7.

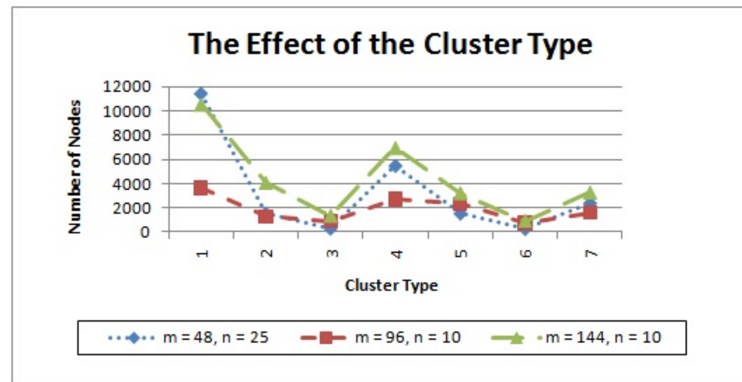

Figure 5.11: The Effect of Cluster Type on the Number of Nodes, $k = 3$

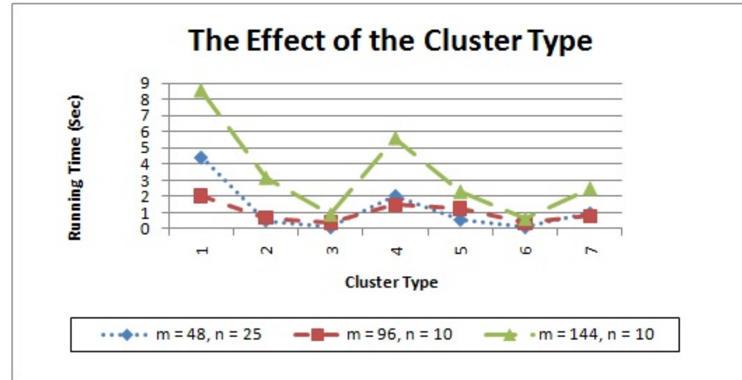

Figure 5.12: The Effect of Cluster Type on the Running Time, $k = 3$

On the one other hand, for the $min - sum$ problem, the cluster types in which the number of vectors in each cluster is the same are harder than the cluster types in which the number of vectors are different in each cluster. This is also reasonable due to the observations given in the Section 5.4.1, because the algorithms try assign m - k + 1 vectors to one cluster and $k - 1$ vectors to the remaining $k - 1$ clusters.

On the other hand, the cluster types that tend to make the problem harder in both cases ($k = 2$ and $k = 3$) have specific properties in common for the $min - sum$ problem. The cluster types in which the largest sphere has more vectors than the smaller ones, usually makes the problem harder. This can be reasonable since, if the number of vectors in the larger sphere is more than the number of vectors in the smaller ones, then the algorithms try to transfer more vectors from the larger sphere to the smaller ones according to the objective function of the $min - sum$ problem. Therefore, this lead to more difficult instances.

5.4.4 The Effect of the Number of Vectors

We also tested the effect of the number of vectors m on the efficiency of the proposed branch-and-bound algorithms. The discussion of these effects are given in this section. As mentioned before, the radius type has a great impact on the efficiency of the branch-and-bound algorithms. Therefore, we initially illustrate the effect of m on the instances with specific radius types that are harder to solve

and then show the effect on those that are easier to solve.

First, we present the results for the radius types that are harder to solve. In Figure 5.13 we plot the relationship between m and the number of examined nodes in the branch-and-bound tree for $k = 2$. The next figure, Figure 5.14, presents the relation between m and the running time in seconds again for $k = 2$. We used the $min-sum$ problem, the BEST and the $meb_u_away_elim$ algorithms and $n = 25$ in these figures. Each of the three lines in the figures shows different data sets for (radius type, cluster type) with numerical values given by (4, 2), (4, 3), (7, 1) respectively. We can easily deduct from the figures that an increase in the number of vectors has significant effect on both the number of examined nodes in the branch-and-bound tree and on the running time for the data sets with specific radius types that are harder to solve. It can be seen that, increasing m by a factor of 5 can increase the number of examined nodes in the branch-and-bound tree by a factor of 100 and also can increase the running time by a factor of 1000.

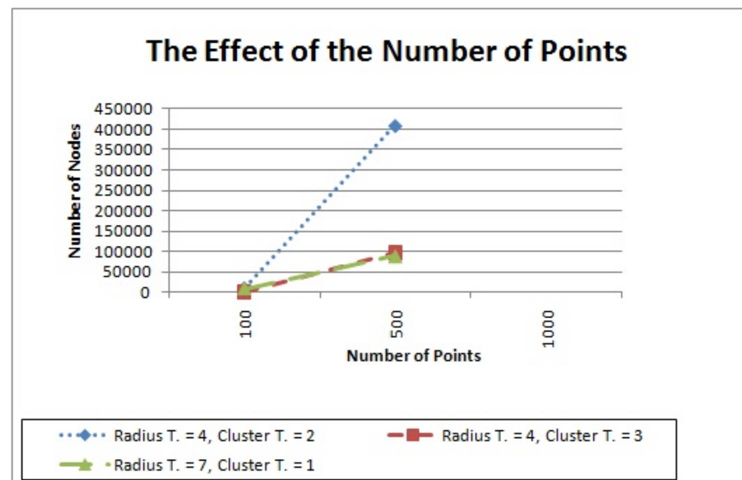

Figure 5.13: The Effect of Number of Vectors on the Number of Nodes, $k = 2$

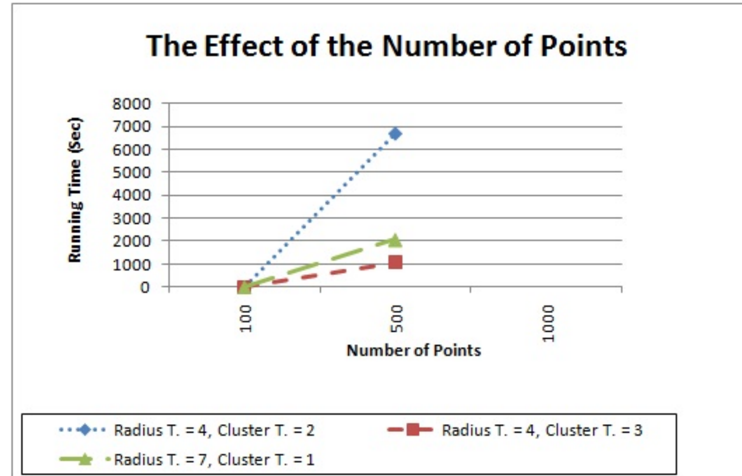

Figure 5.14: The Effect of Number of Vectors on the Running Time, $k = 2$

Next, the results for the radius types that are easier to solve are given. The relationship between m and the number of examined nodes in the branch-and-bound tree for $k = 2$ is illustrated in Figure 5.15. We also show the relation between m and the running time in seconds again for $k = 2$ in the next figure, Figure 5.16. We fixed the $min-max$ problem, the DFS and the $meb_u_away_elim$ algorithms and the uniform spherical distribution in Figures 5.15 and 5.16. Each of the three lines in the figures corresponds to different data sets for (radius type, cluster type) with numerical values given by (1, 1), (2, 1), (5, 1) respectively. The figures illustrate that an increase in m (i.e doubling or increasing the number of vectors by a factor of 5) has a neglible effect on both the number of examined nodes in the branch-and-bound tree and on the running time for the data sets with specific radius types that are easier to solve.

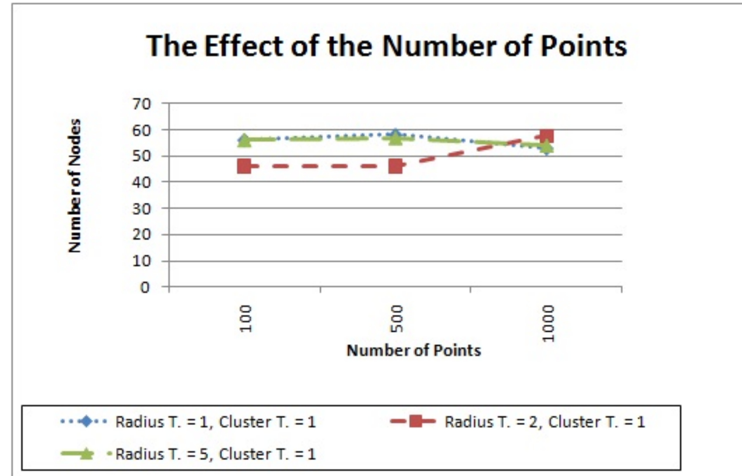

Figure 5.15: The Effect of Number of Vectors on the Number of Nodes, $k = 2$

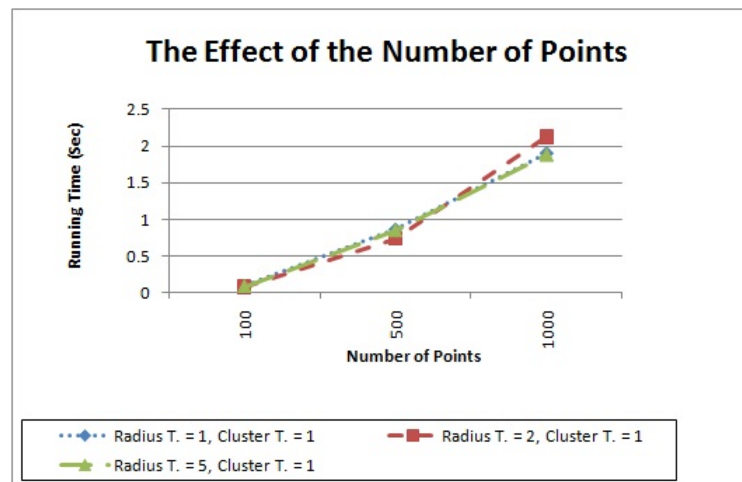

Figure 5.16: The Effect of Number of Vectors on the Running Time, $k = 2$

Figures 5.17, 5.18, 5.19 and 5.20 are organized similarly to Figures 5.13, 5.14, 5.15 and 5.16 respectively, showing the similar plots of parameters for $k = 3$. In all figures the $min-max$ problem, the HS and the $meb_u_away_elim$ algorithms, and $n = 25$ are fixed. Each of the three lines in the figures correspond to different data sets for (radius type, cluster type) with numerical values given by (3, 2), (5, 2), (6, 2), respectively, for Figures 5.17, 5.18 and with numerical values given by (1, 1), (7, 1), (10, 1) respectively for the Figures 5.19 and 5.20. The results are also similar with the case $k = 2$. On the one hand, we can derive from the figures that tripling the number of vectors can increase both the number

of examined nodes and the running time by a factor of 100 for the radius types that are harder to solve. On the other hand, these changes have a negligible effect on both the number of examined nodes in the branch-and-bound tree and the running time for the data sets with specific radius types that are easier to solve.

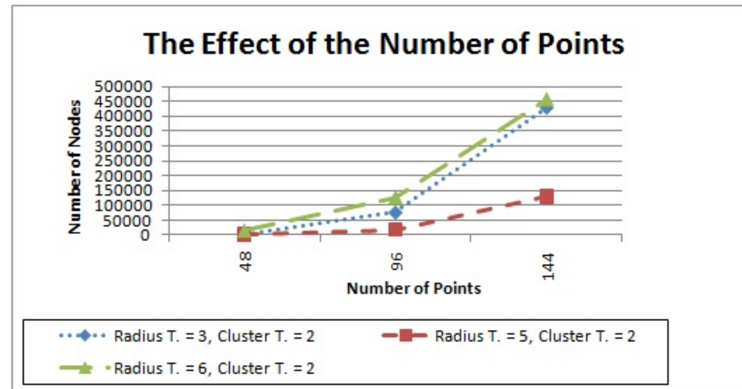

Figure 5.17: The Effect of Number of Vectors on the Number of Nodes, $k = 3$

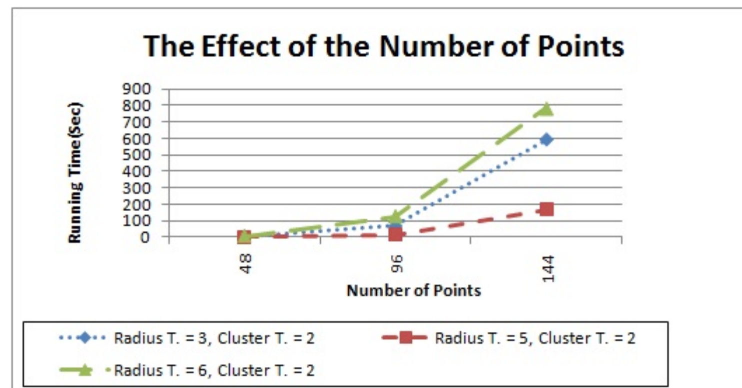

Figure 5.18: The Effect of Number of Vectors on the Running Time, $k = 3$

We can summarize the above results and interpret their reasons as follows. The number of examined nodes in our experiments might grow as the number of vectors increases, while the number of potential nodes in the branch-and-bound tree increases with the number of vectors. Moreover, the size of the subproblems may become larger with the increase in the number of vectors, which also increases the solution times of the subproblems. As a result, the number of examined nodes in the branch-and-bound tree and the running time increase with respect to the increase in m.

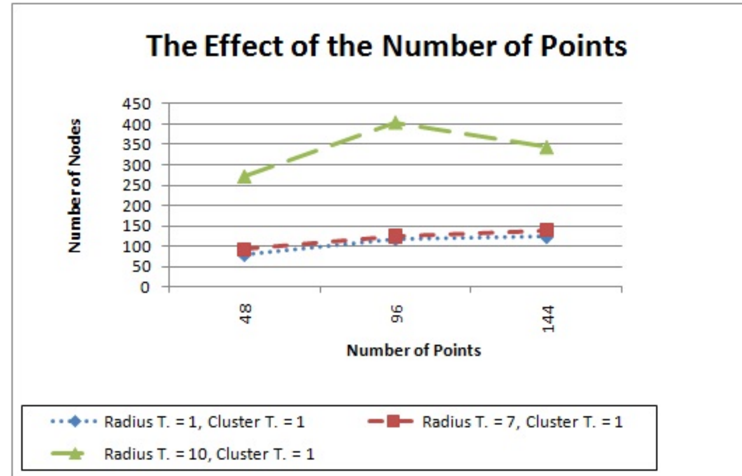

Figure 5.19: The Effect of Number of Vectors on the Number of Nodes, $k = 3$

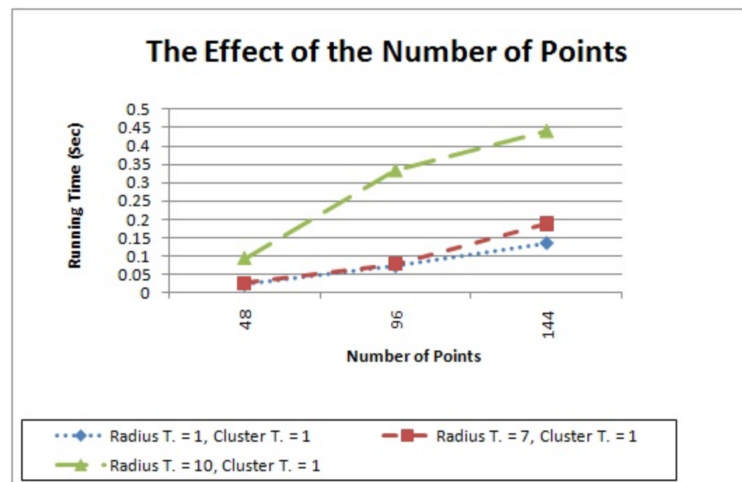

Figure 5.20: The Effect of Number of Vectors on the Running Time, $k = 3$

5.4.5 The Effect of the Number of Dimensions

The effects of n on the efficiency of the proposed algorithms are discussed in this section. We initially illustrate the effect of n on the instances with specific radius types that are harder to solve and then show the same effect on instances that can be solved easily.

We start with more difficult radius types. Figure 5.21 corresponds to the relationship between n and the number of examined nodes in the branch-and-bound

tree for $k = 2$. The relation between n and the running time is given in Figure 5.22. We used the $min - max$ problem, the BEST and the $meb_u_away_elim$ algorithms and $m = 48$ in these figures. Each of the three lines in the figures show different data sets for (radius type, cluster type) with numerical values given by (3, 1), (9, 1), (9, 2) respectively. We can easily deduct from the figures that an increase in the number of dimensions has a significant effect on both the number of examined nodes in the branch-and-bound tree and on the running time for the data sets with specific radius types that are harder to solve. It can be seen that doubling n can increase the number of examined nodes in the branch-and-bound tree by a factor of twenty and also increase the running time by a factor of three hundred.

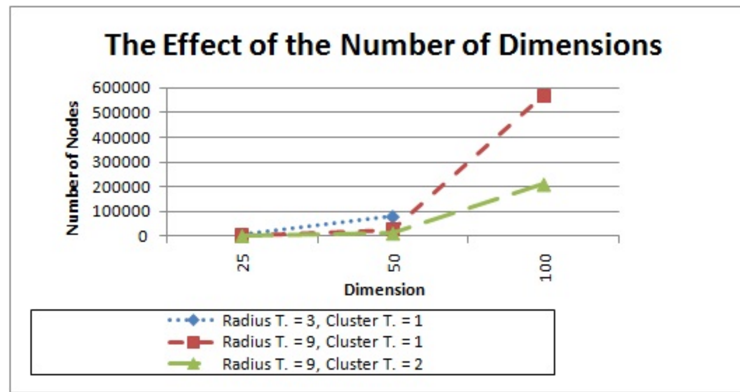

Figure 5.21: The Effect of Number of Dimensions on the Number of Nodes, $k = 2$

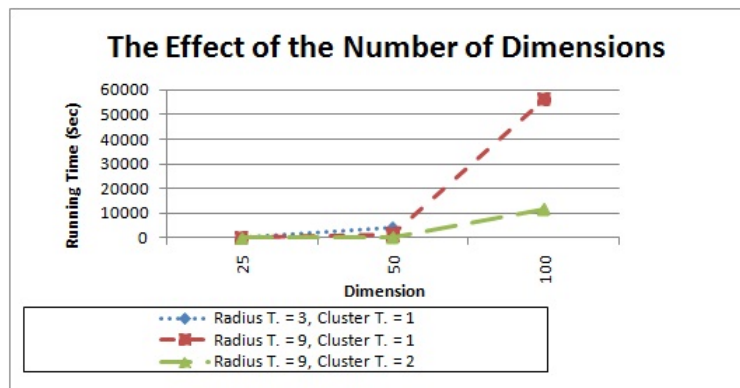

Figure 5.22: The Effect of Number of Dimensions on the Running Time, $k = 2$

Next, we deal with the easier radius types. The relationship between n and the number of examined nodes in the branch-and-bound tree, for $k = 2$ is illustrated

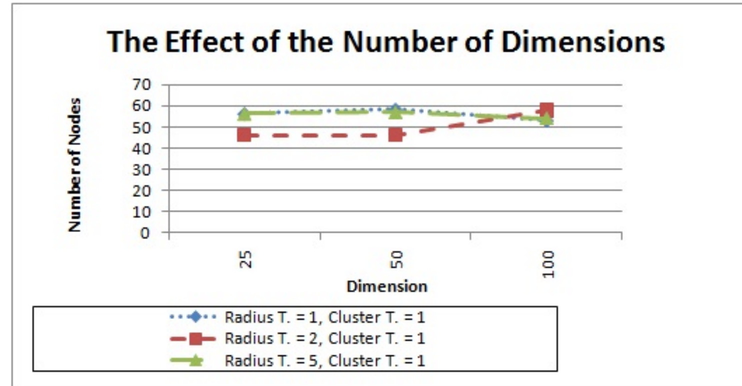

Figure 5.23: The Effect of Number of Dimensions on the Number of Nodes, $k = 2$

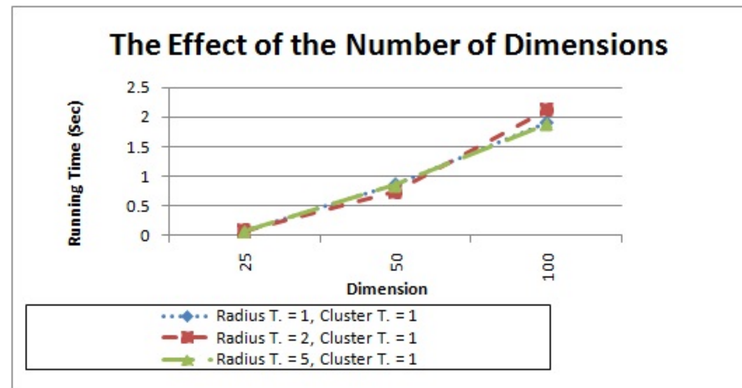

Figure 5.24: The Effect of Number of Dimensions on the Running Time, $k = 2$

in Figure 5.23. We also show the relation between n and the running time in seconds again for $k = 2$ in the next figure, Figure 5.24. We fixed the $min - max$ problem, the BEST and the $meb_u_away_elim$ algorithms and the uniform spherical distribution for results given in Figure 5.23 and 5.24. Each of the three lines in the figures corresponds to different data sets for (radius type, cluster type) with numerical values given by (1, 1), (2, 1), (5, 1) respectively. The figures illustrate that the increase in n (i.e doubling the number of vectors or increasing the number of vectors by a factor of 5) has a neglible effect on both the number of examined nodes in the branch-and-bound tree and has a relatively small effect on the running time for the data sets with specific radius types that are easier to solve.

Figures 5.25, 5.26, 5.27 and 5.28 are organized similarly to Figures 5.21, 5.22, 5.23 and 5.24 respectively, showing the similar pattern of parameters for $k = 3$. In all figures the $min - max$ problem, the HS and the $meb_u_away_elim$ algorithms, and $m = 48$ are fixed. Each of the three lines in the figures corresponds to different data sets for (radius type, cluster type) with numerical values given by (2, 1), (5, 1), (6, 1), respectively, for Figures 5.25, 5.26 and with numerical values given by (1, 1), (7, 1), (10, 1), respectively, for Figures 5.27 and 5.28. The results are also similar the case $k = 2$. On the one hand, we can see from the figures that increasing the number of dimensions by a factor of 5 can incresae the number of examined nodes and the running time by a factor of 1000 and 100, respectively, for the radius types that are harder to solve. On the other hand, these increases do not have a significant effect on both the number of examined nodes in the branch-and-bound tree or the running time for the data sets with specific radius types that are easier to solve.

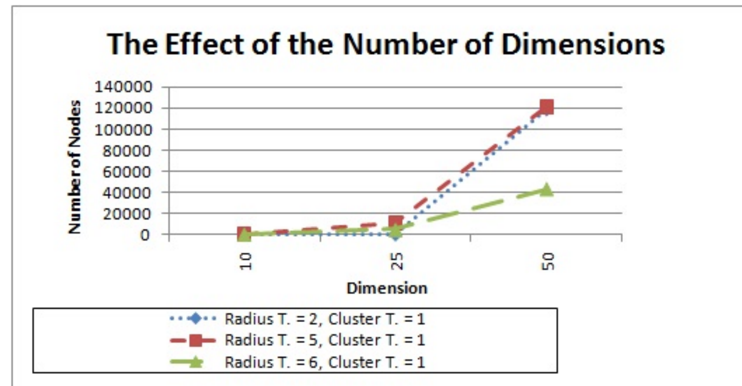

Figure 5.25: The Effect of Number of Dimensions on the Number of Nodes, $k = 3$

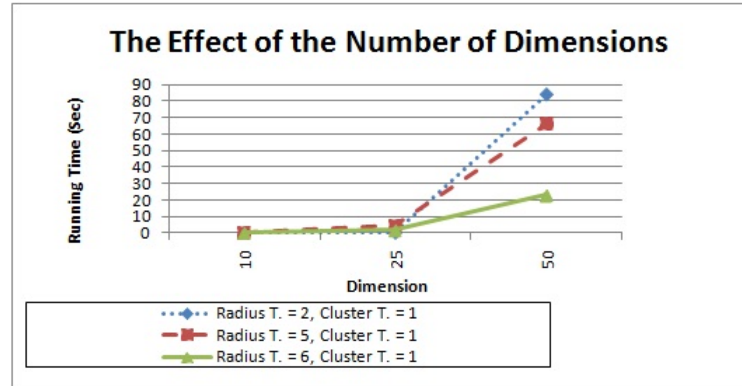

Figure 5.26: The Effect of Number of Dimensions on the Running Time, $k = 3$

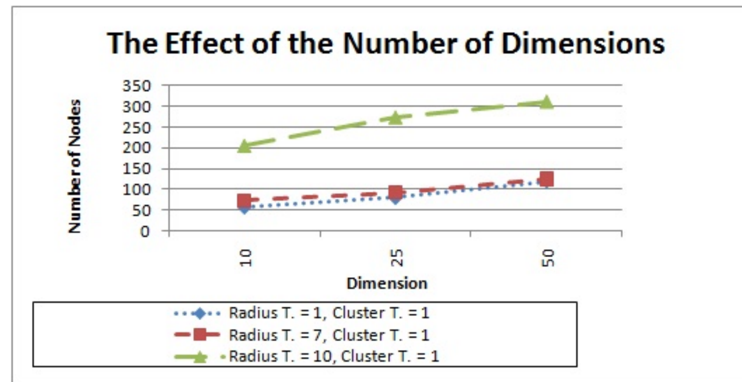

Figure 5.27: The Effect of Number of Dimensions on the Number of Nodes, $k = 3$

These results were not expected since an increase in the number of dimension was expected to affect only the solution time of the MEB algorithms. However, this was not the priori case. Through the increase in the number of dimensions, the vectors get closer to the boundary of the sphere, especially for the uniform spherical distribution. This already changes the structure of the instances. We may find fewer data vectors that are inside the newly constructed cluster (see Algorithm 6, step 14). Hence, this can delay reaching a leaf node in the branch-and-bound tree. In addition, the MEB algorithms perform more efficiently for the data sets in which the vectors are closer to the center of the spheres. As a result, an increase in the dimension of the space also increases the number of examined nodes in the branch-and-bound tree and the running time of the MEB algorithms.

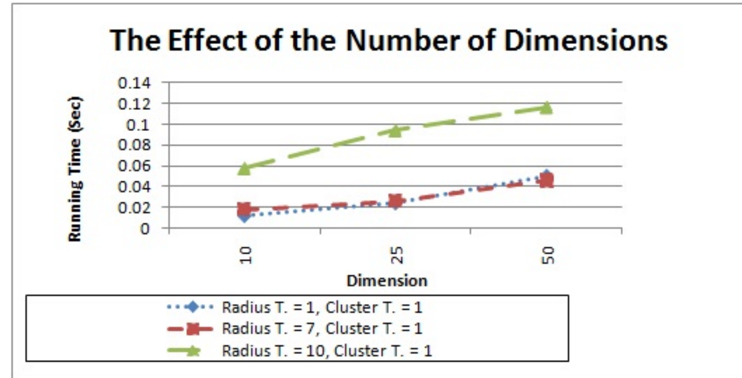

Figure 5.28: The Effect of Number of Dimensions on the Running Time, $k = 3$

5.4.6 The Effect of the Distribution

As mentioned before, we used two different distributions to generate random instances, the uniform and the pseudo-normal spherical distribution. The effects of these distributions on the proposed algorithms are analyzed in this section.

In Figure 5.29, we show the relationship between the distribution and the number of examined nodes in the branch-and-bound tree for $k = 2$. The next figure, Figure 5.30, shows the relation between the distribution and the running time in seconds again for $k = 2$. In Figures 5.29 and 5.30, we used the $min-max$ problem, radius type 3, cluster type 1, the BEST and the $meb_u_away_elim$ algorithms. Each of the three lines in the figures show different sizes of data sets for (n, m) with numerical values given by (25, 100), (25, 500) and (25, 1000). The figures illustrate that the number of examined nodes in the branch-and-bound tree and the running time are larger when the uniform spherical distibution is selected. The increase we see in the parameters (i.e. the number of examined nodes in the branch-and-bound tree and the running time) is much more significant as the size of the data set increases from (25, 100) to (25, 1000) when the uniform spherical distribution is used in comparision to the increase of the same parameters when using the normal distribution.

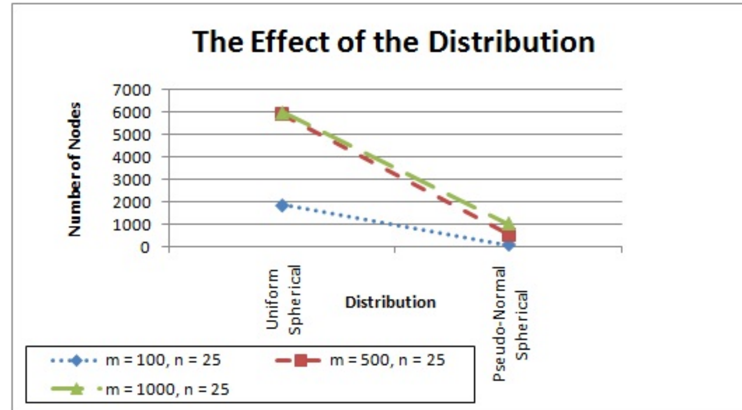

Figure 5.29: The Effect of Distribution on the Number of Nodes, $k = 2$

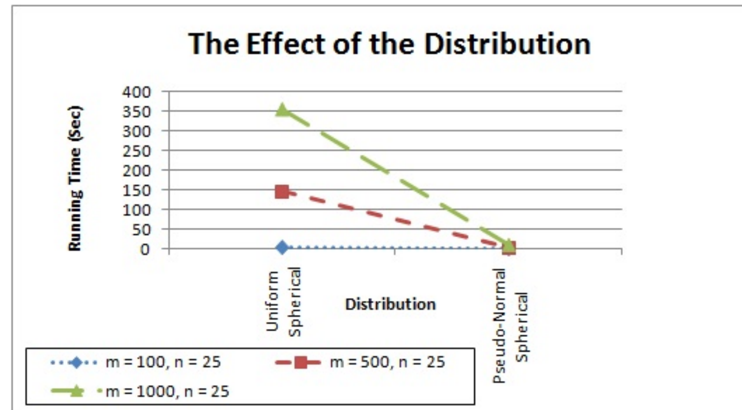

Figure 5.30: The Effect of Distribution on the Running Time, $k = 2$

Figures 5.31 and 5.32 are similar to Figures 5.29 and 5.30, respectively, showing the pattern of parameters for $k = 3$. In these figures, the $min - max$ problem, radius type 5, cluster type 5, the HS and the *meb_u_away_elim* algorithms are fixed. Each line corresponds to a different size of data set for (n, m) as (50, 48), (25, 96) and (10, 144). The number of examined nodes in the branch-and-bound tree and the runnng time are larger for the uniform spherical distribution. The increase observed in the parameters as the data set size increases for the uniform spherical distribution is more than the increase for the pseudo-normal distribution.

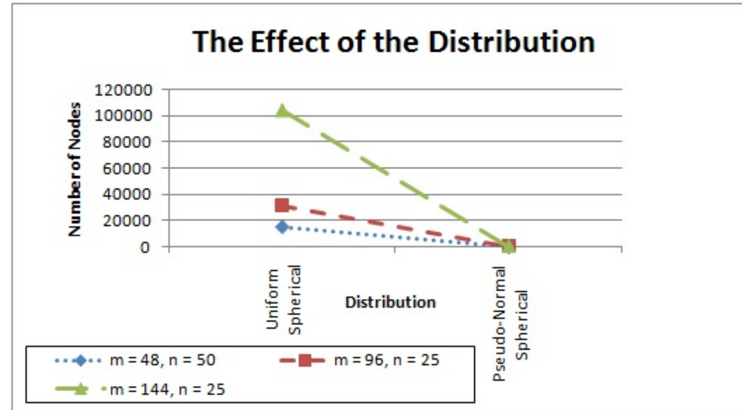

Figure 5.31: The Effect of Distribution on the Number of Nodes, $k = 3$

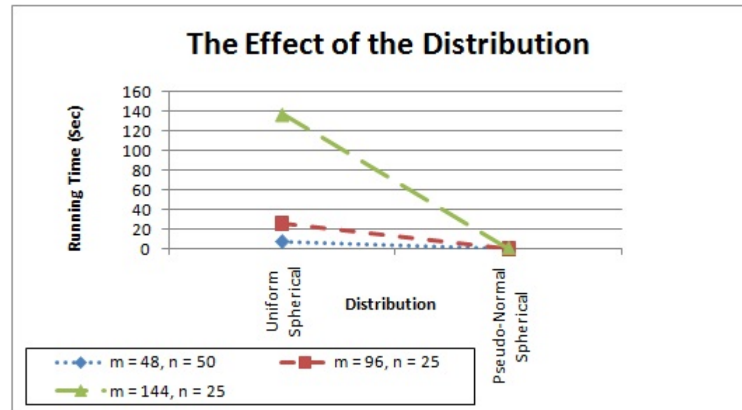

Figure 5.32: The Effect of Distribution on the Running Time, $k = 3$

These results can be explained as follows. The pseudo-normal spherical distribution is a type of centered Gaussian distribution where the center of mass of the data vectors corresponds to the center of the sphere. Therefore, there may exist more vectors that can be inside the newly constructed cluster that is given in the 14th step of the Algorithm 6 with the pseudo-normal spherical distribution. Hence, one can reach a leaf node in a shorter amount of time. On the other hand, the uniform spherical distribution has vectors closer to the outside surface of the sphere than the pseudo-normal spherical distribution. Therefore, computing the MEB of such data set is also more difficult as stated in the literature.

5.4.7 The Effect of the MEB algorithm

Two different MEB algorithms were used to compute the MEB of a given vector set, the meb_u_away and the $meb_u_away_elim$ algorithms, in the scope of this study. This section is devoted to the observation of the effects of these algorithms on the branch-and-bound algorithms.

The relationship between the MEB algorithms and the number of examined nodes in the branch-and-bound tree for $k = 2$ is illustrated in Figure 5.33. We also show the relation between the MEB algorithms and the running time in seconds again for $k = 2$ in the next figure, Figure 5.34. We used the $min-max$ problem, radius type 9, cluster type 1, the DFS algorithm and the uniform spherical distribution for results given in Figure 5.33 and 5.34. Each of the three lines in the figures corresponds to different sizes of data sets for (n, m) with numerical values given by (50, 500), (100, 500) and (100, 1000). Figure 5.33 illustrates that the number of examined nodes in the brach-and-bound tree is the same for both of the MEB algorithms. The subproblems solved in nodes are smaller-scaled problems. Therefore, the difference between the meb_u_away and the $meb_u_away_elim$algorithms are small. Though it can not be seen clearly from Figure 5.34, the branch-and-bound algorithm finds the solution in a shorter amount of time if the $meb_u_away_elim$ algorithm is selected.

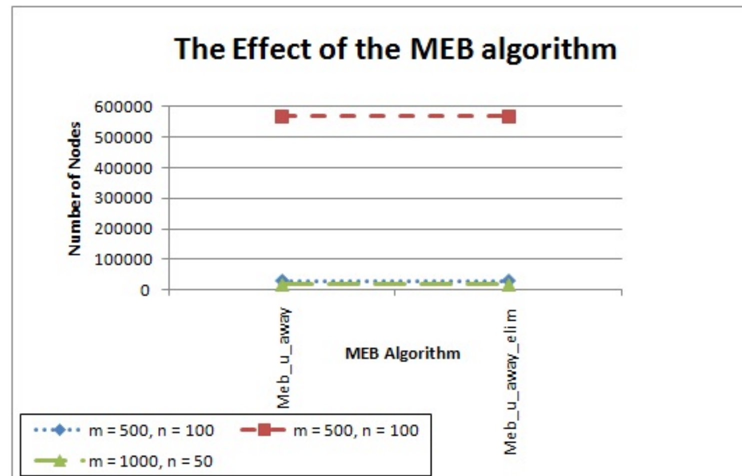

Figure 5.33: The Effect of MEB algorithm on the Number of Nodes, $k = 2$

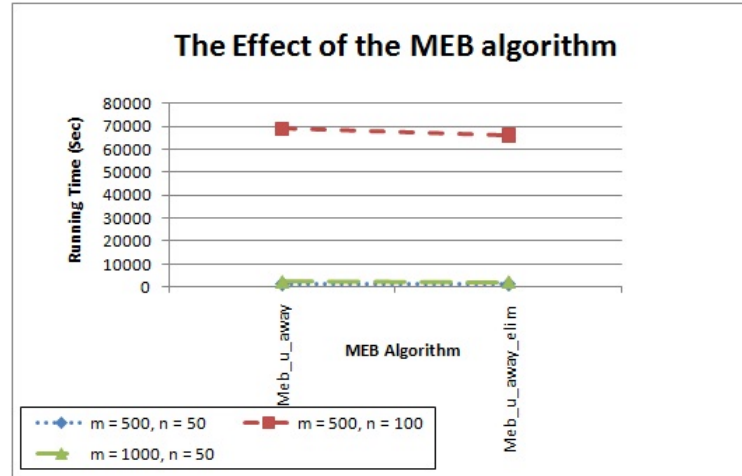

Figure 5.34: The Effect of MEB algorithm on the Running Time, $k = 2$

We used only the *meb_u_away_elim* algorithm for $k = 3$. Therefore, we do not compare two different MEB algorithms for this case. We can summarize the above results as follows. To begin with, both algorithms give same solutions to the same problems (Yıldırım [3]). Therefore, the same radius value of the node was computed in each node of the branch-and-bound tree. Then, the number of examined nodes in the brach-and-bound tree is the same for both of the algorithms if we use the same tree traversal algorithm. In addition, the *meb_u_away_elim* algorithm performed better than the *meb_u_away* algorithm in terms of the solution time in 99% of the instances which is also stated in the literature (Yıldırım [3]).

5.4.8 The Effect of the Tree Traversal Algorithm

As already mentioned before, we used three different tree traveral algorithms to traverse the branch-and-bound tree, the DFS, the BEST and the HS algorithms. We summarize the effects of these algorithms on the problems in this section.

For $k = 2$, we initially solved the problems with the BEST and the DFS algorithms because the HS and the BEST algorithms exibit the same performance unless the predefined memory limit is reached. In Figure 5.35, we show the relationship between the DFS and the BEST algorithms in terms of the number

of nodes in the branch-and-bound tree. On the other hand, we illustrate the relation between the algorithms and the running time in Figure 5.36. We fixed the $min - max$ problem, radius type 4, cluster type 1, the *meb_u_away_elim* algorithm and the uniform spherical distribution in Figures 5.35 and 5.36. Each of the two lines in the figures correspond to different sizes of data sets for (n, m) with numerical values given by (50, 100), (25, 500). The figures illustrate that the number of examined nodes in the branch-and-bound tree and the running time are larger when the DFS algorithm is selected.

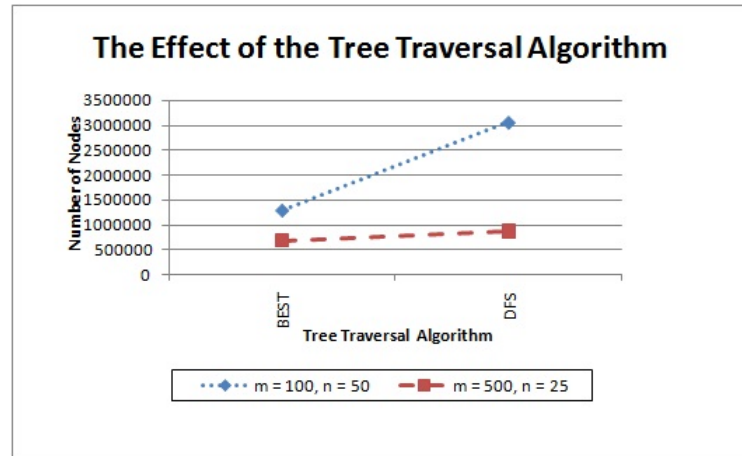

Figure 5.35: The Effect of Tree Traveral Algoritm on the Number of Nodes, $k = 2$

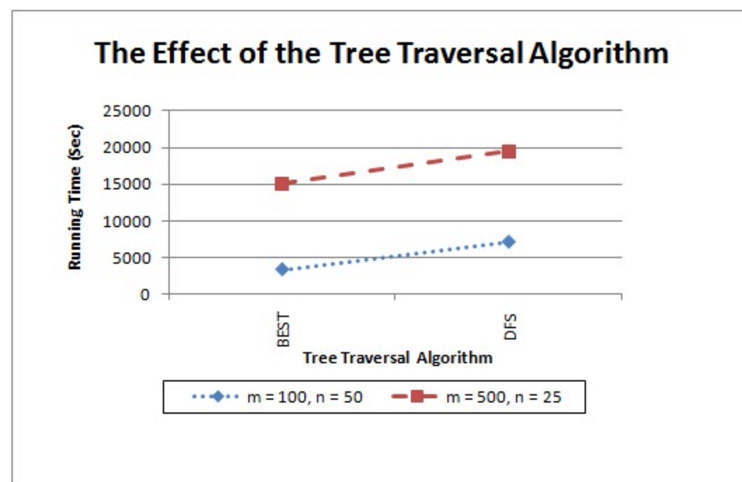

Figure 5.36: The Effect of Tree Traveral Algoritm on the Running Time, $k = 2$

Note that there exists an important issue that must be considered: An increase in the problem size also increases the memory usage of the algorithms. Such an

increase mostly effects the BEST algorithm, while the number of all nodes in the branch-and-bound tree is a constraint, this number can be at most the height of the tree for the DFS algorithm. Therefore, we may face memory problems with the BEST algorithm. As a consequence, we used the HS algorithm for the instances in which the upper memory limit is reached. We, next, examine this case on five specific instances in which the upper memory limit is reached. These instances were rerun with the HS algorithm. The upper memory limit was set to 2.5 GB of the total memory and the lower memory limit was set to 2.0 GB of the total memory. Table 5.7 illustrates these instances and the maximum memory usages of the algorithms for them.

Maximum Memory Usages of the Tree Traversal Algorithms on Specific Instances			
Instance	**BEST Algorithm**	**DFS Algorithm**	**HS Algorithm**
1	3.66 Gb	0.001 Gb	2.58 Gb
2	3.67 Gb	0.001 Gb	2.61 Gb
3	3.58 Gb	0.001 Gb	2.59 Gb
4	3.35 Gb	0.001 Gb	2.52 Gb
5	3.21 Gb	0.001 Gb	2.68 Gb

Table 5.7: Maximum Memory Usages of 5 Specific Instances, $k = 2$

It can be deduced from Table 5.7 that the memory usage can be limited if one uses the HS algorithm. We compare the performances of the algorithms in the following figures. Figure 5.37 and 5.38 correspond to the comparision of the three algorithms in terms of the number of examined nodes in the branch-and-bound tree and running time, respectively, for the specific instances given above.

Figures report that the BEST algorithm is the most efficient one in terms of the number of examined nodes in the branch-and-bound tree. This is expected since the BEST algorithm aims to minimize the total number of examined nodes. On the other hand, notice that the HS algorithm gives similar results to the BEST algorithm.

The HS algorithm is the most efficient algorithm in terms of the running time. As mentioned before, the HS algorithm starts with the BEST algorithm. Hence, it can start with better partial groupings. Then, whenever it switches to the DFS algorithm, it descends as quickly as possible to reach a leaf node. Notice that, while the DFS algorithm starts from scratch, the BEST algorithm starts

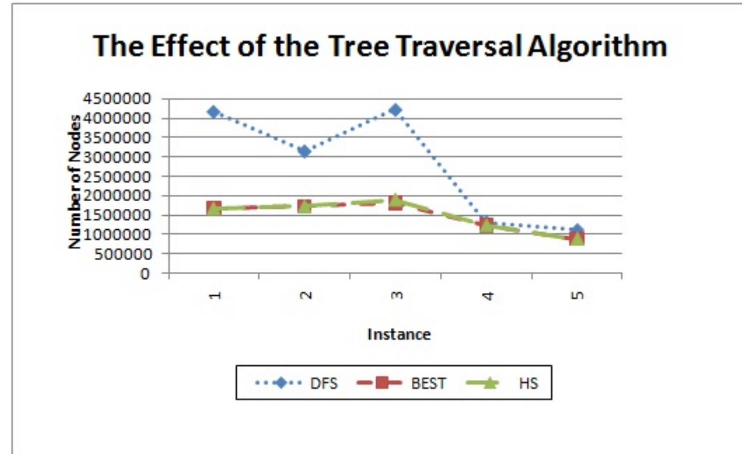

Figure 5.37: Number of Nodes for the Specific Instances

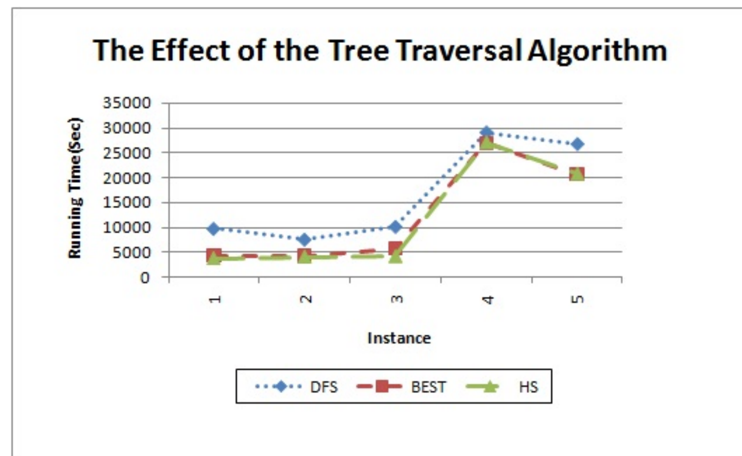

Figure 5.38: Running Time for the Specific Instances

from better partial groupings. Therefore, it might find potentially better upper bounds and might be able to prune more nodes. This observation may explain the better performance of the HS algorithm in terms of the running time.

5.4.9 The General Discussion

This section is devoted to the overall discussion of the computational experiments and their results. First, we tried design not only extensive, but also systematic computational experiments. Therefore, the computational setup was designed in a meaningful way. What we mean by meaningful is that, we took all the

relevant parameters of the problems into consideration one by one and analyzed the effects of each one on the proposed branch-and-bound algorithms for both problems. These parameters are the numner of clusters k, the number of vectors m, the number of dimensions n, the radius and the cluster types, the distribution, the minimum enclosing ball and the tree traveral algorithms. As a result, we can summarize the following results.

The number of spheres, k, and the number of vectors, m, have a significant effect on the efficieny of the proposed algorithms. The number of potential nodes in the branch-and-bound tree increases as a function of k and m. Therefore, the number of examined nodes and accordingly the running time increase as k and/or m increases.

Moreover, the effect of the number of dimensions, n, cannot be underestimated. An increase in n not only makes the MEB algorithms more difficult, but also makes the problems more difficult while it alters the structure of the generated data sets for the distributions used in our computations.

In addition, the difficulty of the problems depends heavily both on the radius and the cluster types. While the radius and the cluster types change the structure of the data vectors, the difficulty of the problems also differs according to the selected radius and cluster types. Furthermore, similar radius and cluster types affects mostly the efficiency of the proposed branch-and-bound algorithms.

Furthermore, two different algorithms were used to compute the MEB of a given vector set. Our results comply with the literature that the *meb_u_away_elim* performs better than the *meb_u_away* algorithm. Therefore, the use of the *meb_u_away_elim* algorithm is suggested.

The DFS, the BEST and the HS algorithms were used to traverse the branch-and-bound tree in the scope of this study. All the algorithms have their own advantages and disadvantages. However, we observed that the HS algorithm is the most efficient algorithm in terms of the running time, and the DFS algorithm is the most efficient algorithm with respect to the memory usage. Therefore, if there is a limited memory or the size of the input set is very large the DFS

algorithm is recommended. Otherwise it is better to use the HS algorithm but with carefully selected upper and lower memory limits.

Besides, we used two different distributions to generate random instances within a sphere, the uniform and the pseudo-normal spherical distributions. It is more difficult for the algorithms to solve the instances generated from the uniform spherical distribution due to the structure of the data sets.

Finally, the proposed branch-and-bound algorithms are capable of solving the $min - sum$ problem more efficiently than the $min - max$ problem for this data set.

Chapter 6

Conclusion

In this study, a specialized algorithm is designed and implemented for clustering problems using minimum enclosing balls. The main clustering problems of focus in this thesis are the computation of k spheres in a high dimensional space that enlose a given set of m vectors, which corresponds to the set of objects, in such a way that the radius of the largest sphere or the sum of the radii of spheres is as small as possible. The aim is to identify the underlying structures and patterns among the objects correctly in order to divide the set of object into k clusters based on the level of similarity among them. The Euclidean distance is used as a measure of similarity among the objects.

General purpose solvers cannot solve the problems efficiently since they are not able to exploit the specific underlying structures of the problems. Therefore, we designed specific algorithms based on a systematic and efficient search of an optimal solution using a Branch-and-Bound framework. A software package is developed in order to implement the proposed branch-and-bound algorithms. We designed the architecture of the software in a flexible and modular fashion. Therefore, it constitutes a solid foundation for further studies. The algorithms were tested in practice via the software package.

Extensive and systematic computational experiments were designed for testing the efficiency of the algorithms in practice. The proposed branch-and-bound

algorithms were tested in a controlled manner of all parameters. We suggested the most efficient MEB and tree traversal algorithms in terms of time and space complexities. The results of the experiments revealed that the algorithms are able to solve medium-scale intances of the problems effectively and efficiently.

For future research, this study can be extended in many ways. First, the proposed algorithms can be tested with larger values of k, n or m values such as 10, 1000 and 10000 respectively. Moreover, it is possible to implement different MEB algorithms to compute the MEB of a given vector set. Furthermore, different tree traversal algorithms can be developed. These algorithms can be a compromise between the proposed tree traversal algorithms, such as the combination of the BEST, BFS and DFS algorithms. A new tree traversal algorithm can be developed as well. In addition, only specific lower and upper memory limits are tested in the scope of this study. However, the HS algorithm can be tested with different upper and lower memory limit in order to find the optimal values of each memory limits. The distributions used to generate the random instances can be choosen differently and various distributions can be used to generate random instances, especially for the $min-sum$ problem. The radius and cluster types affects directly the efficiency of the algorithms. An extensive computational study can be performed to identify the specific reasons of this phenomenon. Moreover, the instances with new radius or cluster types can be generated. The effect of the accuracy parameter of the MEB algorithms, ϵ, can be studied. Finally, it would be interesting to define the clusters with differens geometric objects, such as ellipsoids or boxes.

Bibliography

[1] P. K. Agarwal and M. Sharir. Planar geometric location problems. *Algorithmica*, 11:185–195, 1994.

[2] R. Agrawal, T. Imielinski, B. Iyer, and A. Swami. An interval classifier for database mining applications. *Proc. 18th Internat. Conf. Very Large Databases, Morgan Kauffman, San Mateo, CA*, page 560573, August 1992.

[3] S. D. Ahipaşaoğlu and E. A. Yıldırım. Identification and elimination of interior points for the minimum enclosing ball problem. *SIAM Jourmal On Optimization*, 19:1392–1396, 2008.

[4] M. Badoiu and K. L. Clarkson. Smaller core-sets for balls. *In Proceedings of the 14th Annual Symposium on Discrete Algorithms*, pages 801–802, 2003.

[5] M. Badoiu, S. Har-Peled, and P. Indyk. Approximate clustering via core-sets. *In Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 250–257, 2002.

[6] J. Burkardt. Random data. *http://www.csit.edu/ burkardt/m_src/ random_data/random_data.html*.

[7] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. *Proc. 29th Annu. ACM Sympos. Theory Comput*, pages 626–635, 1997.

[8] M. Charikar, C. Chekuri, A. Goel, and S. Guha. Rounding via trees: deterministic approximation algorithms for group steiner trees and k-median.

Proceedings of the 30th Annual ACM Symposium on Theory of Computing, pages 114–123, 1998.

[9] M. Charikar and S. Guha. Improved combinatorial algorithms for the facility location and k-median problems. *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, pages 378–388, 1999.

[10] N. Christofidesa and P. Viola. The optimum location of multi-centers of a graph. *Opnl. Res. Quart.*, 22:145–154, 1971.

[11] G. Chrystal. On the problem to construct the minimum circle enclosing n given points in the plane. *In Proceedings of the Edinburgh Mathematical Society*, 3:30–33, 1985.

[12] Z. Drezner. The planar two-center and two-median problems. *Transportation Science*, 18:351–361, 1984.

[13] Z. Drezner. Facility location. *Springer-Verlag, New York*, 1995.

[14] D. Elzinga and D. Hearn. The minimum covering sphere problem. *Journal of Physics: Conference Series Create an alert RSS this journal*, 19:96–104, 1972.

[15] E.Minieka. The m-center problem. *SIAM Review*, 12:138–139, 1970.

[16] D. Eppstein. Faster construction of planar two-centers. *In Proceedings of the Eighth ACM-SIAM Symposium on Discrete Algorithms*, 1997.

[17] R. S. Garfinkel, A. W. Neebe, and M. R. Rao. The m-center problem: Bottleneck facility location. *Working Paper, Graduate School of Management, University of Rochester, New York*, 1974.

[18] A. J. Goldman. Optimum locations for centers in a network. *Transportation Science*, 4:352–360, 1969.

[19] A. J. Goldman. Optimal center location in simple networks. *Transportation Science*, 5:212–221, 1971.

[20] T. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoratical Computer Science*, 38:293–306, 1985.

[21] S. L. Hakimi. Optimum locations of switching centers and the absolute centers and medians of a graph. *Operations Research*, 12:450–459, 1964.

[22] S. L. Hakimi. Optimum distribution of switching centers in a communication network and some related graph theoretic problems. *Operations Research*, 13:462–475, 1965.

[23] S. L. Hakimi and S. M. Maheswari. Optimum locations of centers in networks. *Operations Research*, 20:967–973, 1972.

[24] S. L. Hakimi, E. F. Schmeichel, and J. G. Pierce. On p-centers in networks. *Transportation Science*, 12:1–15, 1978.

[25] J. Hershberger. A faster algorithm for the two-center decision problem. *Information Processing Letters*, 47:23–39, 1993.

[26] D. S. Hochbaum and D. Shmoys. A best possible heuristic for the k-center problem. *Math. Oper. Res.*, 10:180–184, 1985.

[27] D. S. Hochbaum and D. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *J. ACM*, 33:533–550, 1986.

[28] W. L. Hsu and G. L. Nemhauser. Easy and hard bottleneck location problems. *Discrete Applied Mathematics*, 1:209–215, 1979.

[29] M. Hung and J. G. Morris. Solving constrained discrete space minimax location-allocation problems. *Working Paper, Center for Business and Economic Research, Kent State University, Kent, Ohio*, 1975.

[30] K. Jain and V. V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *Journal of the ACM*, 48:274–296, 2001.

[31] J. W. Jaromczyk and M. Kowaluk. A geometric proof of the combinatorial bounds for the number of optimal solutions to the 2-center euclidean problem. *Proceedings of the Seventh Canadian Conference on Computational Geometry*, pages 19–24, 1995a.

[32] O. Kariv and S. L. Hakimi. An algorithmic approach to network location problems. I: The p-centers. *SIAM Journal on Applied Mathematics*, 37:513–538, 1979a.

[33] O. Kariv and S. L. Hakimi. An algorithmic approach to network location problems. II: The p-medians. *SIAM Journal on Applied Mathematics*, 37:539–560, 1979b.

[34] P. Kumar, J. S. B. Mitchell, and E. A. Yıldırım. Approximate minimum enclosing balls in high dimensions using core-sets. *The ACM Journal of Experimental Algorithmics*, 8, 2003.

[35] J. Levy. An extended theorem for location on a network. *Opnl. Res. Quart*, 18:433–442, 1967.

[36] S. Leyffer, J. Linderoth, J. Luedtke, A. Miller, and T. Munson. Applications and algorithms for mixed integer nonlinear programming. *Journal of Physics: Conference Series*, 180, 2009.

[37] J. Matousek. Randomized optimal algorithm for slope selection. *Inf. Process. Lett.*, 39:183–187, 1991b.

[38] N. Megiddo and K. Supowit. On the complexity of some common geometric location problems. *SIAM Journal on Computing*, 13:182–196, 1984.

[39] R. Panigrahy. Minimum enclosing polytope in high dimensions. *Unpublished manuscript*, 2006.

[40] M. Sharir. A near-linear algorithm for the planar 2-center problem. *Discrete Comput. Geom*, 18:125–134, 1997.

[41] J. J. Sylvester. On Poncelets approximate linear valuation of surd forms. *Philosophical Magazine*, 1860.

[42] B. C. Tansel, R. L. Francis, and T. J. Lowe. Location on networks: A survey. Part I: The p-center and p-median problems. *Management Science*, 29:482–497, 1983a.

[43] B. C. Tansel, R. L. Francis, and T. J. Lowe. Location on networks: A survey. Part II: Exploiting tree network structure. *European Journal of Operations Research*, 29:498–511, 1983b.

[44] C. Toregas, R. Swain, C. ReVelle, and L. Bergman. The location of emergency service facilities. *Operations Research*, 19:1363–1373, 1971.

[45] R. E. Wendell and A. P. Hurter. Optimum locations on a network. *Transportation Science*, 7:18–33, 1973.

[46] E. A. Yıldırım. Two algorithms for the minimum enclosing ball problem. *SIAM Journal on Optimization*, 19:1368–1391, 2008.

Appendix A

Numerical Results

Table A.1: Number of Examined Nodes in the Branch-and-Bound Tree, $k = 2$

Radius Type 1, Cluster Type 1

n/m	100	500	1000
25	56,2	58,2	53
50	91	99,4	99,8
100	125,8	181	179

Radius Type 1, Cluster Type 2

n/m	100	500	1000
25	59,4	57,8	57,8
50	92,2	99,4	99,4
100	137,4	180,6	159

Radius Type 1, Cluster Type 3

n/m	100	500	1000
25	54,6	55,8	56,2
50	83,4	97,4	91,4
100	123,4	177,8	169,8

Radius Type 2, Cluster Type 1

n/m	100	500	1000
25	46,2	46,2	57,8
50	75,4	81	78,2
100	103	112,2	96,6

Radius Type 2, Cluster Type 2

n/m	100	500	1000
25	54,6	53	45,4
50	77	89	84,2
100	83,8	147,8	160,2

Radius Type 2, Cluster Type 3

n/m	100	500	1000
25	32,2	61	40,2
50	48,6	91,8	52,6
100	62,6	133,8	93,8

Radius Type 3, Cluster Type 1			
n/m	**100**	**500**	**1000**
25	1844,2	5893	5973
50	19081	80143,8	68977,8
100	67407,4		

Radius Type 3, Cluster Type 2			
n/m	**100**	**500**	**1000**
25	854,6	5827,4	3169,8
50	4605,4	69609,4	51103
100	24591,4		

Radius Type 3, Cluster Type 3			
n/m	**100**	**500**	**1000**
25	276,2	2815	3215
50	711,6	23814,2	32175
100	1910,2		

Radius Type 4, Cluster Type 1			
n/m	**100**	**500**	**1000**
25	21878,2	684929	
50	1296431,4		
100			

Radius Type 4, Cluster Type 2			
n/m	**100**	**500**	**1000**
25	10390,6	409195	
50	221661		
100			

Radius Type 4, Cluster Type 3			
n/m	**100**	**500**	**1000**
25	752,6	97178,2	
50	12143		
100			

Radius Type 5, Cluster Type 1			
n/m	**100**	**500**	**1000**
25	56,2	57	54,2
50	83,4	96,2	98,6
100	125,8	179	173,8

Radius Type 5, Cluster Type 2			
n/m	**100**	**500**	**1000**
25	55,8	60,2	55,8
50	91,4	99	99,8
100	132,6	180,6	156,6

Radius Type 5, Cluster Type 3			
n/m	**100**	**500**	**1000**
25	53	57,4	53,8
50	82,6	99	96,2
100	118,2	175	167,4

Radius Type 6, Cluster Type 1			
n/m	**100**	**500**	**1000**
25	344,6	194,2	248,2
50	647,8	567,4	701
100	1362,2	1376,2	1083

Radius Type 6, Cluster Type 2			
n/m	**100**	**500**	**1000**
25	118,2	217,8	223,4
50	304,2	504,6	414,2
100	514,2	1215	957

Radius Type 6, Cluster Type 3			
n/m	**100**	**500**	**1000**
25	84,6	155,8	161
50	167,8	313,4	350,6
100	247,4	875,8	857,4

Radius Type 7, Cluster Type 1			
n/m	**100**	**500**	**1000**
25	8513,4	89448,2	
50	421281		
100			

Radius Type 7, Cluster Type 2			
n/m	**100**	**500**	**1000**
25	2880,6	38811	
50	47114,2		
100			

Radius Type 7, Cluster Type 3			
n/m	**100**	**500**	**1000**
25	291	13581,8	
50	2023,4		
100			

Radius Type 8, Cluster Type 1			
n/m	**100**	**500**	**1000**
25	59,8	62,6	63
50	84,2	115	100,2
100	123	187	180,6

Radius Type 8, Cluster Type 2			
n/m	**100**	**500**	**1000**
25	63,8	66,6	62,6
50	94,2	107,8	101,4
100	134,2	189,4	182,2

Radius Type 8, Cluster Type 3			
n/m	**100**	**500**	**1000**
25	59,4	62,2	59,8
50	82,6	107,8	102,2
100	124,2	178,6	171,4

Radius Type 9, Cluster Type 1			
n/m	**100**	**500**	**1000**
25	546,2	1262,6	1479,4
50	3675	12281,8	6692,2
100	17787	209881	

Radius Type 9, Cluster Type 2			
n/m	**100**	**500**	**1000**
25	336,2	767,6	987,4
50	1675	7521,8	3897,2
100	10787	121881	

Radius Type 9, Cluster Type 3			
n/m	**100**	**500**	**1000**
25	260,2	925,8	1034,6
50	457	7225,4	6998,2
100	1659	75384,2	

Radius Type 10, Cluster Type 1			
n/m	**100**	**500**	**1000**
25	88,2	68,6	68,2
50	111,8	115,8	113,8
100	141,8	209	201,4

Radius Type 10, Cluster Type 2			
n/m	**100**	**500**	**1000**
25	75,4	72,2	69,4
50	114,6	123	112,6
100	161,8	203,4	194,2

Radius Type 10, Cluster Type 3			
n/m	**100**	**500**	**1000**
25	97,4	75,8	68,2
50	114,2	123,4	109,4
100	136,2	193,4	194,2

Table A.2: Running Time, $k = 2$

Radius Type 1, Cluster Type 1			
n/m	**100**	**500**	**1000**
25	0,092	0,862	1,9
50	0,238	2,46	6,97
100	0,54	9,112	21,89

Radius Type 1, Cluster Type 2			
n/m	**100**	**500**	**1000**
25	0,074	0,832	1,944
50	0,204	2,49	6,6
100	0,486	8,01	18,418

Radius Type 1, Cluster Type 3			
n/m	**100**	**500**	**1000**
25	0,084	0,748	2,038
50	0,256	2,628	5,892
100	0,548	9,162	21,808

Radius Type 2, Cluster Type 1			
n/m	**100**	**500**	**1000**
25	0,078	0,734	2,124
50	0,232	2,642	5,792
100	0,49	7,272	17,074

Radius Type 2, Cluster Type 2			
n/m	**100**	**500**	**1000**
25	0,064	0,68	1,452
50	0,192	2,198	5,292
100	0,26	6,484	21,41

Radius Type 2, Cluster Type 3			
n/m	**100**	**500**	**1000**
25	0,036	0,654	0,996
50	0,114	1,964	1,816
100	0,15	4,104	4,816

Radius Type 3, Cluster Type 1			
n/m	**100**	**500**	**1000**
25	3,196	145,612	354,122
50	53,774	4060,932	8844,912
100	274,868		

Radius Type 3, Cluster Type 2			
n/m	**100**	**500**	**1000**
25	1,062	89,926	122,104
50	8,402	2213,16	4415,528
100	69,896		

Radius Type 3, Cluster Type 3			
n/m	**100**	**500**	**1000**
25	0,212	24,74	71,056
50	0,856	341,024	1375,416
100	3,236		

Radius Type 4, Cluster Type 1			
n/m	**100**	**500**	**1000**
25	34,958	15089,042	
50	3367,93		
100			

Radius Type 4, Cluster Type 2			
n/m	**100**	**500**	**1000**
25	14,246	6674,424	
50	446,796		
100			

Radius Type 4, Cluster Type 3			
n/m	**100**	**500**	**1000**
25	0,82	1079,016	
50	20,958		
100			

Radius Type 5, Cluster Type 1			
n/m	**100**	**500**	**1000**
25	0,084	0,85	1,884
50	0,254	2,574	7,064
100	0,522	9,236	22,37

Radius Type 5, Cluster Type 2			
n/m	**100**	**500**	**1000**
25	0,076	0,862	2,048
50	0,246	2,568	6,806
100	0,434	8,114	18,338

Radius Type 5, Cluster Type 3			
n/m	**100**	**500**	**1000**
25	0,09	0,836	1,836
50	0,232	2,642	7,08
100	0,46	9,062	21,216

Radius Type 6, Cluster Type 1			
n/m	**100**	**500**	**1000**
25	0,608	3,184	10,984
50	1,872	20,396	66,692
100	6,07	100,034	189,698

Radius Type 6, Cluster Type 2			
n/m	**100**	**500**	**1000**
25	0,13	2,576	7,202
50	0,536	11,646	22,988
100	1,338	52,406	106,63

Radius Type 6, Cluster Type 3			
n/m	**100**	**500**	**1000**
25	0,078	1,058	2,788
50	0,238	3,418	9,81
100	0,462	17,696	49,196

Radius Type 7, Cluster Type 1			
n/m	**100**	**500**	**1000**
25	13,256	2079,43	
50	1056,912		
100			

Radius Type 7, Cluster Type 2			
n/m	**100**	**500**	**1000**
25	3,364	585,132	
50	82,738		
100			

Radius Type 7, Cluster Type 3			
n/m	**100**	**500**	**1000**
25	0,208	112,944	
50	2,51		
100			

Radius Type 8, Cluster Type 1			
n/m	**100**	**500**	**1000**
25	0,09	0,908	2,414
50	0,218	3,306	6,81
100	0,504	9,69	23,992

Radius Type 8, Cluster Type 2			
n/m	**100**	**500**	**1000**
25	0,086	0,922	2,164
50	0,224	2,93	7,104
100	0,452	8,732	24,896

Radius Type 8, Cluster Type 3			
n/m	**100**	**500**	**1000**
25	0,098	0,906	2,278
50	0,204	2,868	6,9
100	0,542	9,338	24,064

Radius Type 9, Cluster Type 1			
n/m	**100**	**500**	**1000**
25	2,102	37,264	76,768
50	20,652	1311,088	1997,276
100	515,43	56090,52	

Radius Type 9, Cluster Type 2			
n/m	**100**	**500**	**1000**
25	0,642	17,782	56,004
50	7,638	345,754	474,562
100	49,51	11852,894	

Radius Type 9, Cluster Type 3			
n/m	**100**	**500**	**1000**
25	0,21	8,168	22,844
50	0,536	106,6	274,826
100	3,06	1995,006	

Radius Type 10, Cluster Type 1			
n/m	**100**	**500**	**1000**
25	0,126	0,932	2,37
50	0,31	3,078	7,58
100	0,488	10,506	25,294

Radius Type 10, Cluster Type 2			
n/m	**100**	**500**	**1000**
25	0,094	0,988	2,496
50	0,26	3,006	7,372
100	0,498	9,286	23,71

Radius Type 10, Cluster Type 3			
n/m	**100**	**500**	**1000**
25	0,148	1,062	2,416
50	0,318	3,416	7,962
100	0,526	9,468	26,45

Table A.3: Number of Examined Nodes for Branch-and-Bound Tree, $k = 3$

Radius Type 1, Cluster Type 1			
n/m	**48**	**96**	**144**
10	56,4	67,8	66
25	79,2	118,2	123,6
50	118,8	173,4	

Radius Type 1, Cluster Type 2			
n/m	**48**	**96**	**144**
10	52,8	58,2	66,6
25	89,4	108	122,4
50	111,6	156	

Radius Type 2, Cluster Type 1			
n/m	**48**	**96**	**144**
10	156,6	349,8	364,2
25	522,6	739,6	3367,2
50	118897,2	89565984	

Radius Type 2, Cluster Type 2			
n/m	**48**	**96**	**144**
10	163,8	376,2	368,4
25	431,2	1823,2	24792,6
50	46564,8	64614968,4	

Radius Type 2, Cluster Type 3			
n/m	**48**	**96**	**144**
10	70,2	159,6	187,8
25	134,4	292,8	844,2
50	175,8	570	

Radius Type 2, Cluster Type 4			
n/m	**48**	**96**	**144**
10	391,2	505,2	1052,4
25	1215,6	3049722,6	5662,6
50	1531189,2		

Radius Type 3, Cluster Type 1			
n/m	**48**	**96**	**144**
10	243,6	2242,4	4489,4
25	1931	52873,4	559707
50	17282,		

Radius Type 3, Cluster Type 2			
n/m	**48**	**96**	**144**
10	551,8	2130,4	6951
25	2397,2	76890,2	428882,4
50	13560,2		

Radius Type 3, Cluster Type 3			
n/m	**48**	**96**	**144**
10	143,4	436,8	711,4
25	200	4106,2	19038
50	348,6		

Radius Type 3, Cluster Type 4			
n/m	**48**	**96**	**144**
10	790,8	5056,2	16998,6
25	11897,6	620487	14960882,2
50	202719,2		

Radius Type 4, Cluster Type 1			
n/m	**48**	**96**	**144**
10	162	210	290,4
25	232,2	578,4	545,4
50	429,6		

Radius Type 4, Cluster Type 2			
n/m	**48**	**96**	**144**
10	212,4	379,2	395,4
25	418,8	782,4	703,2
50	586,2		

Radius Type 4, Cluster Type 3			
n/m	**48**	**96**	**144**
10	214,2	298,2	248,4
25	274,2	506,4	640,2
50	0445		

Radius Type 4, Cluster Type 4			
n/m	**48**	**96**	**144**
10	136,2	228,6	230,8
25	378,6	394,6	949,2
50	293,4		

Radius Type 5, Cluster Type 1			
n/m	**48**	**96**	**144**
10	545	3641,4	10542,2
25	11364,2	134829,8	1016488,6
50	121168,8		

Radius Type 5, Cluster Type 2			
n/m	**48**	**96**	**144**
10	469,6	1303,8	4046,2
25	1556,2	18642,6	129424,8
50	11190		

Radius Type 5, Cluster Type 3			
n/m	**48**	**96**	**144**
10	186,2	907,4	1325,6
25	310,2	1998,8	5877,2
50	428,8		

Radius Type 5, Cluster Type 4			
n/m	**48**	**96**	**144**
10	885,2	2720,2	6947,4
25	5451,6	145158	1487217,4
50	108591		

Radius Type 5, Cluster Type 5			
n/m	**48**	**96**	**144**
10	571,2	2328,2	3183,6
25	1543,4	31415,8	103862,6
50	14972		

Radius Type 5, Cluster Type 6			
n/m	**48**	**96**	**144**
10	123,6	736,2	891,4
25	276,6	2901,6	5932,4
50	262,4		

Radius Type 5, Cluster Type 7			
n/m	**48**	**96**	**144**
10	506,2	1583,4	3242,6
25	2382,2	23916	130412,6
50	6582,8		

Radius Type 6, Cluster Type 1			
n/m	**48**	**96**	**144**
10	769,2	3085,6	4874,2
25	5298,8	44855,6	128304,8
50	43574,6		

Radius Type 6, Cluster Type 2			
n/m	**48**	**96**	**144**
10	1268,4	4613,8	5620,8
25	15096,8	127272,8	457889
50	95833,4		

Radius Type 6, Cluster Type 3			
n/m	**48**	**96**	**144**
10	1072,8	1938	2597,6
25	8676,8	72758,6	92582
50	20709,2		

Radius Type 6, Cluster Type 4			
n/m	**48**	**96**	**144**
10	597,4	1355	2246,4
25	6948,2	54156,8	208994
50	15378,8		

Radius Type 7, Cluster Type 1			
n/m	**48**	**96**	**144**
10	73,2	86,4	107,2
25	92,4	124,2	139,2
50	124,6		

Radius Type 7, Cluster Type 2			
n/m	**48**	**96**	**144**
10	113,8	99	97,6
25	101,8	123	135,6
50	120,6		

Radius Type 8, Cluster Type 1			
n/m	**48**	**96**	**144**
10	519,8	1673,6	2375,6
25	4104	39534,4	43179,8
50	3723,4		

Radius Type 8, Cluster Type 2			
n/m	**48**	**96**	**144**
10	578,8	1295,8	2373,8
25	2352	14926,2	2309691,2
50	3097		

Radius Type 8, Cluster Type 3			
n/m	**48**	**96**	**144**
10	215	440,8	632,2
25	654,2	3050,4	16029,6
50	952		

Radius Type 8, Cluster Type 4			
n/m	**48**	**96**	**144**
10	1147	2613	3715,4
25	3047,2	34668	1041264
50	24915		

Radius Type 9, Cluster Type 1			
n/m	**48**	**96**	**144**
10	459,8	767,6	1415,8
25	1607,6	9416	17677,6
50	5292,4		

Radius Type 9, Cluster Type 2			
n/m	**48**	**96**	**144**
10	416	1530,4	1825,8
25	3613,2	12139	40375,8
50	12240,8		

Radius Type 9, Cluster Type 3			
n/m	**48**	**96**	**144**
10	361,4	635	1289
25	2857,6	6530,2	10849,6
50	4645		

Radius Type 9, Cluster Type 4			
n/m	**48**	**96**	**144**
10	443,8	871,4	1758,6
25	1962	5044,8	16104,8
50	9895,4		

Radius Type 10, Cluster Type 1			
n/m	**48**	**96**	**144**
10	204,8	223,4	198
25	272,6	403,2	343,6
50	310,4		

Radius Type 10, Cluster Type 2			
n/m	**48**	**96**	**144**
10	222,2	229,4	291,8
25	402,4	516,6	441,4
50	417,6		

Table A.4: Running Time, $k = 3$

Radius Type 1, Cluster Type 1			
n/m	**48**	**96**	**144**
10	0,012	0,03	0,048
25	0,024	0,074	0,136
50	0,05	0,174	

Radius Type 1, Cluster Type 2			
n/m	**48**	**96**	**144**
10	0,014	0,028	0,05
25	0.228	0,078	0,144
50	0,05	0,16	

Radius Type 2, Cluster Type 1			
n/m	**48**	**96**	**144**
10	0,032	0,146	0,216
25	0,142	0,414	3,142
50	83,638	148618,604	

Radius Type 2, Cluster Type 2			
n/m	**48**	**96**	**144**
10	0,03	0,162	0,224
25	0,122	1,05	26,64
50	26,938	100031,708	

Radius Type 2, Cluster Type 3			
n/m	**48**	**96**	**144**
10	0,016	0,058	0,11
25	0,034	0,156	0,696
50	0,062	0,436	

Radius Type 2, Cluster Type 4			
n/m	**48**	**96**	**144**
10	0,08	0,224	0,812
25	0,356	2816,05	6,63
50	1018,926		

Radius Type 3, Cluster Type 1			
n/m	**48**	**96**	**144**
10	0,072	1,222	3,664
25	0,72	46,922	790,268
50	8,78		

Radius Type 3, Cluster Type 2			
n/m	**48**	**96**	**144**
10	0,142	1,212	5,94
25	0,944	68,65	594,536
50	7,104		

Radius Type 3, Cluster Type 3			
n/m	**48**	**96**	**144**
10	0,028	0,238	0,536
25	0,058	3,078	21,712
50	0,142		

Radius Type 3, Cluster Type 4			
n/m	**48**	**96**	**144**
10	0,22	2,944	14,782
25	4,878	599,596	21470,892
50	126,368		

Radius Type 4, Cluster Type 1			
n/m	**48**	**96**	**144**
10	0,028	0,08	0,21
25	0,062	0,364	0,624
50	0,158		

Radius Type 4, Cluster Type 2			
n/m	**48**	**96**	**144**
10	0,048	0,174	0,308
25	0,13	0,536	0,956
50	0,234		

Radius Type 4, Cluster Type 3			
n/m	**48**	**96**	**144**
10	0,046	0,132	0,182
25	0,082	0,352	0,794
50	0,18		

Radius Type 4, Cluster Type 4			
n/m	**48**	**96**	**144**
10	0,028	0,106	0,174
25	0,118	0,27	1,276
50	0,114		

Radius Type 5, Cluster Type 1			
n/m	**48**	**96**	**144**
10	0,148	2,036	8,522
25	4,418	120,384	1430,75
50	66,078		

Radius Type 5, Cluster Type 2			
n/m	**48**	**96**	**144**
10	0,126	0,682	3,142
25	0,516	14,204	167,228
50	5,596		

Radius Type 5, Cluster Type 3			
n/m	**48**	**96**	**144**
10	0,042	0,398	0,868
25	0,1	1,306	6,184
50	0,188		

Radius Type 5, Cluster Type 4			
n/m	**48**	**96**	**144**
10	0,252	1,494	5,578
25	2,042	141,1	1944,594
50	60,864		

Radius Type 5, Cluster Type 5			
n/m	**48**	**96**	**144**
10	0,152	1,244	2,288
25	0,566	25,9	136,64
50	7,242		

Radius Type 5, Cluster Type 6			
n/m	**48**	**96**	**144**
10	0,028	0,346	0,606
25	0,102	2,016	6,632
50	0,1		

Radius Type 5, Cluster Type 7			
n/m	**48**	**96**	**144**
10	0,128	0,79	2,48
25	0,96	20,96	157,88
50	3,214		

Radius Type 6, Cluster Type 1			
n/m	**48**	**96**	**144**
10	0,206	1,658	4,216
25	1,834	36,462	179,746
50	23,104		

Radius Type 6, Cluster Type 2			
n/m	**48**	**96**	**144**
10	0,336	2,66	4,746
25	6,218	125,856	782,528
50	48,434		

Radius Type 6, Cluster Type 3			
n/m	**48**	**96**	**144**
10	0,276	1,058	2,082
25	3,27	63,736	139,42
50	10,196		

Radius Type 6, Cluster Type 4			
n/m	**48**	**96**	**144**
10	0,166	0,732	1,834
25	2,696	54,066	350,178
50	7,306		

Radius Type 7, Cluster Type 1			
n/m	**48**	**96**	**144**
10	0,018	0,038	0,08
25	0,026	0,078	0,188
50	0,046		

Radius Type 7, Cluster Type 2			
n/m	**48**	**96**	**144**
10	0,022	0,044	0,08
25	0,032	0,09	0,178
50	0,05		

Radius Type 8, Cluster Type 1			
n/m	**48**	**96**	**144**
10	0,132	0,81	1,566
25	1,424	27,976	55,57
50	1,77		

Radius Type 8, Cluster Type 2			
n/m	**48**	**96**	**144**
10	0,136	0,592	1,688
25	0,83	11,074	3497,314
50	1,348		

Radius Type 8, Cluster Type 3			
n/m	**48**	**96**	**144**
10	0,04	0,194	0,388
25	0,204	1,896	16,396
50	0,382		

Radius Type 8, Cluster Type 4			
n/m	**48**	**96**	**144**
10	0,33	1,334	2,714
25	1,104	28,336	1633,006
50	12,256		

Radius Type 9, Cluster Type 1			
n/m	**48**	**96**	**144**
10	0,13	0,402	1,198
25	0,558	8,026	23,496
50	2,04		

Radius Type 9, Cluster Type 2			
n/m	**48**	**96**	**144**
10	0,094	0,872	1,588
25	1,366	11,14	68,572
50	6,21		

Radius Type 9, Cluster Type 3			
n/m	**48**	**96**	**144**
10	0,104	0,304	1,038
25	0,976	4,968	14,502
50	1,936		

Radius Type 9, Cluster Type 4			
n/m	**48**	**96**	**144**
10	0,106	0,476	1,474
25	0,67	4,004	21,504
50	4,75		

Radius Type 10, Cluster Type 1			
n/m	**48**	**96**	**144**
10	0,058	0,132	0,162
25	0,094	0,334	0,442
50	0,116		

Radius Type 10, Cluster Type 2			
n/m	**48**	**96**	**144**
10	0,062	0,134	0,268
25	0,16	0,448	0,618
50	0,154		