MELİKŞAH UNIVERSITY
KAYSERİ

The Graduate Institute of Sciences and Engineering

M.Sc. Thesis in Electrical and Computer Engineering

ON A GREEDY HEURISTIC FOR THE MULTICOMMODITY RENT-OR-BUY PROBLEM

by

Osman Melih KÜRTÜNCÜ

July 2014
Kayseri, Turkey

ON A GREEDY HEURISTIC FOR THE MULTICOMMODITY RENT-OR-BUY PROBLEM

by

Osman Melih KÜRTÜNCÜ

A thesis submitted to

the Graduate Institute of Sciences and Engineering

of

Melikşah University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Electrical and Computer Engineering

July 2014
Kayseri, TURKEY

APPROVAL PAGE

This is to certify that I have read the thesis entitled "On a Greedy Heuristic for the Multicommodity Rent-or-Buy Problem" by Osman Melih KÜRTÜNCÜ and that in my opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science in Electrical and Computer Engineering, the Graduate Institute of Science and Engineering, Melikşah University.

____

July 3, 2014 Asst. Prof. Dr. Ali ÇİVRİL
Supervisor

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

____

July 3, 2014 Prof. Dr. Murat UZAM
Head of Department

<u>Examining Committee Members</u>

Title and Name		Approved
Asst. Prof. Dr. Ali ÇİVRİL	July 3, 2014	__________________
Assoc. Prof. Dr. Ahmet UYAR	July3, 2014	__________________
Assoc. Prof. Dr. Emel KIZILKAYA AYDOĞAN	July3,2014	__________________

It is approved that this thesis has been written in compliance with the formatting rules laid down by the Graduate Institute of Science and Engineering.

____

Prof. Dr. M. Halidun KELEŞTİMUR
Director

July 2014

ON A GREEDY HEURISTIC FOR THE MULTICOMMODITY RENT-OR-BUY PROBLEM

Osman Melih KÜRTÜNCÜ

M.S. Thesis – Electrical and Computer Engineering
July 2014

Supervisor: Asst. Prof. Dr. Ali ÇİVRİL

ABSTRACT

This thesis introduces three new algorithms for an important network design problem called the Multicommodity Rent-or-Buy Problem which is a generalization of the famous Steiner Forest Problem. These algorithms are inspired by the well-known minimum spanning tree algorithms of Kruskal, Prim and Boruvka. Although our algorithms do not have good approximation ratio compared to the state-of-art, we show that they are much faster than the well-known approximation algorithm of Agrawal, Klein and Ravi (AKR) with similar solution costs, especially when the edge weights span a wide range. In particular, our algorithms turn out to be a very good alternative for AKR on real world data, where for example the points to be connected in the problem represents the cities of a country on the Euclidean plane.

The running time of our algorithms for the Steiner Forest Problem is $O((m + n\log n)k)$ which is an improvement over the previous $(2 - {}^{1}/_{k})$ approximate algorithm with $O(n^2 \log n)$ running time where m, n and k are the number of edges, vertices and terminal pairs in the graph respectively.

Keywords: Multicommodity Rent-or-Buy Problem, Steiner Forest Problem, Sample and Augment algorithm, Strictness, Greedy Heuristics, Approximation Algorithms

ÇOKLU EŞYA SATIN AL YA DA KİRALA PROBLEMİNE AÇ GÖZLÜ BİR SEZGİSEL

Osman Melih KÜRTÜNCÜ

Yüksek Lisans Tezi – Elektrik ve Bilgisayar Mühendisliği
Temmuz 2014

Tez Yöneticisi: Yrd. Doç. Dr. Ali ÇİVRİL

ÖZ

Bu tez, ünlü Steiner Ormanı Probleminin genelleştirilmiş bir hali ve önemli bir ağ tasarım problemi olan Çoklu Eşya Kirala veya Satın Al Problemi için üç yeni algoritma öne sürmektedir. Bu algoritmalar Kruskal, Boruvka ve Prim'in iyi billinen minimum yayılan ağaç algoritmalarından esinlenmişlerdir. Bizim algoritmalarımızın son geliştirilen algoritmalara kıyasla kötü olmasına rağmen, bizim algoritmalarımızın özellikle kenar ağırlıklarının yüksek aralıklarla değiştiği çizgelerde iyi bilinen Agrawal, Klein ve Ravi'nin (AKR) algoritmasından çok daha hızlı çalıştığını ve ona benzer sonuç verdiğini gösterdik. Özellikle, algoritmalarımız Öklid düzlemde bir ülkenin şehirlerini birbirine bağlayan gerçek dünya verisi için çok iyi bir alternatif teşkil etmektedirler. .

Algoritmalarımızın Steiener Ormanı problemi için çalışma zamanları $O((m + n\log n)k)$ olup $(2 - {}^{1}/_{k})$ yaklaşık ve çalışma zamanı $O(n^2 \log n)$ olan bir önceki algoritmaya göre daha iyidir ki burada m, n ve k sırası ile çizgedeki köşe, düğüm ve terminal çiftlerinin sayısıdır.

Anahtar Kelimeler: Çoklu Eşya Kirala veya Satın Al Problemi, Steiner Ormanı Problemi, Açgözlü Algoritmalar, Yaklaştırma Algoritmaları

DEDICATION

Dedicated to my family for their endless support and patience during the forming phase of this thesis.

ACKNOWLEDGEMENT

I would like to thank to The Scientific and Technological Research Council of Turkey (TÜBITAK) that supported this thesis under the project number 112E192.

I would like to express my gratitude to my supervisor Assoc. Prof. Dr. Ali ÇİVRİL whose help, stimulating suggestions and encouragement helped me during the course of research and writing of this thesis.

I also want to thank Teaching Assistant Bilge Kağan DEDETÜRK for his significant contributions to my research endeavors.

I express my thanks and appreciation to my family for their understanding, motivation and patience. Lastly, but in no sense the least, I am thankful to all of my colleagues and friends who believe in me about my researches

TABLE OF CONTENTS

LIST OF TABLES

TABLE

LIST OF FIGURES

FIGURE

CHAPTER 1

INTRODUCTION

In the past few decades, combinatorial optimization has been one of the most important paradigms in scientific research. In this field, one is required to find an optimal solution with respect to an objective function from a finite set of object which defines the solution space. However, it is difficult to find an optimal solution for many important combinatorial optimization problems such as finding shortest/cheapest round trips (TSP), planning, scheduling, time tabling, internet data packet routing to name a few. In fact, many combinatorial optimization problems which arise in various areas of computer science and mathematics are NP-hard which means that no polynomial-time algorithm is possible for these problems unless the widely believed conjecture $P \neq NP$.

One of the main approaches to tackle with these difficult problems is to give up the requirement to find an exact solution and find an approximate solution in polynomial time. This is the most popular approach taken in computer science and the field of approximation algorithms provides the following definition in this framework.

Definition 1.1: An α-approximation algorithm where α is called approximation ratio or approximation factor is a polynomial time algorithm whose value is within α of the optimal solution's value for all instances of the problem [2]. For minimization problems $\alpha > 1$ while $\alpha < 1$ for maximization problems.

This definition guarantees that the algorithm at hand finds an approximate solution for all instances of the problem. This fact makes the design of approximation algorithms a challenging endeavor and such algorithms might be complicated with a high running time. On the other hand, it is also well-known that simple heuristics that do not have a proven approximation ratio might provide reasonably good results on random instances and real world data. This thesis describes three new algorithms for an important network design problem called the Multicommodity Rent-or-Buy Problem which is a generalization of the famous Steiner Forest Problem. These algorithms are inspired by the well-known minimum spanning tree algorithms of Kruskal, Prim and Boruvka. In fact, we show that all of our algorithms are equivalent. We provide instances on which our algorithms perform poorly, i.e. they do not have good approximation ratios. However, they are very simple and easy to implement. Furthermore, the quality of the solutions they return are as good as the best approximation algorithm for the problem and their running time are much better than the approximation algorithm on instances where the edges weights span a wide range. In particular, our algorithms turn out to be a very good alternative for real world data, where the points to be connected in the problem are represented by cities of a country on the Euclidean plane.

CHAPTER 2

PROBLEMS CONSIDERED IN THIS THESIS

2.1 THE MULTICOMMODITY RENT-OR-BUY PROBLEM

In the Multicommodity Rent-or-Buy Problem (MRoB), we are given a weighted graph (i.e. $G = (V, E)$ with a cost function $w : E \rightarrow \mathcal{R}^{+}$) together with k terminal pairs $R = \{(s_1, t_1), \dots, (s_k, t_k)\}$ where $s_i, t_i \in V$ for $i = 1, \dots, k$. We are also given a positive demand d_i for each terminal pair $(s_i, t_i) \in R$, and a parameter $M \geq 1$. The goal is to install capacities on the edges of G such that for all $(s_i, t_i) \in R$, we can simultaneously route d_i units of flow on edge e, or we can buy infinite capacity on an edge at cost $M. w(e)$.

As for the importance of the MRoB Problem in real world applications, it plays an important role in approximately solving some network design problems with economies of scale since it is a central special case of the buy-at-bulk network design problem [2]. Thus, it arises in many real world applications. For example, consider an electricity network with sources that produce energy and customers who absorb energy from a specific source as terminal pairs. The produced energy at source must be dispatched to the customers according to their demands. And the capacities of cables which form the edges of the network can be rent or bought with a cost related with capacities. So, the goal is to meet the needs of customers with a minimum expense.

The following is an example for the MRoB Problem: Assume that there is a given undirected graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ with costs $\mathrm{c_e}$ for all $\mathrm{e} \in \mathrm{E}$ and terminal pairs $\mathrm{R} = \{(\mathrm{s_1}, \mathrm{t_1}), (\mathrm{s_2}, \mathrm{t_2})\}$ with positive demands $\mathrm{d_1} = 2$ and $\mathrm{d_2} = 3$ and parameter $M = 4$ as shown in the figure 2.1.1. The goal is to install minimum cost of capacities on edges such that all flows can be routed simultaneously and it can be either rent capacity at cost $\lambda_\mathrm{e}. \mathrm{c_e}$ or buy infinite capacity at cost $\mathrm{M}. \mathrm{c_e}$.

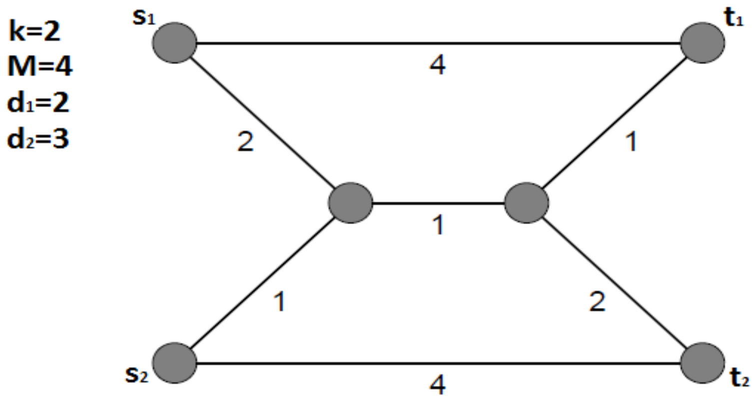

Figure 2.1.1: An example for the MRoB

Since structure of the problem is an optimization problem, there can be some several solutions for that kind of problem. Figure 2.2.2, Figure 2.2.3 and Figure 2.2.4 are just one of the solutions.

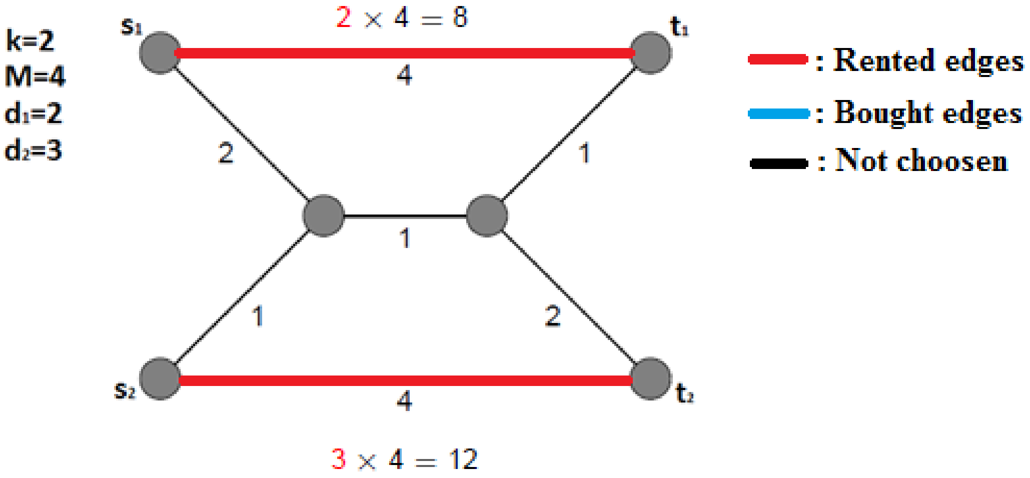

Figure 2.1.2: 1st solution for the example

The solution for the example given in Figure 2.1.1 consists of the red edges which are enough to connect s_1 to t_1 and s_2 to t_2 as shown in Figure 2.2.2. And renting these edges is more profitable than buying because the cost of buying these edges will be $(M \times 4) + (M \times 4) = 32$ while the cost of renting these edges will be $(d_1 \times 4) + (d_2 \times 4) = (2 \times 4) + (3 \times 4) = 20$.

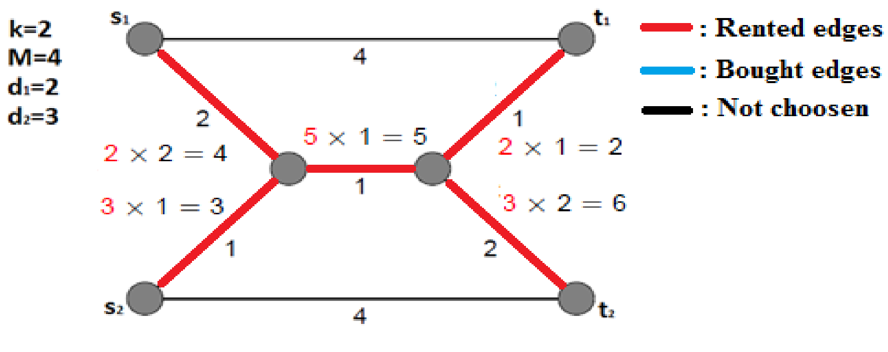

Figure 2.1.3: 2nd solution for the example

Another solution for the example given in Figure 2.1.1 consists of the red edges which are enough to connect s_1 to t_1 and s_2 to t_2 as shown in Figure 2.2.3. And the cost of renting these edges is

$$(d_1 \times 2) + (d_2 \times 1) + \big((d_1 + d_2) \times 1\big) + (d_1 \times 1) + (d_2 \times 2)$$

$$= (2 \times 2) + (3 \times 1) + \big((3 + 2) \times 1\big) + (2 \times 1) + (3 \times 2) = 20$$

However, if the edge with demand (2+3=5) in Figure 2.2.3 is bought as shown in Figure 2.1.4 the solution will be more profitable. This time, the cost will be

$$(d_1 \times 2) + (d_2 \times 1) + (M \times 1) + (d_1 \times 1) + (d_2 \times 2)$$

$$= (2 \times 2) + (3 \times 1) + (4 \times 1) + (2 \times 1) + (3 \times 2) = 19$$

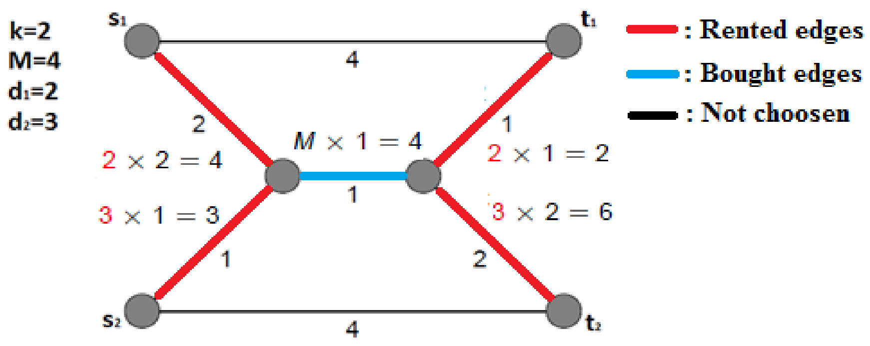

Figure 2.1.4: 3rd solution for the example

MRoB is NP-Hard, and even Max-SNP hard like Steiner Forest which will be discussed in chapter 2.2, since it contains Steiner Tree as a special case [3]. Note that for $M = 1$ and unit demands, this problem reduces to the Steiner Forest Problem. And with a root node r to be simultaneously sent a specified number of flows, it reduces to Single-commodity Rent-or-Buy Problem (SRoB). In addition, another NP hard problem named Multicast Rent-or-Buy Problem (MuRoB) is a generalization of MRoB. In this problem, we are given a set of terminals $R = \{g_1, \dots, g_k\}$ with $g_i \subseteq V$ and $|g_i| \geq 2$ for $1 \leq i \leq k$.The goal is to install capacities on the edges of G so that one can route d_i units of flow between the terminals of every group g_i. If $|g_i| = 2$ for $1 \leq i \leq k$, then MuRoB reduces to MRoB.

The best known performance guarantee for MRoB was the $O(\log|V|\ loglog|V|)$-approximation algorithm by Awerbuch and Azar [4] in 1997 and Bartal [5] in 1998. Later in 2002, Kumar, Gupta and Roughgarden [6] gave the first constant approximation algorithm for this problem. Then, Gupta, Kumar and Roughgarden [7] provide a framework called *sample-and-augment* to give approximation algorithms for a number of network problems including a special case of MRoB. This framework is then generalized to incorporate MRoB by Gupta et al.[8]. The same authors provide the final framework which also applies MuRoB in [9]. The sample-and-augment algorithm for MRoB works as follows:

1. *Sampling*: Select a random subset $S \subseteq R$ of terminal pairs by picking every terminal pair $(s_i, t_i) \in R$ independently with probability $p_i = \min\{\frac{d_i}{M}, 1\}$
2. *Subproblem:* Compute an α-approximate Steiner forest F_s in S and buy all the edges in F_s.
3. *Augmentation:* Augment F_s to a feasible solution for R by renting additional edges to connect all terminal pairs in $R\backslash S$ in the least costly manner.

There is a relationship between the approximation ratio obtained for the MRoB in this framework and what is called *strict cost sharing scheme*. Gupta et al. [8, 9] show that if the Steiner forest algorithm has approximation ratio α and admits β-strict cost share, then the sample and augment algorithm is an $(\alpha + \beta)$-approximate algorithm for the MRoB problem.

The notation from Fleischer et al. [10] is adapted to define strictness of a Steiner forest algorithm for the rest of this thesis. Given a forest F in G, let $G|F$ denote the graph resulting from contracting all trees of F. Let $c_{G|F}(u,v)$ denote the minimum cost of any $u-v$ path in $G|F$. A Steiner forest algorithm A is said to be β-strict for $\beta \geq 1$, if there exists nonnegative cost shares $\xi_{s,t}$ for all $(s,t) \in R$ satisfying the following two conditions:

1. $\sum_{(s,t)\in R} \xi_{s,t} \leq opt_R$, where opt_Rdenotes the minimum cost of a Steiner forest for R.
2. $c_{G|F_{-s,t}} \leq \beta . \xi_{s,t}$ for all $(s,t) \in R$, where $F_{-s,t}$ is a Steiner forest for the terminal set $R_{-s,t} = R \setminus \{(s,t)\}$ returned by A.

The sample-and-augment framework can be adapted to yield $(\alpha+\beta)$-approximate algorithm for the MuRoB problem [7], stochastic Steiner tree (SST) problem in the black-box model [11] and the stochastic Steiner forest (SSF) problem in the independent decision model [12].

Finally, sample and augment framework provides Gupta et al. [8, 9] to improve 12-approximate algorithm to the MRoB problem. Bechetti et al. [13] improved approximation ratio to 6.828. The best approximation algorithm obtained for this problem is due to Fleischer et al. [14] which is 5.

Since MRoB is a generalization of Steiner Forest, we need to examine the Steiner Forest Problem in more detail. In fact, the algorithms that we provide also work for the Steiner Forest Problem and they were inspired as a solution to this were special case.

2.2 THE STEINER FOREST PROBLEM

In the Steiner Forest Problem, which is also known as the Generalized Steiner Tree Problem, we are given an undirected graph $G = (V, E)$ with a cost function on the edges $w : E \rightarrow \mathbb{R}^+$, and a set of k terminal pairs $R = \{(s_1, t_1), \ldots, (s_k, t_k)\} \subseteq V \times V$. The goal is to find a minimum cost subset of edges $F \subseteq E$ such that there is at least one path between each terminal pair (s_i, t_i) in (V, F). Since this problem is both a generalization of the famous Steiner Tree Problem, and a special case of several network design problems such as Survivable Network Problem, it is one of the central problems in the field of approximation algorithms and combinatorial optimization. Indeed, if $t_i = s_{i+1}$ for $i = 1, \ldots, k-1$, the Steiner Forest Problem reduces to Steiner Tree Problem. Moreover, if it is required that there exists r_i edge-disjoint paths between s_i and t_i in (V, F) (where $r_i \in \mathbb{Z}^+$), then Steiner Forest Problem turns into Survivable Network Design Problem.

Since the Steiner Forest Problem is a generalization of Steiner Tree Problem, it is NP-hard and in fact MAX-SNP hard [3, 15, 16]. Starting with the work of Takahashi and Matsuyama [17] which yields a 2-approximation ratio to the problem in 1980, a series of algorithms sequentially improved the ratio to 1,55 [18, 19, 20, 21, 22, 23, 24]. Finally, Byrka et al. [25] achieved 1.39 approximation with a new LP-based algorithm which is the best algorithm obtained thus far as shown in Table 2.1.

Table 2.1: Previous works for Steiner Tree Problem

Year	**Performance Ratio**	**Authors**
1980	2	Takahashi, Matsuyama [17]
1993	1,834	Zelikovsky [18]
1994	1,734	Berman, Ramaiyer [19]
1995	1,694	Zelikovsky [20]
1997	1,667	Prömel, Steger [21]
1997	1,664	Karpinski, Zelikovsky [22]
1998	1,598	Hougardy, Prömel [23]
2005	1,55	Robins, Zelikovsky [24]
2013	1,39	Byrka, Grandoni, Rothvoss, Sanita [25]

Even though many algorithmic improvements were recorded for the Steiner Tree Problem, the same is not true for the Steiner Forest Problem. In fact, there is only one approximation algorithm stated in two different languages. Obtaining a genuinely different approximation algorithm for this problem has been a challenge for the past two decades. Indeed, our attempts towards this thesis were along the lines of a possible such algorithm.

One of the approximation algorithms for the Steiner Forest Problem is stated in purely combinatorial terms by Agrawal, Klein and Ravi [26] while other is parameterized by a certain variable in LP relaxation of a primal-dual approach by Goemans and Williamson [27]. In this approach , it has become customary to express the dual variables as *moats* and the increase in dual variables of [26] as *growing moats*. As usual, we will briefly call this version of algorithm which is stated by Agrawal, Klein and Ravi as AKR. The approximation ratio of AKR is $2-\frac{1}{k}$ and this ratio is tight since LP relaxation is known to be $2-\frac{1}{k}$. Apart from this algorithm, a slightly different algorithm with the same approximation ratio for achieving a game theoretic constraint is introduced by Könemann et al. [28] in 2008. This algorithm which we will briefly call as KLS is also based on natural LP relaxation. The main difference from the previous algorithm is that the moats are growing for an extended period of time in KLS.

CHAPTER 3

PREVIOUS ALGORITHMS FOR THE STEINER FOREST PROBLEM

3.1 ALGORITHMS AKR AND KLS

The standard LP Relaxation of Steiner Forest Problem consists of a variable x_e for each $e \in E$. This variable is 1 if e is in the resulting forest and 0 otherwise. Let $\mathcal{S}$ be the set of subsets S of V that separate at least one terminal pair in R. In other words, $S \in \mathcal{S}$ if and only if there is $(s,t) \in R$ satisfying $|\{s,t\} \cap S| = 1$. Let also $\delta(S)$ denote the set of edges with exactly one endpoint in S. The integer linear programming formulation for the problem is then as follows:

$$\text{minimize} \quad \sum_{e \in E} w_e \,.\, x_e \qquad \text{(IP)}$$

$$\text{subject to} \quad \sum_{e \in \delta(S)} x_e \geq 1, \qquad \forall S \in \mathcal{S},$$

$$x_e \in \{0,1\}, \qquad \forall e \in E.$$

The constraints enforce that for any cut S separating s_i and t_i for some i, it must be selected one edge from $\delta(S)$. If the constraint $x_e \in \{0,1\}$ is dropped and replaced with $x_e \geq 0$ to obtain an LP relaxation, the dual of this linear program is

$$\text{maximize} \quad \sum_{S \in \mathcal{S}} y_S \qquad \text{(D)}$$

$$\text{subject to} \quad \sum_{S \in \mathcal{S}: e \in \delta(S)} y_S \leq w_e, \qquad \forall e \in E,,$$

$$y_S \geq 0, \qquad \forall S \in \mathcal{S}.$$

AKR algorithm considers all connected components C of (V,F) such that $|C \cap \{s_1, t_1\}| = 1$ for some i. Therefore, at the beginning, C is completely S. Then, y_C is uniformly increased for all such connected components until the dual inequality for some $e \in \delta(C')$ becomes tight where C' is some component at the current iteration. This edge is then included in the forest. If all the terminal pairs which are in the component connected, then y_C is not increased for that component in the next iteration. So the iterations are continued like this until all pairs are connected. After all iterations are completed, *reverse-delete step* is performed which excludes the edges from resulting forest in a reverse order of included edges to the resulting forest. For each excluded edge, it checks whether the resulting forest is feasible or not. If it is not feasible, then this edge is again included to the resulting forest.

The difference between AKR and KLS is the period of time for which the set of moats are grown. In AKR, two initial moats m_s and m_t which belong to the terminal pairs (s, t) might not be able to collide each other during the execution because they may collide and unit with other components. However, in KLS, the growing of these moats continues until they meet each other. Hence, the main difference of KLS from AKR is that the moats corresponding to a specific pair are grown as if the other terminal pairs do not exist.

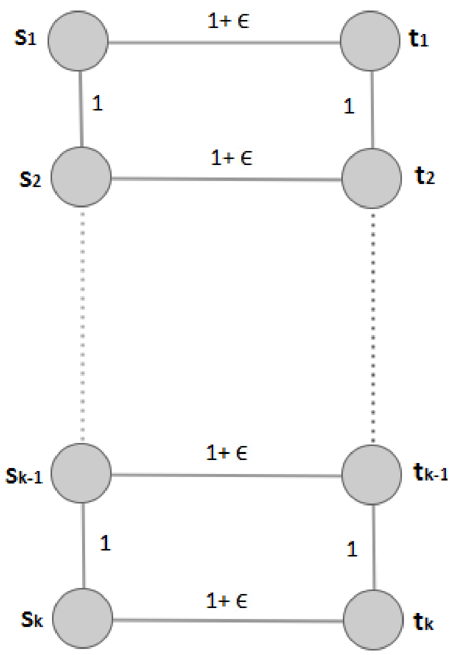

Figure 3.1.1: A tight example for AKR

The instance given in Figure 3.1.1 is tight for AKR since optimal solution equals to $(1 + \epsilon k)$ and the solution returned by AKR has cost $(2k - 1 + \epsilon)$ which means that when ϵ goes to 0, the approximation ratio of AKR is $\frac{2k-1+\epsilon}{k+k\epsilon} = 2 - 1/k$. The optimum takes all the edges of costs $(1+\epsilon)$ as shown in Figure 3.1.2. On the other hand, AKR first takes the edges of costs 1 since the grown moats first collide on these edges. Then, it complete execution with taking one of the edge of cost $(1+\epsilon)$ since all of the terminal pairs are connected. The solution returned by this algorithm is shown in Figure 3.1.3.

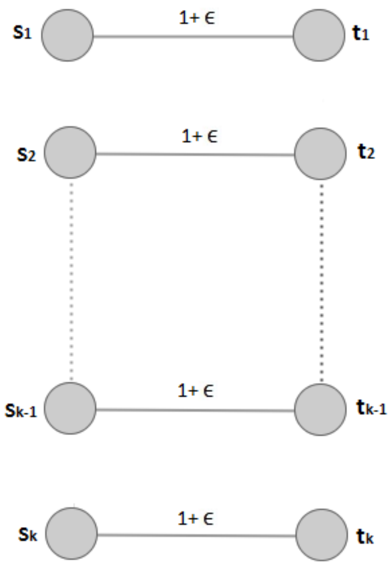

Figure 3.1.2: Optimum solution of the given instance

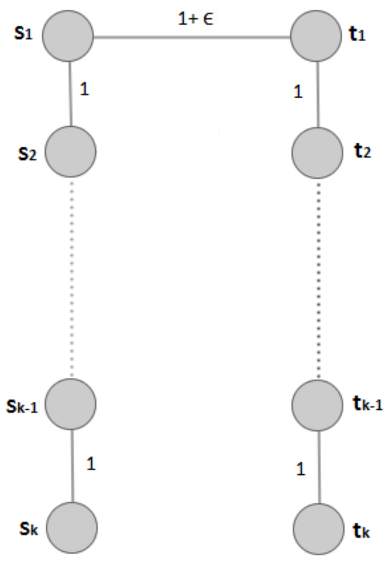

Figure 3.1.3: The forest returned by AKR

3.2 A GREEDY HEURISTIC

Even though the algorithms given up to now have a 2-approximation ratio for the Steiner Forest Problem, i.e. AKR and KLS, a proverbial greedy algorithm which has worse approximation ratio already exists. This algorithm is stated as ‘Greedy’ for the rest of this thesis. Although it is known that this algorithm has a 2-approximation ratio for the Steiner Tree Problem, this can not be said for the Steiner Forest Problem. In fact, there is an example that shows this algorithm is worse than AKR, i.e. its approximation ratio is greater than 2. In this section, we briefly overview this heuristic and provide an instance on which the cost of the solution it gives is 4 times as large as the optimum.

Starting from $s_1 - t_1$ pair, Greedy Algorithm finds the shortest path between them. Then this path is included to the resulting forest and the pair is contracted which means the length of the path is zeroed out. This computation performed iteratively up to $s_k - t_k$.

As example to how good the Greedy Heuristic can perform on a graph, consider the graph which has a total of $k^2/2$ terminal pair and every terminal pairs are adjacent to each other with a cost of $4 - \epsilon$ as given in Figure 3.2.1.

In this example, the cost of optimum solution is $k^2 + 2k$. More specifically, it is all of the solid edges with cost 2 and 1 as given in Figure 3.2.2. However, Greedy takes all the adjacent edges of terminal pairs with a cost of $(4 - \epsilon)k^2$ which is four times as large as the optimum as shown in Figure 3.2.3 while AKR Algorithm finds a solution with a cost of $k^2 + (k - 1)(4 - \epsilon)$ as shown with red edges in Figure 3.2.4.

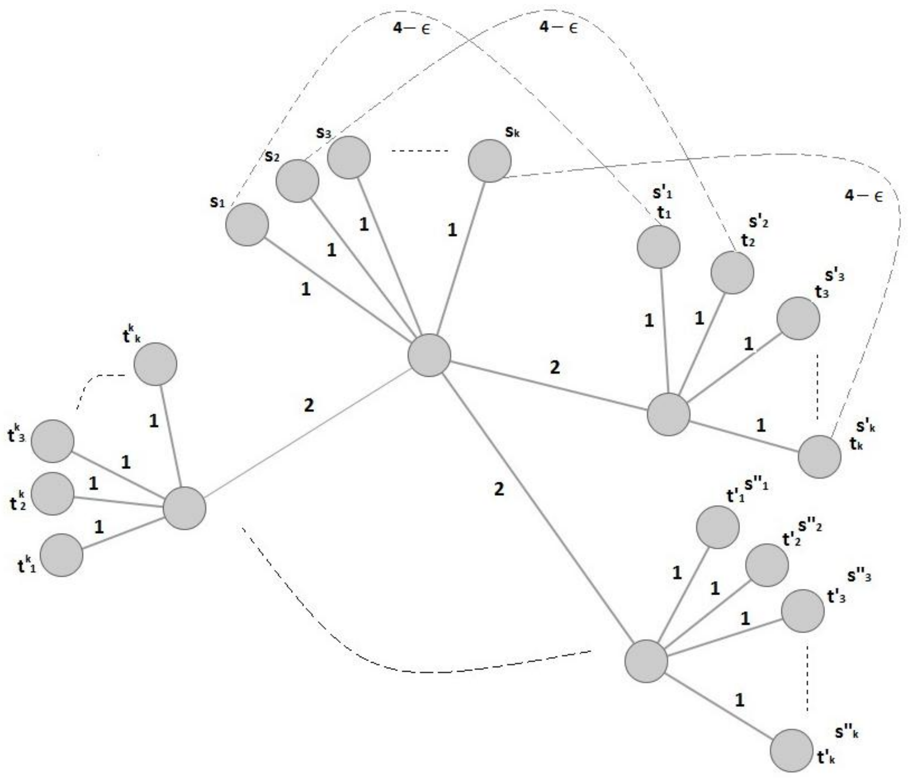

Figure 3.2.1: A sample graph

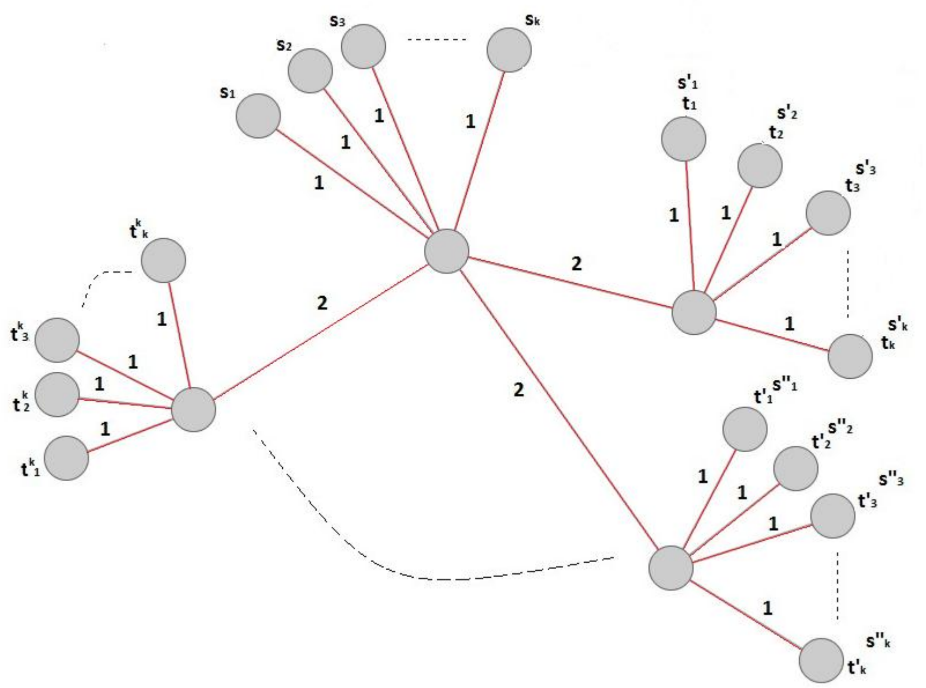

Figure 3.2.2: The optimum solution of sample graph with red edges

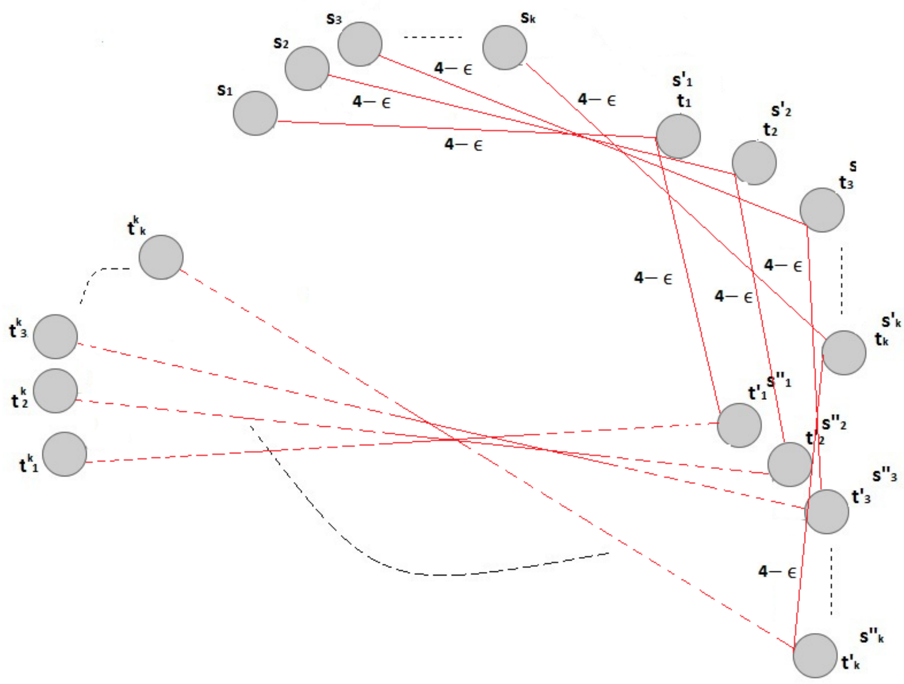

Figure 3.2.3: The solution of the Greedy Heuristic

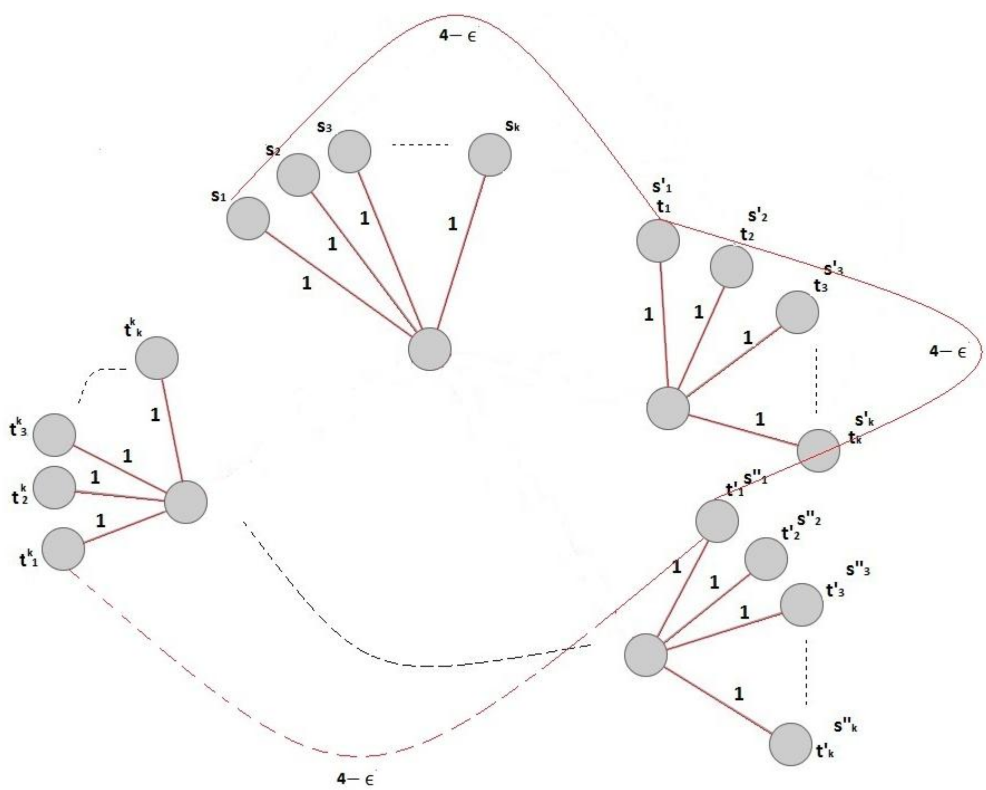

Figure 3.2.4: The solution of AKR

CHAPTER 4

THE NEW ALGORITHMS

In this chapter, three equivalent new algorithms for the Steiner Forest Problem and the MRoB Problem are given. These algorithms are simpler than AKR in that they are extensives of basic greedy algorithms for finding minimum spanning trees. They are appropriately named as ÇDK-Kruskal, ÇDK Prim and ÇDK-Boruvka since they are analogues of the well-known algorithms of Kruskal [29], Prim [30] and Boruvka [31]. To be more specific, we run our algorithms on an adjunct graph *H* which is derived from the input graph *G*. Even though our algorithms are quite similar to the algorithms of Kruskal, Prim and Boruvka in spirit, a necessary modification is required by the structure of *H*. Since these algorithms provide a basis for our approach and the notation, they are stated in section 4.1 The notation from [32] is used in the statement of these algorithms as well as in the statements of our new algorithms.

4.1 THE ALGORITHMS OF KRUSKAL, BORUVKA AND PRIM

Each algorithm of Kruskal, Boruvka and Prim provide to find a minimum spanning tree of a graph *G* in different ways. The pseudocodes for these algorithms are given in Algorithm 1, Algorithm 2 and Algorithm 3, respectively.

Kruskal's algorithm non-decreasingly sorts the edges and processes all of sorted edges iteratively starting from the edge which has the smallest weight. Then the current edge is added to the final list if it does not form a new cycle in the graph. In order to check whether adding this edge forms a new cycle or not, disjoint set structure is used.

At the beginning of Kruskal's algorithm, a set is created for each vertex $v \in V$ by the *MAKE-SET* command. Additionally *FIND-SET* command checks whether the set associated with two vertices are identical. *UNION* command takes a union of the sets associated with the two vertices given as parameters. Boruvka's algorithm is similar to Kruskal's algorithm. The difference is that the cheapest edges that are going out of each set are considered at each step. At the final iteration, algorithm merges the sets appropriately and continues until just one set remains. Completely different from Kruskal's and Boruvka's algorithm, Prim's algorithm starts processing edges rom a root vertex r and greedily grows this single set until it contains all the vertices.

Algorithm 1 Kruskal's algorithm to find a minimum spanning tree of a graph G

---

procedure KRUSKAL($G = (V, E), w: E \longrightarrow \mathbb{Q}^+$)
  $F \leftarrow \emptyset$
  for each vertex $v \in V$ **do**
    $MAKE - SET(v)$
  Sort the edges of E in non-decreasing order by w
  for each edge $(u, v) \in E$, taken in non-decreasing order by w **do**
    if $FIND - SET(u) \neq FIND - SET(v)$ **then**
      $F \leftarrow F \cup \{(u, v)\}$
      $UNION(u, v)$
  return A

Algorithm 2 Boruvka's algorithm to find a minimum spanning tree of a graph G

procedure BORUVKA($G = (V, E), w: E \longrightarrow \mathbb{Q}^{+}$)
  $F \leftarrow \emptyset$
  for each vertex $v \in V$ **do**
    $MAKE - SET(v)$
  while there are more than 1 set **do**
    for each set S **do**
      $F_S \leftarrow \emptyset$
      for each vertex u in S **do**
        $F_s \leftarrow F_s \cup$ {the cheapest edge (u, v)such that
          $FIND - SET(u) \neq FIND - SET(v)$ }
      $F \leftarrow F \cup$ {the cheapest edge (u_s, v_s) in F_s }
    for each set S **do**
      $UNION(u_s, v_s)$
  return A

Algorithm 3 Prim's algorithm to find a minimum spanning tree of a graph G

procedure PRIM($G = (V, E), w: E \longrightarrow \mathbb{Q}^{+}, r$)
  $F \leftarrow \emptyset$
  $S \leftarrow MAKE - SET(r)$
  while $|S| < |V|$ **do**
    $F' \leftarrow \emptyset$
    for each vertex u in S **do**
      $F' \leftarrow F' \cup$ {the cheapest edge (u, v) such that
        $FIND - SET(u) \neq FIND - SET(v)$}
    $F \leftarrow F \cup$ {the cheapest edge (u', v') in F'}
    $UNION(u', v')$
  return A

4.2 INTUITION FOR THE NEW ALGORITHMS

The algorithms of ÇDK-Kruskal, ÇDK-Boruvka and ÇDK-Prim are stated in purely combinatorial terms and do not use the language imposed by the LP relaxation to the Steiner Forest Problem. ÇDK-Kruskal imitates Kruskal's Algorithm by first computing all the shortest paths between terminal pairs and sorts them in non-decreasing order with respect to their weighted lengths. Then, if the endpoints of the paths are not in the same set, it includes them in to the solution in this order. After a path is included into the solution, a union operation is performed between the sets corresponding to the terminals which are the endpoints of the path. This is different from both AKR and KLS. During the execution of these algorithms, the edges that are not on the shortest path might be included to the solution. ÇDK-Kruskal ensures that the structure between two terminals remains as a path since the unnecessary edges which are excluded in the reverse delete step of AKR are not included in ÇDK-Kruskal. However, our new algorithms might include multiple copies of edges since the shortest paths between terminal pairs might intersect. Note that it is possible to exclude the duplicate edges that are found by ÇDK-Kruskal and so decrease the cost of the forest with an overhead in running time.

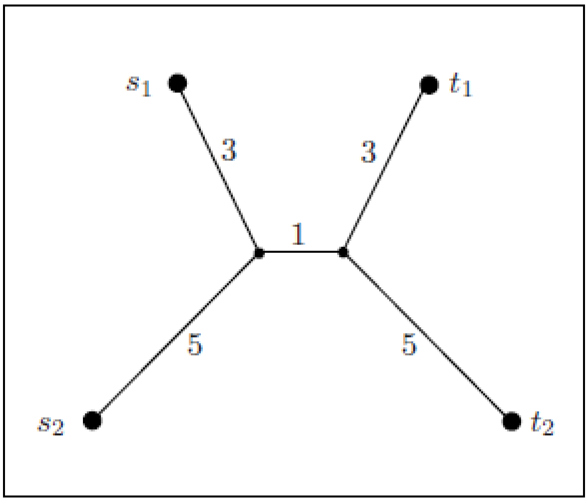

Figure 4.2: A graph on which the compared algorithms return different solutions

The difference between the algorithms mentioned in this thesis is best seen on an example. Consider the graph in Figure 4.2. Obviously, the cost of the optimal and the feasible solution is 17. AKR takes all the edges and finds the optimal cost. However, KLS Algorithm takes the edges of costs 3 and 5 once, and the edge with the cost 1 twice since it continues to grow moats m_{s2} and m_{t2} which belong to the pair(s_2, t_2) even if they collide with the moats m_{s1} and m_{t1} of the pair (s_1, t_1) before meeting each other. Hence, the edge with cost 1 is included into solution twice: once for m_{s1} and m_{t1} and once for m_{s2} and m_{t2}. Therefore, the cost returned by KLS is 18. On the other hand, Greedy first finds the shortest path between s_1 and t_1 and then contracts them. In the next iteration, it finds the shortest path between s_2 and t_2 in the graph where s_1 and $t_1$are contracted with the shortest path between them. So, the forest returned by Greedy is just a node where all the terminal pairs are contracted and the cost of solution is 17, same as AKR. ÇDK-Kruskal computes all the 6 shortest paths between 4 terminals and processes them in a greedy manner starting from the least cost path. And the cost of the forest returned by ÇDK-Kruskal is $w(s_1, t_1) + w(s_1, s_2) + w(t_1, t_2) = 7 + 8 + 8 = 23$.

4.3 THE NEW ALGORITHMS ÇDK-KRUSKAL, ÇDK-BORUVKA AND ÇDK-PRİM

As mentioned before, all of our new algorithms run on an adjunct graph *H* which is derived from the input graph *G* and represents all the shortest paths between terminals as edges. More specifically, each vertex in the vertex set of *H* corresponds to a terminal and edges are shortest paths between these terminals. Hence, there are $2k$ vertices and $\binom{2k}{2}$ edges in *H*.

After *H* is computed, taking into account of the shortest paths of the original graph *G*, our algorithms construct a forest by processing the edges of *H*. Even though our algorithms are similar to the minimum spanning tree algorithms, they are different since the computed graph is a complete graph and connecting terminal pairs is enough to terminate the algorithm instead of a full connection of all the vertices. Therefore, the solution returned by our algorithms is a forest.

Computing *H* determines the running time of our algorithms since the running time of the remaining part of the algorithm which is performed on a graph of $2k$ vertices is asymptotically smaller than the running time of computing *H*. We use Dijktra’s shortest path algorithm for all the terminals to compute *H*. This takes $O((m +$

$n \log n)k)$ using a Fibonacci Heap where m is the number of edges, n is the number of vertices and k is the number of terminal pairs.

The pseudocodes for our new algorithms are given in Algorithm 4, Algorithm 5 and Algorithm 6.

Algorithm 4 Algorithm ÇDK-Kruskal to find a Steiner forest in a graph G

---

procedure ÇDK-Kruskal($G = (V, E), R = \{s_1, t_1, \dots, s_k, t_k\}, w: E \longrightarrow \mathbb{Q}^+, r$)
    $F \leftarrow \emptyset$
    for each $r \in R$ **do**
        $MAKE - SET(r)$
    $H = (V', E', w') \leftarrow COMPUTEADJUNCT(G, R, w)$
    $\sum |1..\binom{2k}{2}|$ ← sort the edges of E' in non-decreasing order by w'
    for $i \leftarrow 1$ to $\binom{2k}{2}$ **do**
        Let $\sum |i|$ be between $u \in R$ and $v \in R$
        if $FIND - SET(u) \neq FIND - SET(v)$ **then**
            $F \leftarrow F \cup$ {the set of edges of $\sum |i|$}
            $UNION(u, v)$
        if all s_i and t_i are connected via F **then**
            break
    for $k \leftarrow l$ down to 1 **do**
        if $F' - p_k$ is a feasible solution **then**
            remove p_k from F
    return F

Algorithm 5 Algorithm ÇDK-BORUVKA to find a Steiner forest in a graph G

procedure ÇDK-Boruvka($G = (V, E), R = \{s_1, t_1, \dots, s_k, t_k\}, w: E \longrightarrow \mathbb{Q}^+$)
  $F \leftarrow \emptyset$
  for each $r \in R$ **do**
    $MAKE - SET(r)$
  $H = (V', E', w') \leftarrow COMPUTEADJUNCT(G, R, w)$
  $\sum |1..\binom{2k}{2}| \leftarrow$ sort the edges of E' in non-decreasing order by w'
  while not all pairs (s_i, t_i) are connected via F **do**
    for each set S **do**
      $F_S \leftarrow \emptyset$
      for each vertex u S **do**
        $F_s \leftarrow F_s \cup$ {the cheapest edge (u, v) in Σ such that $FIND - SET(u) \neq FIND - SET(v)$}
      $A \leftarrow A \cup$ {the cheapest edge (u_S, v_S) in F_S}
    for each set S **do**
      $UNION(u_s, v_s)$
  for $k \leftarrow l$ down to 1 **do**
    if $F' - p_k$ is a feasible solution **then**
      remove p_k from F
  return F

Algorithm 6 Algorithm ÇDK-PRIM to find a Steiner forest in a graph G

procedure ÇDK-Prim($G = (V, E), R = \{s_1, t_1, \dots, s_k, t_k\}, w: E \longrightarrow \mathbb{Q}^+, r$)
  $F \leftarrow \emptyset$
  $S \leftarrow MAKE - SET(r)$
  $H = (V', E', w') \leftarrow COMPUTEADJUNCT(G, R, w)$
  $\sum |1..\binom{2k}{2}| \leftarrow$ sort the edges of E' in non-decreasing order by w'
  while not all pairs (s_i, t_i) are connected via F **do**
      $F' \leftarrow \emptyset$
      for each vertex u S **do**
        $F' \leftarrow F' \cup$ {the cheapest edge (u, v) in Σ such that

$FIND - SET(u) \neq FIND - SET(v)\}$

$F \leftarrow F \cup$ {the cheapest edge (u', v') in F'}

$UNION(u', v')$

for $k \leftarrow l$ down to 1 **do**

if $F' - p_k$ is a feasible solution **then**

remove p_k from F

return F

4.4 EQUIVALENCE OF THE NEW ALGORITHMS

In this part, a lemma that claims given a full ordering of the edges of H with respect to their weights, ÇDK-Kruskal, ÇDK Boruvka and ÇDK-Prim algorithms return the same set of edges on H.

Lemma 4.1. Given a full ordering of the edges of H, ÇDK-Kruskal, ÇDK Boruvka and ÇDK-Prim are equivalent.

Proof. Let $E' = \{e_1,\ e_2,\ \ldots, e_{\binom{2k}{2}}\}$ be the set of the edges in increasing order. We argue by induction on ℓ, the number of the edges selected by ÇDK-Kruskal algorithm during its execution. For $\ell = 1$, ÇDK-Kruskal selects the smallest weighted edge e_1 and let $e_1 = (u, v)$. Consider ÇDK-Boruvka and ÇDK-Prim algorithms are at a situation where u and v are not in the same set and the edges that are adjacent to these nodes are considered. So at this stage, by the nature of these algorithms, the edge e_1 will be included in the forest since it is the smallest weighted edge. Thus, the base case of the induction is formed. Assume, as the induction hypothesis that, the set $F_\ell = \{e_1, e_{i_2}, \ldots, e_{i_\ell}\}$ has been already selected before ÇDK-Kruskal selects $(\ell + 1)$st edge and all the other edges up to e_{i_ℓ} excluding the edges in F_ℓ are not selected, and the set of selected and unselected edges are the same for ÇDK-Boruvka and ÇDK-Prim. If there is an edge, say $e = \{u, v\}$, between e_{i_ℓ} and $e_{i_{\ell+1}}$in ordering, then e is not selected by ÇDK-Kruskal since it creates a cycle, i.e. u and v are in the same set. Also the edge e will not be selected by ÇDK-Boruvka and ÇDK-Prim since u and v will be same set for these algorithms via the edges in F_ℓ by induction hypothesis. Otherwise, these algorithms will not select one of the edges in F_ℓ which contradicts the induction hypothesis. All that remain is to show that $e_{i_{\ell+1}}$ is selected by ÇDK-Boruvka and ÇDK-Prim. Let $e_{i_{\ell+1}} = (u, v)$ where u and v are not in the same set and then consider these algorithms are in a stage that they consider the edges adjacent to u and v. The edge $e_{i_{\ell+1}}$ will be selected since this edge is the smallest weighted edge which does not create a cycle by the choice of ÇDK-Kruskal. Finally, the termination conditions are also equivalent since the condition is a full connection of terminal pairs. Therefore the proof of the Lemma 4.1 is completed.

4.5 A BAD INSTANCE FOR THE NEW ALGORITHMS

Consider the graph given in Figure 4.5.1. All of the edges between the terminal pairs have a cost ϵ and the edges between unpaired terminals have cost 1 except the edge that connects terminal pairs s_k to t_k. This edge has a cost 3 while the cost of the edges that connect s_k and t_k to other terminals are 2.

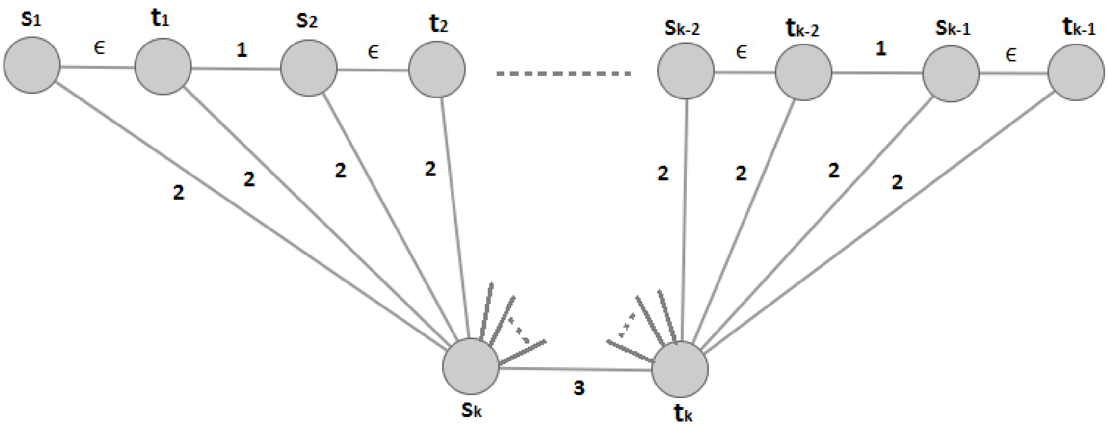

Figure 4.5.1: A sample graph

Clearly, the optimum solution is the adjacent edges to the terminal pairs as shown in Figure 4.5.2. Thus, it has a cost $\epsilon(k-1)+3$

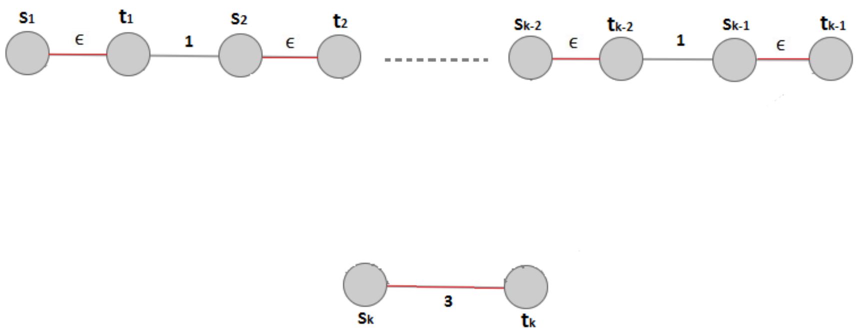

Figure 4.5.2: Optimum solution of given instance

Consider the computation of the ÇDK-Kruskal Algorithm. It first sorts the paths between terminals. Then, the edges that have cost ϵ are included to the resulting forest since all the terminals are not in the same set. In the next step, the algorithm includes the edges that have cost 1 to connect the remaining terminal pair $s_k - t_k$ since these edges have the smallest weight. Since the included edges are not enough to connect s_k to t_k, algorithm includes the next two smallest weight edges to the resulting forest which have cost 2. Therefore, the computation of ÇDK-Kruskal is completed as shown in Figure 4.4.3. The solution returned by the algorithm has cost $\epsilon(k-1)+(k-2)+4$ which means that the approximation ratio of our algorithms is k since $\frac{\epsilon(k-1)+(k-2)+4}{\epsilon(k-1)+3} = k$ when ϵ goes to 0.

The instance given in this section shows that the approximation ratio of our algorithms is not constant, i. e. it depends on k. Thus, even if we can find a small strictness (β) for our new algorithms since their approximation factor depends on k for the Steiner Forest Problem, the same thing holds for the MRoB Problem. This is because the approximation factor for MRoB is $\alpha + \beta$, where β is the strictness and α is the approximation factor for Steiner Forest.

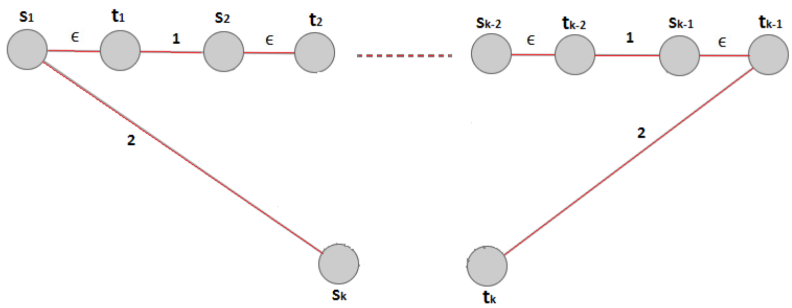

Figure 4.5.3: The solution of ÇDK-Kruskal for given instance

4.6 AN EXECUTION OF THE NEW ALGORITHMS ON AN MROB INSTANCE

In this section, we explain the new algorithms on an example. Consider the graph given in Figure 4.6.1 where the number of terminal pairs $k = 3$ and each node is represented with an integer. The demands for each terminal pair are unit ($d_i = 1$) for all i where $i = \{1,2,3\}$ and $M = 3$. The goal is to determine minimum-cost capacity installation such that all demands can be routed simultaneously.

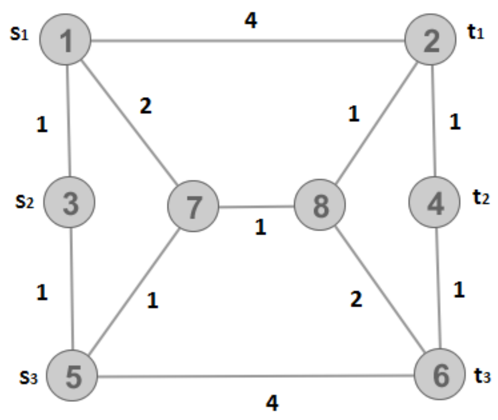

Figure 4.6.1: An MROB instance

We use the sample and augment framework of Gupta et.al [9] which is mentioned in section 2.1 to solve the given instance. By the definition of this framework, in sampling step, a random subset of terminal pairs is determined by picking every terminal pair with a probability of $1/3$. Thus, assume that the terminal pairs $s_1 - t_1$ and $s_2 - t_2$ are picked in this step as shown in Figure 4.6.2. In the subproblem step, we run the Steiner Forest Algorithm on the terminal pairs picked in sampling step and buy the edges returned by the algorithm. This step plays an important role in solving the MRoB Problem since the Steiner Forest computed in this step determines the solution of the problem. For this reason, we show the execution of both ÇDK-Kruskal and AKR for the instance defined in this section.

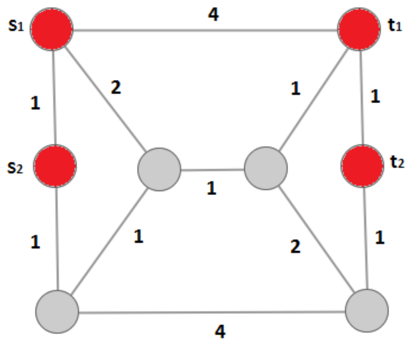

Figure 4.6.2: The picked terminal pairs in subproblem step of the Sample and Augment Framework

By the definition of ÇDK-Kruskal, it creates sets for each terminal and then computes an adjunct graph by computing $\binom{4}{2} = 6$ shortest paths as shown in Figure 4.6.3. It then sorts all the shortest paths in the adjunct graph H in non-decreasing order by their weighted lengths as shown in Table 4.6.1. In addition, with this order it checks whether the end points of the paths are in the same set.

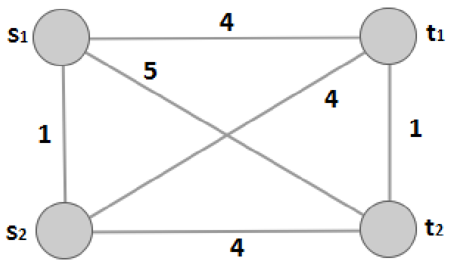

Figure 4.6.3: The adjunct graph H

Table 4.6.1: The sorted shortest paths between each terminal in H

	Node1	Node2	Distance
1	s_1	s_2	1
2	t_1	t_2	1
3	s_2	t_1	4
4	s_1	t_1	4
5	s_2	t_2	4
6	s_1	t_2	5

ÇDK-Kruskal first checks whether s_1 and s_2 terminals are in the same set since $s_1 - s_2$ path is the shortest path in H. Thus, it takes this path to the resulting forest and performs a union of the sets associated with these vertices as shown in Figure 4.6.4.

Figure 4.6.4: 1st iteration of ÇDK-Kruskal after computing H

Then, the algorithm checks whether t_1 and t_2 terminals are in the same set. It also adds $t_1 - t_2$ path to the solution and performs a union of the sets associated with these terminals as shown in Figure 4.6.5.

Figure 4.6.5: 2nd iteration of ÇDK-Kruskal after computing H

In the next step, the algorithm checks whether s_2 and t_1 are in the same set. Since they are not in the same set, it adds $s_2 - t_1$ path to the resulting forest and performs a union the sets associated with these terminals as shown in Figure 4.6.6.

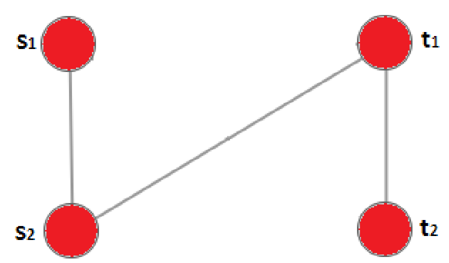

Figure 4.6.6: 3rd iteration of ÇDK-Kruskal after computing H

Finally, all the terminal pairs are connected via the shortest path between the terminals (s_1, s_2), (t_1, t_2) and (s_2, t_1) with a cost of $1 + 1 + 4 = 6$. The shortest path between s_2 and t_1 consists of $3 - 5 - 7 - 8 - 2$ path. Thus, the resulting forest is given in Figure 4.6.7 with the red edges. Consequently, sample and augment buys the forest returned by the algorithm with a cost of $M * 6 = 3 * 6 = 18$.

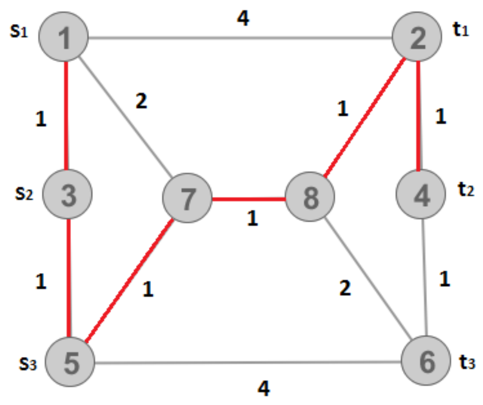

Figure 4.6.7: The forest returned by ÇDK-Kruskal

In the augmentation step, we need to augment the forest returned by the Steiner Forest algorithm to a feasible solution for all terminal pairs by renting additional edges to connect the terminal pairs which are not chosen in the sampling step. Hence, in this instance, we need to augment the forest returned by ÇDK-Kruskal to feasible solution by renting additional edges to connect s_3 to t_3 in the least costly manner. Thus, the blue edge between t_3 and t_2 is rented with a cost of $d_3 * 1 = 1 * 1 = 1$ as shown in Figure 4.6.8. So the final cost is $18 + 1 = 19$ for this MRoB instance when ÇDK-Kruskal is used in subproblem step of the sample and augment framework.

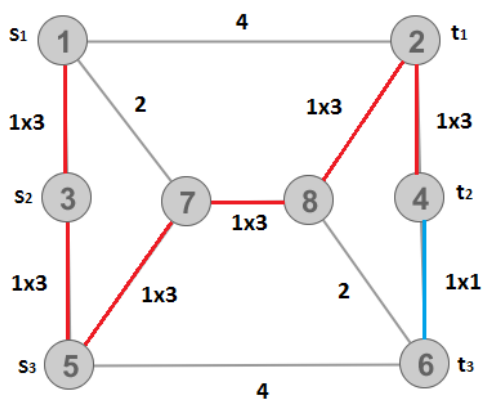

Figure 4.6.8: The solution of MRoB instance using ÇDK-Kruskal

In the subproblem step of sample and augment, AKR starts by growing moats for all terminals chosen in sampling step as shown in Figure 4.6.9. In the first step, the growth of moats is 0,5 units since the shortest distance between two components is1.

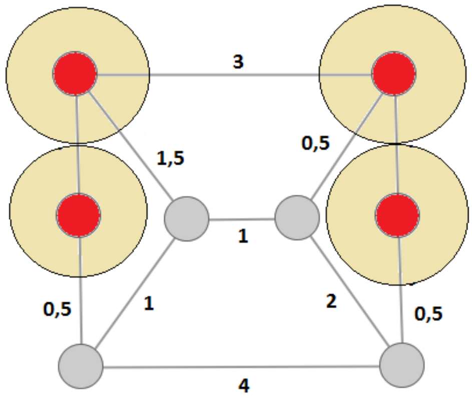

Figure 4.6.9: Growth of moats

AKR includes the edges $(s_{1,}s_2)$ and $(t_{1,}t_2)$ since the sum of the span of moats covers these edges as shown in Figure 4.6.10 and performs a union of the tight components corresponding to these vertices.

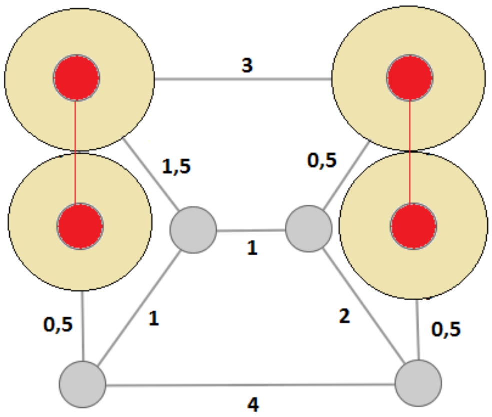

Figure 4.6.10: Including the edges corresponding to tight components

Then, AKR continues to grow the moats by 0,5 and then 1 unit as the same manner and again it includes the edges corresponding to the tight components as shown with red in Figure 4.6.11. So, AKR takes all the edges $(1,7), (7,8), (8,2), (3,5), (5,7), (1,2), (6,8), (4,6)$ simultaneously. However, in the reverse delete step it excludes the edges $(1,7), (7,8), (8,2), (3,5), (5,7), (4,6), (8,6)$.

Thus, the forest returned by AKR which is shown in Figure 4.6.12 by red edges is the $3 - 1 - 2 - 4$ path with a cost of $1 + 4 + 1 = 6$.

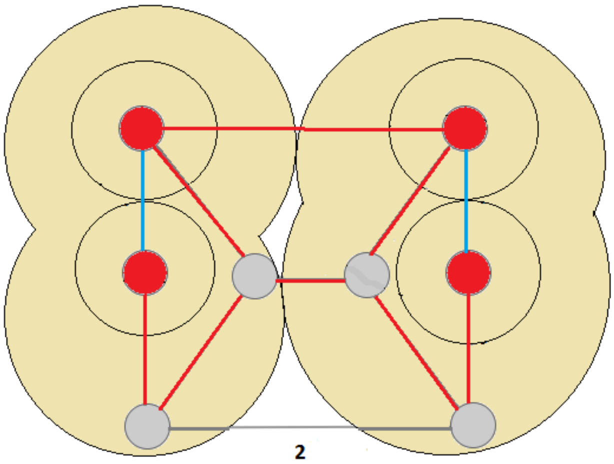

Figure 4.6.11: 2nd iteration of AKR

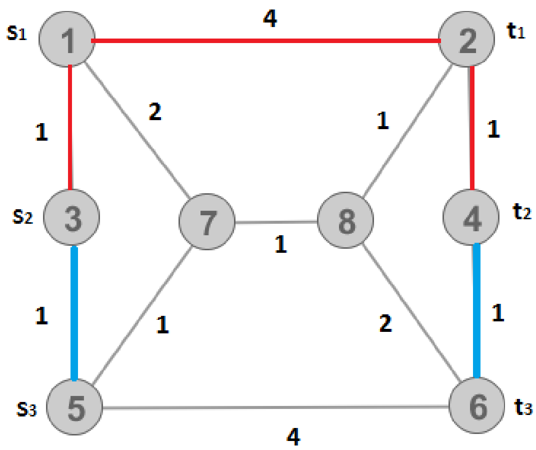

Figure 4.6.12: The forest returned by AKR and augment step

Consequently, sample and augment buys the forest returned by the algorithm with a cost of $M * 6 = 3 * 6 = 18$. Now, we need to augment the forest returned by AKR to feasible solution by renting additional edges to connect s_3 to t_3 in the least costly manner. Thus, (s_2, s_3) and (t_2, t_3) edges are rented with a cost of $(d_3 * 1) + (d_3 * 1) = (1 * 1) + (1 * 1) = 2$ as shown in Figure 4.6.12 by the blue edges. So the final cost is $18 + 2 = 20$ for this MRoB instance when AKR is used in subproblem step of the sample and augment framework.

CHAPTER 5

EXPERIMENTS

In this chapter, the performance of AKR, Greedy and ÇDK-Kruskal for the Multicommodity Rent-or-Buy Problem are compared on a set of test graphs. We use JAVA on a computer which has Intel (R) Core (TM) i5-3470 CPU @ 3.20 GHz, 4.00 GB RAM and 64 bit operating system. All of the algorithms are applied to three types of graphs:

1. Random Graphs of Erdöş-Renyi model where the probability of having an edge between two nodes is a constant $0 < p \leq 1$.
2. Real World Geometric Graphs which is obtained from TSP National Collection data that can be downloaded from [33]. This data consists of 734 cities of Uruguay as nodes and the distances between each city as weights. In the rest of the thesis, this graph is called as TSP Uruguay Graph. We have tried all the algorithms on several other real world data from the same source and the results were similar.
3. Random Geometric Graphs where the nodes are randomly chosen points on the Euclidean plane in a square shaped area, and there is an edge between two nodes if the distance between them is smaller than some specified value.

Each generated random graph in the experiments has 1000 nodes and a variable p where p is the probability that there is an edge between a pair of nodes. Moreover, according to demands of every $s - t$ terminal pairs d_i and cost of edges w_i, two types of random graphs are defined: Random demand-Unit weight, Random demand-Random weight for $p = \{0.1, 02, 0.4, 0.6, 0.8, 1\}$ and for the number of terminal pairs $k = \{2, 3, 5, 10, 20, 30, 40, 50\}$. Note that if $p = 1$, the graph turns out to be a complete graph. TSP Uruguay Graph is defined as TSP Uruguay Graph with Random demands for a value $m * y$ which determines whether there is an edge between two cities in the graph where m is the maximum distance between two cities and $y = \{0.1, 02, 0.4, 0.6, 0.8, 1\}$. If the distance between two cities is smaller than or equal to $m * y$, the edge corresponding to this distance is included to TSP Uruguay Graph. Moreover, according to TSP data, the maximum distance between two cities, m, equals 5746,907405630345. However, for a better computation, all the weights are rounded in our experiment. So m equals 5747 for TSP Uruguay in our experiments. We have observed that if the weights are in double precision, the running time of AKR is exceedingly high, probably due to the excessive number of updates performed in floating point arithmetic. Similarly, Random Geometric Graphs have 1000 nodes with randomly weighted edges between 1 and 1000 with the same y and k values.

The reason why we are separating graphs according to demands and weights as random and unit is that we would like to see various computational results since the demand value of the terminal pairs may result in a different forest. The computational results that we get with unit demands are similar to graphs with random demands. Thus, we just state the experiments with the graphs with random demands.

A total of 144 experiments on random graphs, 48 experiments on the real world graph and 48 experiments on a geometric random graph are performed to test AKR, ÇDK-Kruskal and Greedy Heuristic for the Multicommodity Rent-or-Buy Problem.

This chapter consists of five parts. Each part represents 48 experiments. First three parts show the computational results for random graphs while the last two show the computational result for TSP Uruguay Graph and Geometric Random Graph. The framework of Gupta et al. [27], sample and augment algorithm, is used to compute all of the algorithms.

5.1 EXPERIMENTS ON RANDOM GRAPHS WITH RANDOM DEMAND-UNIT WEIGHT

In this experiment, a random graph with 1000 nodes and unit weighted edges, $w_i = 1$, is generated for each p and k where $p = \{0.1, 02, 0.4, 0.6, 0.8, 1\}$ is the probability that there is an edge between a pair of nodes and $k = \{2, 3, 5, 10, 20, 30, 40, 50\}$ is the number of terminal pairs in the definition of

MRoB. In addition, demands for each terminal are randomly generated between 1 and 5 and $M = 5$. Therefore, in the sampling step, every terminal pair is picked with a probability of $(\frac{d_i}{5})$. A total of $|k| * |p| = 6 * 8 = 48$ experiments are performed in this section. The computational results for each p are given as follows:

Table 5.1.1: Computational results on Random Graphs with Random demand-Unit weight for $p = 0.1$

Unit Weight (wi=1)						
Random Demand (di=1-5) and M=5						
Probability that there is an edge between a pair of nodes= 0.1						
k	RESULT			RUNNING TIME (ms)		
	AKR	CDK_Kruskal	Greedy	AKR	CDK_Kruskal	Greedy
2	30	30	20	624	125	156
3	30	34	24	109	110	46
5	55	55	40	250	187	109
10	70	70	70	78	312	124
20	147	152	127	281	468	219
30	236	236	243	109	842	265
40	258	258	259	188	889	437
50	372	372	375	172	1357	483

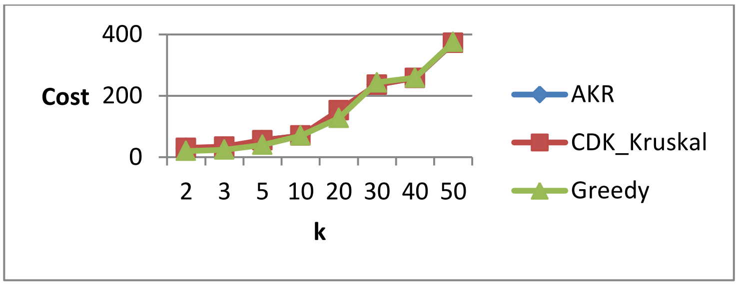

Figure 5.1.1: Cost comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Unit weight for $p = 0.1$

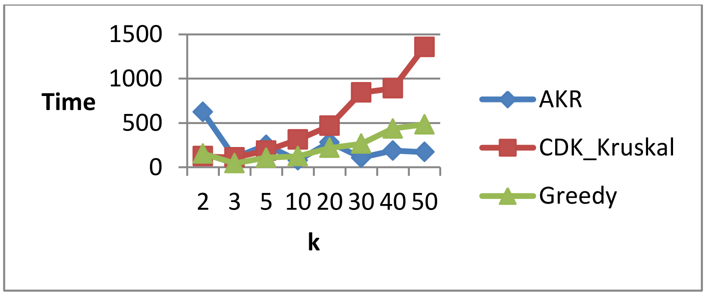

Figure 5.1.2: Running time comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Unit weight for $p = 0.1$

Table 5.1.2: Computational results on Random Graphs with Random demand-Unit weight for $p = 0.2$

Unit Weight (wi=1)						
Random Demand (di=1-5) and M=5						
Probability that there is an edge between a pair of nodes= 0.2						
k	RESULT			RUNNING TIME (ms)		
	AKR	CDK_Kruskal	Greedy	AKR	CDK_Kruskal	Greedy
2	20	25	20	140	125	47
3	18	18	18	124	63	141
5	24	24	24	125	109	156
10	77	77	72	328	421	156
20	146	146	136	140	1030	452
30	239	239	234	172	1607	390
40	307	307	292	484	1950	764
50	360	360	316	390	2745	827

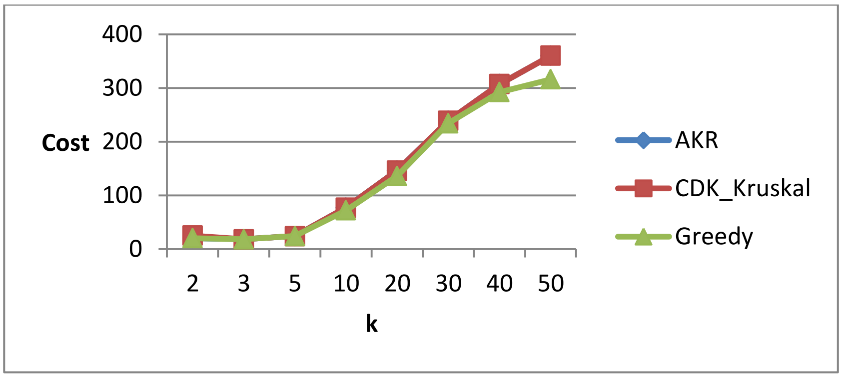

Figure 5.1.3: Cost comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Unit weight for $p = 0.2$

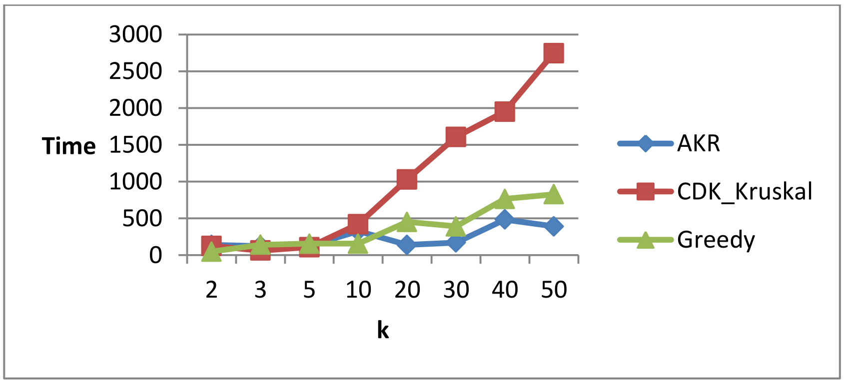

Figure 5.1.4: Running time comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Unit weight for $p = 0.2$

Table 5.1.3: Computational results on Random Graphs with Random demand-Unit weight for $p = 0.4$

Unit Weight (wi=1)						
Random Demand (di=1-5) and M=5						
Probability that there is an edge between a pair of nodes=0.4						
k	RESULT			RUNNING TIME (ms)		
	AKR	CDK_Kruskal	Greedy	AKR	CDK_Kruskal	Greedy
2	20	25	20	1045	327	110
3	17	17	17	281	187	125
5	38	38	33	390	437	1420
10	63	63	63	187	874	484
20	153	153	126	936	2746	655
30	185	205	194	453	3666	951
40	293	293	257	546	5148	1216
50	364	364	306	624	6474	1732

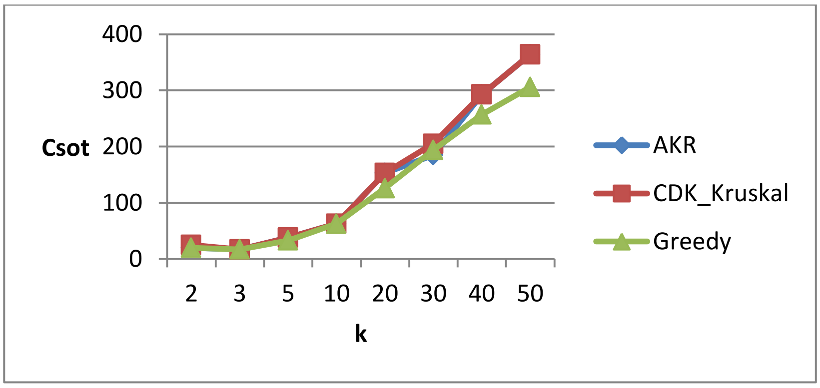

Figure 5.1.5: Cost comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Unit weight for $p = 0.4$

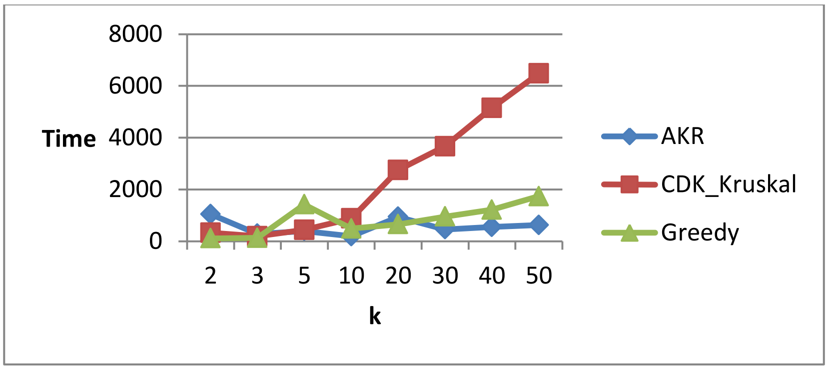

Figure 5.1.6: Running time comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Unit weight for $p = 0.4$

Table 5.1.4: Computational results on Random Graphs with Random demand-Unit weight for $p = 0.6$

Unit Weight (wi=1)						
Random Demand (di=1-5) and M=5						
Probability that there is an edge between a pair of nodes= 0.6						
k	RESULT			RUNNING TIME (ms)		
	AKR	CDK_Kruskal	Greedy	AKR	CDK_Kruskal	Greedy
2	12	12	12	827	374	406
3	19	19	19	312	1107	359
5	36	36	31	125	1762	406
10	58	58	48	624	3666	1076
20	153	153	123	453	6676	1560
30	214	214	176	764	9111	2823
40	253	253	211	1701	10062	2901
50	385	385	297	639	9953	2200

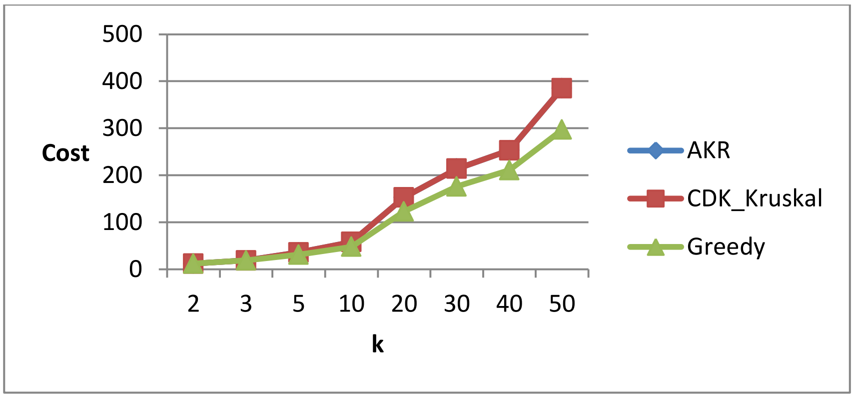

Figure 5.1.7: Cost comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Unit weight for $p = 0.6$

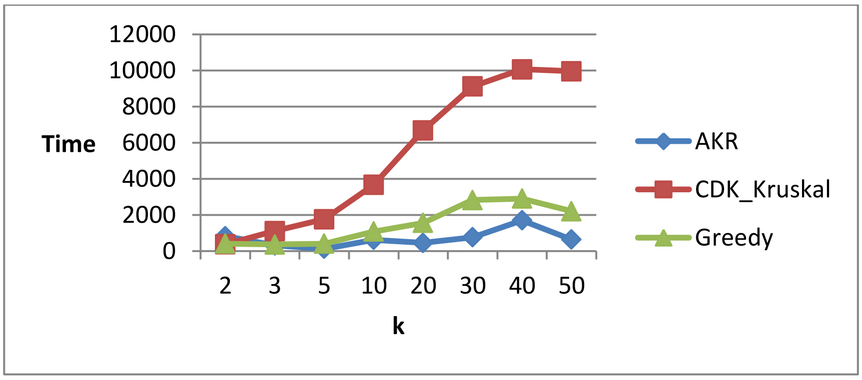

Figure 5.1.8: Running time comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Unit weight for p=0.6

Table 5.1.5: Computational results on Random Graphs with Random demand-Unit weight for $p = 0.8$

Unit Weight (wi=1)						
Random Demand (di=1-5) and M=5						
Probability that there is an edge between a pair of nodes= 0.8						
k	RESULT			RUNNING TIME (ms)		
	AKR	CDK_Kruskal	Greedy	AKR	CDK_Kruskal	Greedy
2	7	7	7	110	265	125
3	17	17	12	125	639	203
5	45	45	30	156	1716	328
10	55	55	35	406	1840	609
20	122	122	77	577	4040	1124
30	199	204	124	764	6692	1701
40	288	288	187	1201	13588	5600
50	367	362	252	1466	14492	3744

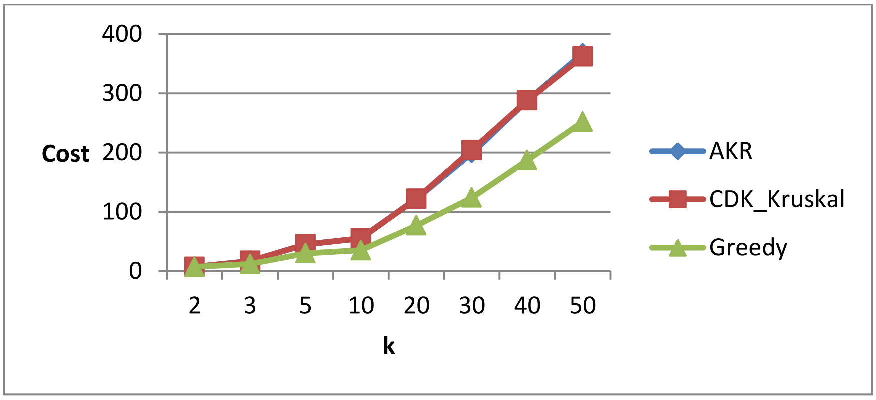

Figure 5.1.9: Cost comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Unit weight for $p = 0.8$

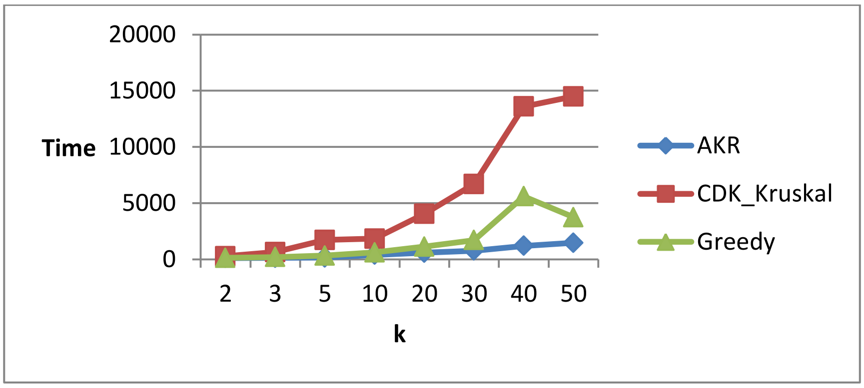

Figure 5.1.10: Running time comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Unit weight for $p = 0.8$

Table 5.1.6: Computational results on Random Graphs with Random demand-Unit weight for $p = 1$

Unit Weight (wi=1)						
Random Demand (di=1-5) and M=5						
Probability that there is an edge between a pair of nodes = 1						
k	RESULT			RUNNING TIME (ms)		
	AKR	CDK_Kruskal	Greedy	AKR	CDK_Kruskal	Greedy
2	6	6	6	172	109	156
3	11	11	11	171	422	218
5	45	45	25	62	1342	406
10	95	95	50	62	3417	842
20	121	121	76	592	4134	1311
30	207	207	117	765	6832	1716
40	230	230	135	1295	8736	2512
50	289	289	174	1684	9906	3370

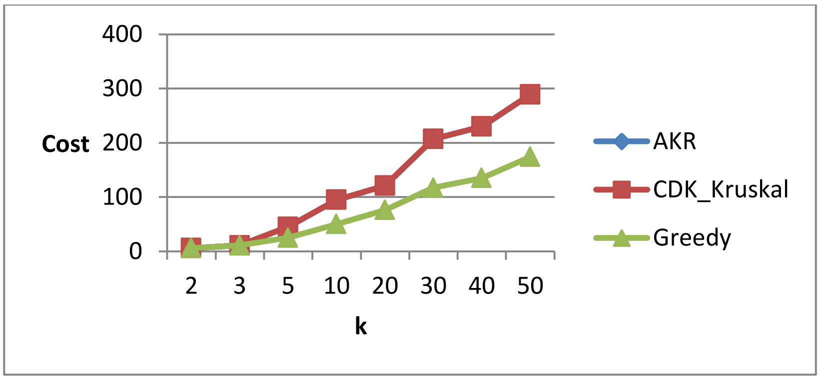

Figure 5.1.11: Cost comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Unit weight for $p = 1$

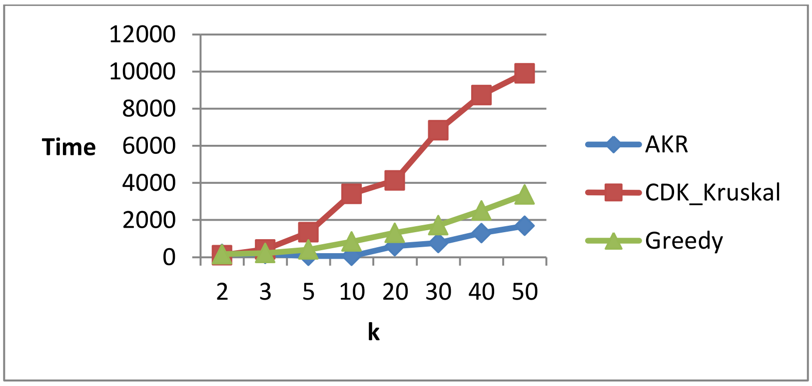

Figure 5.1.12: Running time comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Unit weight for $p = 1$

The experiments over the generated random graphs show that Greedy gives better cost results while weights of all edges are unit and demands of terminal pairs are randomly assigned. In addition, the results of ÇDK-Kruskal and AKR are close to each other. In many cases, AKR gives slightly better results and the differences between the results become more significant as the probability of having an edge between two nodes increases. More specifically, AKR has better running time compared to both ÇDK-Kruskal and Greedy since unit weights cause all the moats to collide during the first few iterations. Besides, the running time of Greedy is better than ÇDK-Kruskal, because ÇDK-Kruskal is computing an adjunct graph which takes time when it is compared to finding shortest path.

One of the main results of this experiment is that when edge weights span a narrow range, AKR runs much faster than ÇDK-Kruskal and Greedy since all the moats grown by AKR collide during the first few iterations and the result is immediately returned.

5.2 EXPERIMENTS ON RANDOM GRAPHS WITH RANDOM DEMAND-RANDOM WEIGHT (Wi=1-100)

In this experiment, a random graph with 1000 nodes and random edge costs between 1 and 100 for each vertex are generated for each p and k where $p = \{0.1, 02, 0.4, 0.6, 0.8, 1\}$ is the probability that there is an edge between a pair of nodes and $k = \{2, 3, 5, 10, 20, 30, 40, 50\}$ is the number of terminal pairs in the definition of MRoB. In addition, demands for each terminal are randomly generated between 1 and 5 and $M = 5$. Therefore, in the sampling step, every terminal pair is picked with a probability of $d_i/5$. A total of $|k| * |p| = 6 * 8 = 48$ experiments are performed in this section. The computational results for each p are given as follows:

Table 5.2.1: Computational results on Random Graphs with Random demand-Random weight (wi=1-100) for $p = 0.1$

Random Weight (wi=1-100)						
Random Demand (di=1-5) and M=5						
Probability that there is an edge between a pair of nodes = 0.1						
k	RESULT			RUNNING TIME (ms)		
	AKR	CDK_Kruskal	Greedy	AKR	CDK_Kruskal	Greedy
2	145	145	135	640	93	172
3	205	215	185	187	125	47
5	195	200	180	93	172	93
10	120	430	365	93	328	140
20	705	765	721	156	500	312
30	1053	1113	1005	219	889	312
40	1307	1377	1366	296	1108	561
50	1582	1693	1656	515	1233	561

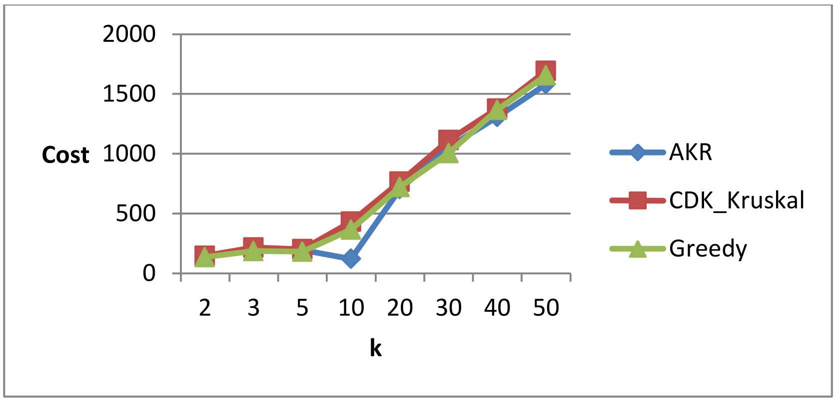

Figure 5.2.1: Cost comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Random weight (wi=1-100) for $p = 0.1$

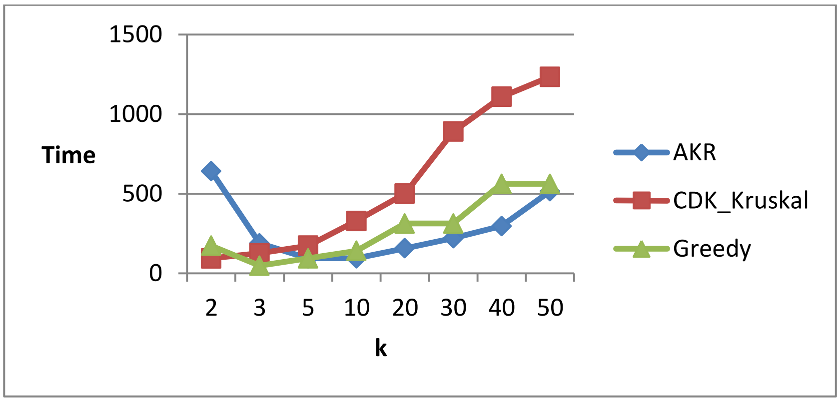

Figure 5.2.2: Running time comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Random weight (wi=1-100) for $p = 0.1$

Table 5.2.2: Computational results on Random Graphs with Random demand-Random weight (wi=1-100) for $p = 0.2$

Random Weight (wi=1-100)						
Random Demand (di=1-5) and M=5						
Probability that there is an edge between a pair of nodes = 0.2						
k	RESULT			RUNNING TIME (ms)		
	AKR	CDK_Kruskal	Greedy	AKR	CDK_Kruskal	Greedy
2	85	100	80	94	140	47
3	107	112	82	141	203	78
5	126	136	106	109	218	172
10	267	289	244	140	484	234
20	514	544	483	468	1061	468
30	637	695	652	359	1576	468
40	896	961	959	468	2075	858
50	1038	1096	1098	468	3198	858

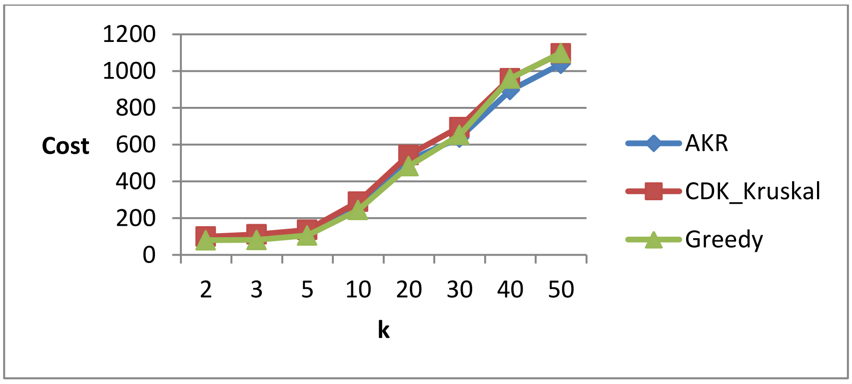

Figure 5.2.3: Cost comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Random weight (wi=1-100) for $p = 0.2$

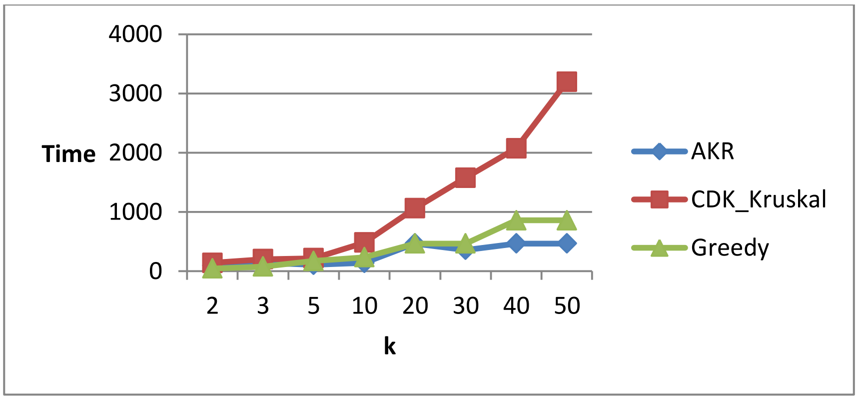

Figure 5.2.4: Running time comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Random weight (wi=1-100) for $p = 0.2$

Table 5.2.3: Computational results on Random Graphs with Random demand-Random weight for $p = 0.4$

Random Weight (wi=1-100)						
Random Demand (di=1-5) and M=5						
Probability that there is an edge between a pair of nodes = 0.4						
k	RESULT			RUNNING TIME (ms)		
	AKR	CDK_Kruskal	Greedy	AKR	CDK_Kruskal	Greedy
2	16	16	16	109	63	78
3	75	95	65	188	561	141
5	118	118	98	156	546	172
10	168	178	164	250	1185	375
20	359	394	361	437	2589	655
30	461	480	466	608	3323	982
40	691	723	700	764	4805	1357
50	755	804	786	968	5865	1576

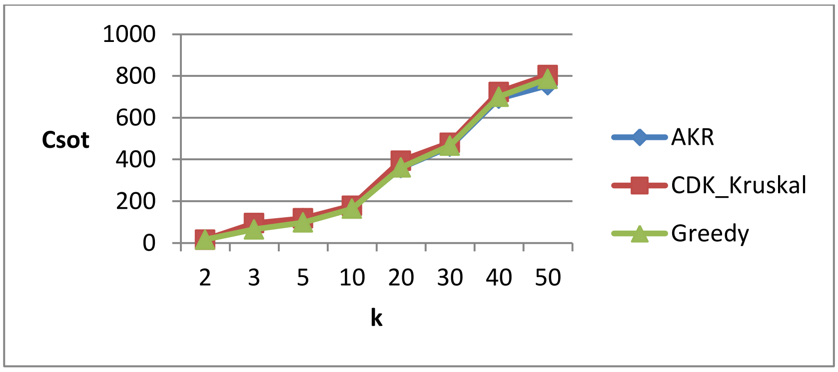

Figure 5.2.5: Cost comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Random weight (wi=1-100) for $p = 0.4$

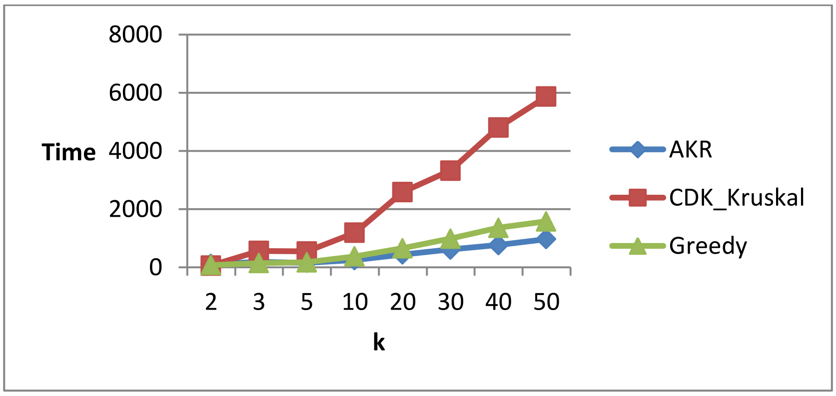

Figure 5.2.6: Running time comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Random weight (wi=1-100) for $p = 0.4$

Table 5.2.4: Computational results on Random Graphs with Random demand-Random weight (wi=1-100) for $p = 0.6$

Random Weight (wi=1-100)						
Random Demand (di=1-5) and M=5						
Probability that there is an edge between a pair of nodes = 0.6						
k	RESULT			RUNNING TIME (ms)		
	AKR	CDK_Kruskal	Greedy	AKR	CDK_Kruskal	Greedy
2	45	45	40	172	515	140
3	42	42	42	172	280	172
5	110	120	90	218	999	249
10	165	165	140	327	1451	499
20	332	327	287	421	3744	1061
30	426	428	413	765	5273	1326
40	567	569	565	1061	6006	1809
50	666	691	730	1107	8424	2481

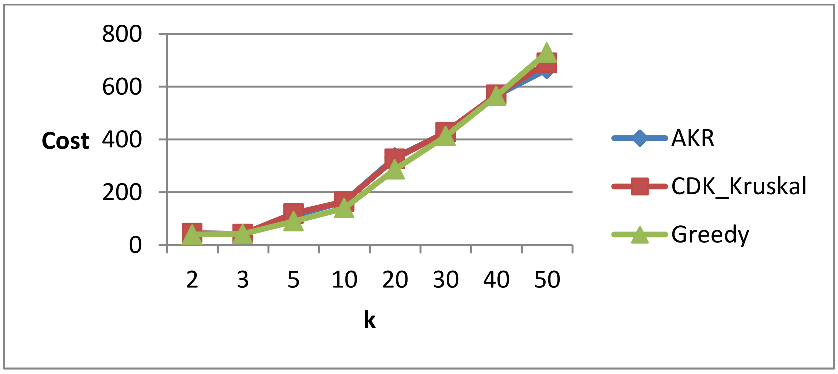

Figure 5.2.7: Cost comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Random weight (wi=1-100) for $p = 0.6$

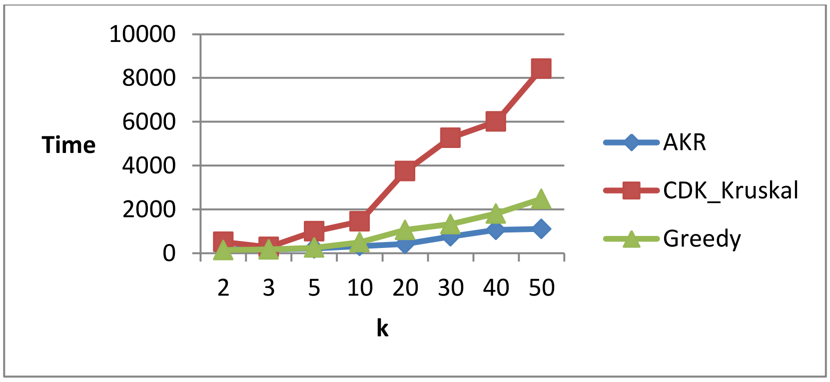

Figure 5.2.8: Running time comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Random weight (wi=1-100) for $p = 0.6$

Table 5.2.5: Computational results on Random Graphs with Random demand-Random weight (wi=1-100) for $p = 0.8$

Random Weight (wi=1-100)						
Random Demand (di=1-5) and M=5						
Probability that there is an edge between a pair of nodes = 0.8						
k	RESULT			RUNNING TIME (ms)		
	AKR	CDK_Kruskal	Greedy	AKR	CDK_Kruskal	Greedy
2	40	45	35	62	546	141
3	27	27	27	348	394	218
5	85	85	62	374	1014	336
10	186	216	156	202	3318	672
20	315	330	260	733	4836	1123
30	395	408	360	952	6224	1935
40	486	503	511	1186	7675	2184
50	574	603	605	1872	11159	3104

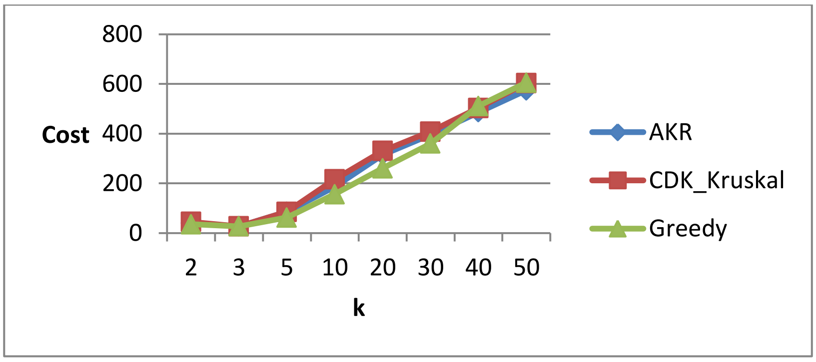

Figure 5.2.9: Cost comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Random weight (wi=1-100) for $p = 0.8$

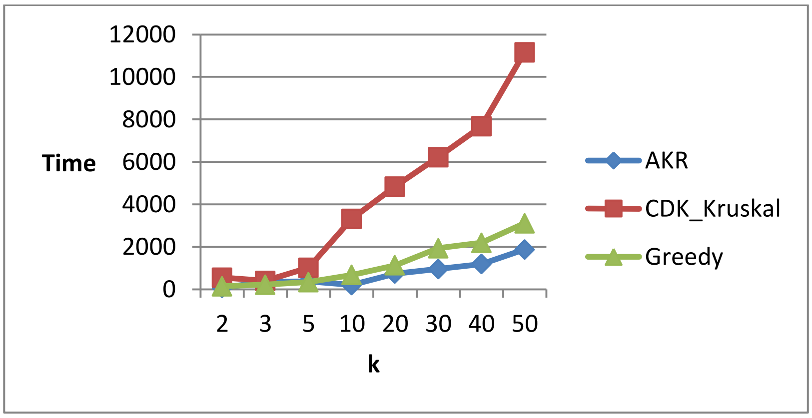

Figure 5.2.10: Running time comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Random weight (wi=1-100) for $p = 0.8$

Table 5.2.6: Computational results on Random Graphs with Random demand-Random weight (w_i=1-100) for $p = 1$

Random Weight (w_i=1-10000)						
Random Demand (d_i=1-5) and M=5						
Probability that there is an edge between a pair of nodes = 1						
k	RESULT			RUNNING TIME (ms)		
	AKR	CDK_Kruskal	Greedy	AKR	CDK_Kruskal	Greedy
2	45	45	35	94	655	172
3	24	24	24	187	390	218
5	57	57	47	390	936	374
10	148	151	114	483	2512	718
20	271	271	236	858	5195	1342
30	377	372	301	1310	7239	2246
40	530	553	521	1076	12527	2605
50	620	640	635	1825	13119	3495

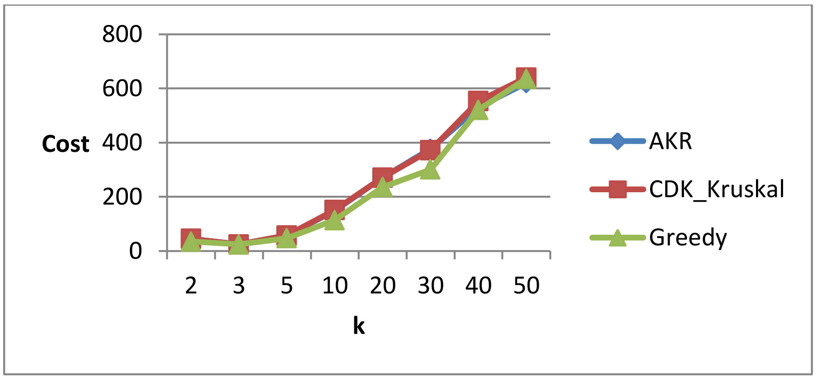

Figure 5.2.11: Cost comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Random weight (w_i=1-100) for $p = 1$

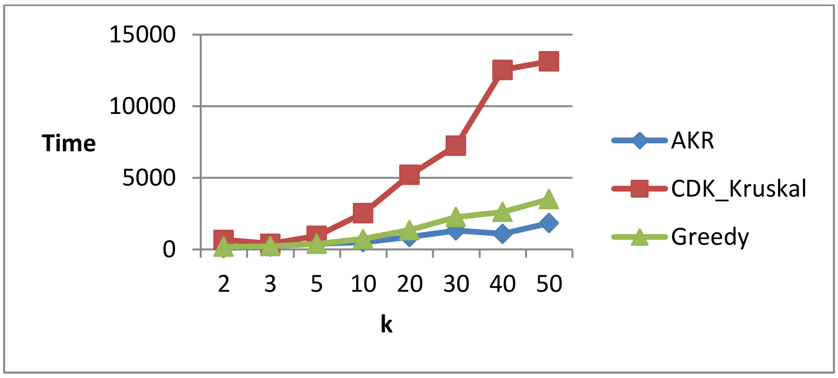

Figure 5.2.12: Running time comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Random weight (w_i=1-100) for $p = 1$

The experiments over the generated random graphs show that Greedy gives better cost results while weights of all edges and demands of terminal pairs are randomly assigned. In addition, the results of ÇDK-Kruskal and AKR are close to each other. In many cases, AKR gives slightly better results and the differences between the results become more significant as the probability of having an edge between two nodes increases. More specifically, AKR has better running time according to both ÇDK-Kruskal and Greedy since small weights causes all the moats to collide in a short period of time. Besides, the running time of Greedy is better than ÇDK-Kruskal because ÇDK-Kruskal is computing an adjunct graph which takes much time when it is compared to finding shortest path. Thus, the results of the experiments in this section are quite similar to the results stated in section 5.1 when AKR and ÇDK-Kruskal are compared since the weights of the edges are still small. The main difference is that, in this section Greedy is not better than the other algorithms give similar cost results.

Similar to the previous section, since edge weights span a relatively narrow range and the moats grown by AKR collide during the first few iterations and the result is immediately returned, AKR runs much faster than ÇDK-Kruskal and Greedy.

5.3 EXPERIMENTS ON RANDOM GRAPHS WITH RANDOM DEMAND-RANDOM WEIGHT (Wi=1-10000)

In this experiment, a random graph with 1000 nodes and random edge costs between 1 and 10000 for each vertex are generated for each p and k where $p = \{0.1, 02, 0.4, 0.6, 0.8, 1\}$ is the probability that there is an edge between a pair of nodes and $k = \{2, 3, 5, 10, 20, 30, 40, 50\}$ is the number of terminal pairs in the definition of MRoB. In addition, demands for each terminal are randomly generated between 1 and 5 and $M = 5$. Therefore, in the sampling step, every terminal pair is picked with a probability of $d_i/5$. A total of $|k| * |p| = 6 * 8 = 48$ experiments are performed in this section. The computational results for each p are given as follows:

Table 5.3.1: Computational results on Random Graphs with Random demand-Random weight (wi=1-10000) for $p = 0.1$

Random Weight (wi=1-10000)						
Random Demand (di=1-5) and M=5						
Probability that there is an edge between a pair of nodes = 0.1						
k	RESULT			RUNNING TIME (ms)		
	AKR	CDK_Kruskal	Greedy	AKR	CDK_Kruskal	Greedy
2	4903	4903	4903	811	78	172
3	7034	7034	7034	655	78	63
5	18232	18732	15972	1373	218	78
10	33984	36769	29441	4415	219	140
20	48133	51543	45646	3728	437	312
30	67371	75721	69979	17363	780	281
40	78277	83517	84601	6115	1623	452
50	107833	117103	106094	7941	1263	780

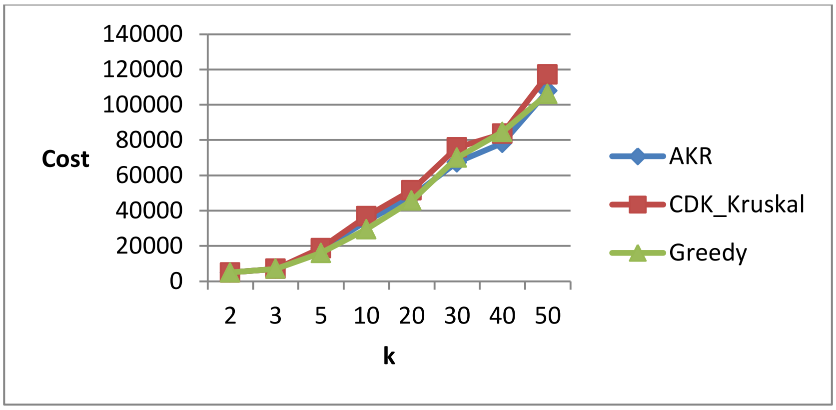

Figure 5.3.1: Cost comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Random weight (wi=1-10000) for $p = 0.1$

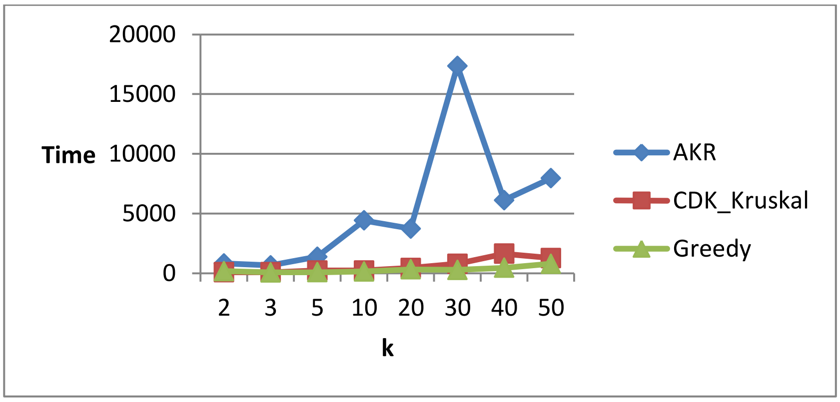

Figure 5.3.2: Running time comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Random weight (wi=1-10000) for $p = 0.1$

Table 5.3.2: Computational results on Random Graphs with Random demand-Random weight (wi=1-10000) for $p = 0.2$

Random Weight (wi=1-10000)						
Random Demand (di=1-5) and M=5						
Probability that there is an edge between a pair of nodes = 0.2						
k	RESULT			RUNNING TIME (ms)		
	AKR	CDK_Kruskal	Greedy	AKR	CDK_Kruskal	Greedy
2	1053	1053	1053	125	62	78
3	3616	3616	3616	125	78	63
5	5366	5366	5366	655	234	140
10	11467	11801	10919	1139	499	203
20	19025	19960	18796	3525	1326	531
30	37291	42206	36994	4914	1763	499
40	45452	48052	45073	6988	1966	546
50	43277	46437	44808	8736	2465	905

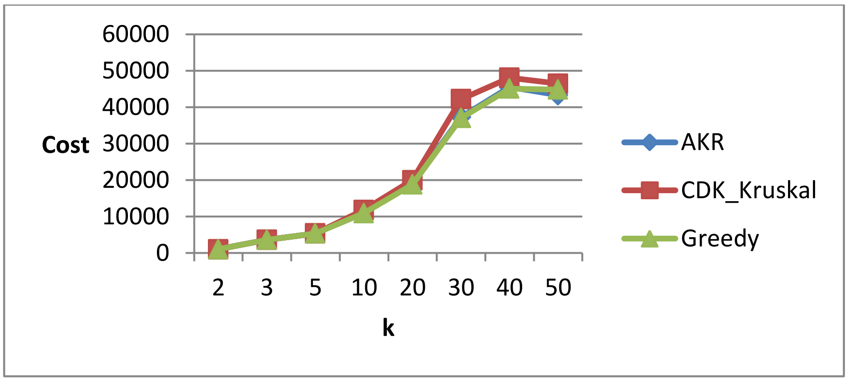

Figure 5.3.3: Cost comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Random weight (wi=1-10000) for $p = 0.2$

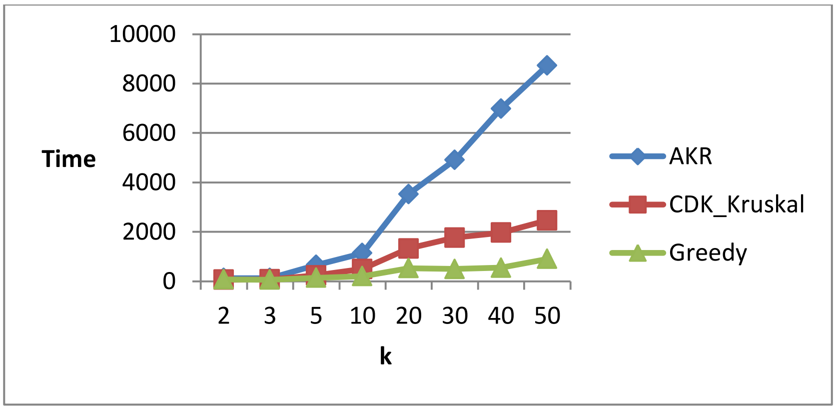

Figure 5.3.4: Running time comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Random weight (wi=1-10000) for $p = 0.2$

Table 5.3.3: Computational results on Random Graphs with Random demand-Random weight (wi=1-10000) for $p = 0.4$

Random Weight (wi=1-10000)						
Random Demand (di=1-5) and M=5						
Probability that there is an edge between a pair of nodes = 0.4						
k	RESULT			RUNNING TIME (ms)		
	AKR	CDK_Kruskal	Greedy	AKR	CDK_Kruskal	Greedy
2	2330	2645	2090	219	343	93
3	1542	1542	1542	967	203	125
5	5595	6095	4685	2153	655	172
10	8457	9092	8040	2714	1388	375
20	13010	14660	12099	9345	1809	624
30	19393	21153	19527	7613	3697	873
40	20781	23276	21119	8019	5569	1295
50	27965	30320	29611	9266	4852	1342

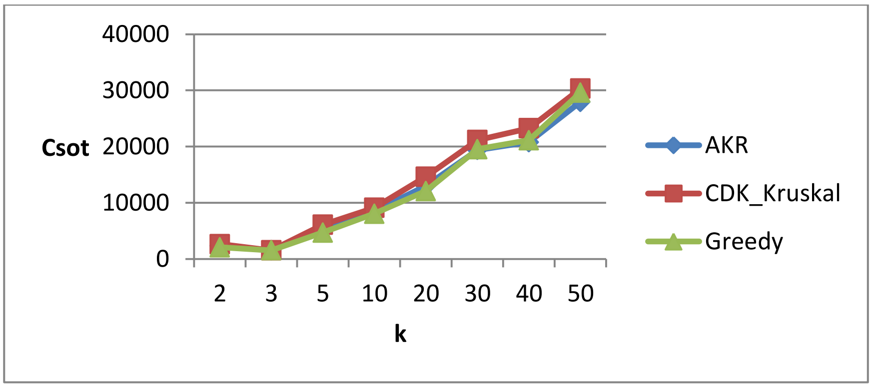

Figure 5.3.5: Cost comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Random weight (wi=1-10000) for $p = 0.4$

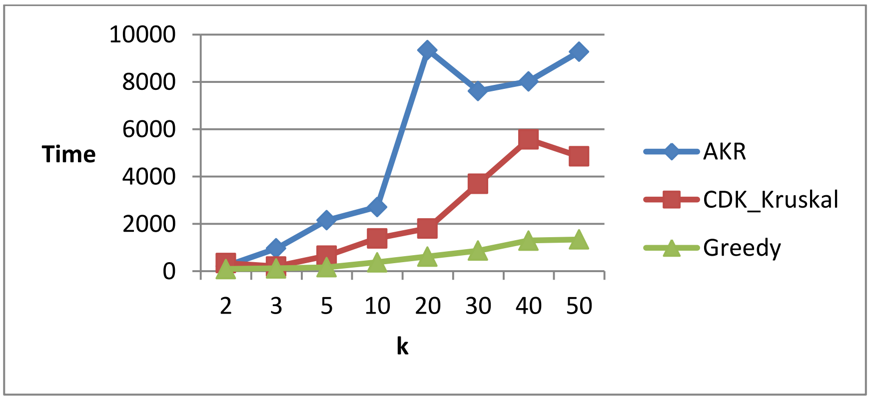

Figure 5.3.6: Running time comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Random weight (wi=1-10000) for $p = 0.4$

Table 5.3.4: Computational results on Random Graphs with Random demand-Random weight (wi=1-10000) for $p = 0.6$

Random Weight (wi=1-10000)						
Random Demand (di=1-5) and M=5						
Probability that there is an edge between a pair of nodes = 0.6						
k	RESULT			RUNNING TIME (ms)		
	AKR	CDK_Kruskal	Greedy	AKR	CDK_Kruskal	Greedy
2	760	760	760	312	249	125
3	734	734	734	312	296	172
5	1362	1362	1362	749	375	265
10	4671	4791	4170	1436	936	577
20	9794	10584	9126	4368	3338	905
30	12927	13537	12435	7410	4929	1451
40	11809	12694	12575	11170	5226	1888
50	17193	18428	17213	9391	8330	2403

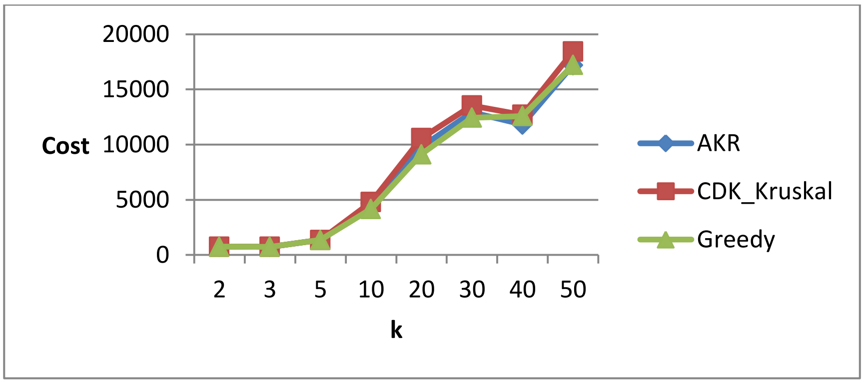

Figure 5.3.7: Cost comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Random weight (wi=1-10000) for $p = 0.6$

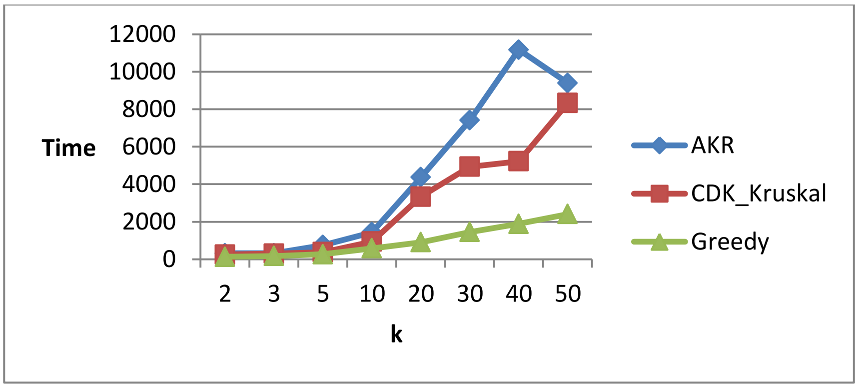

Figure 5.3.8: Running time comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Random weight for (wi=1-10000) $p = 0.6$

Table 5.3.5: Computational results on Random Graphs with Random demand-Random weight (wi=1-10000) for $p = 0.8$

Random Weight (wi=1-10000)						
Random Demand (di=1-5) and M=5						
Probability that there is an edge between a pair of nodes = 0.8						
k	RESULT			RUNNING TIME (ms)		
	AKR	CDK_Kruskal	Greedy	AKR	CDK_Kruskal	Greedy
2	1250	1345	1160	561	562	140
3	1581	1721	1531	1357	609	187
5	2513	2802	2121	3651	1248	421
10	4853	5368	4331	4134	2823	624
20	6331	6971	6016	2558	4417	1225
30	9969	10559	9851	5570	5717	1701
40	11299	11899	12054	8596	7207	2059
50	13035	13991	13216	14492	10393	2772

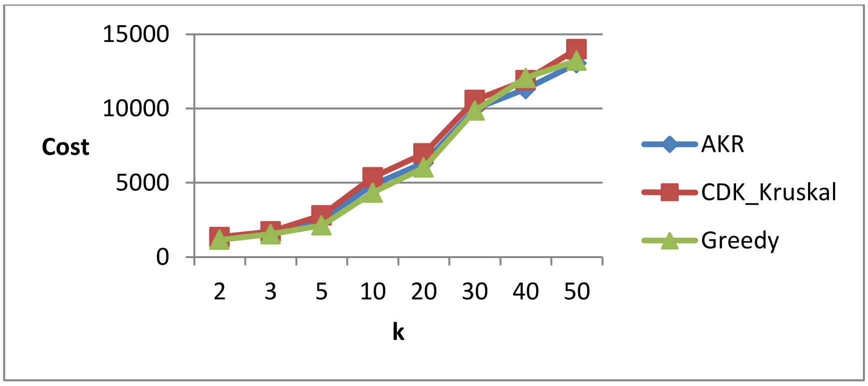

Figure 5.3.9: Cost comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Random weight (wi=1-10000) for $p = 0.8$

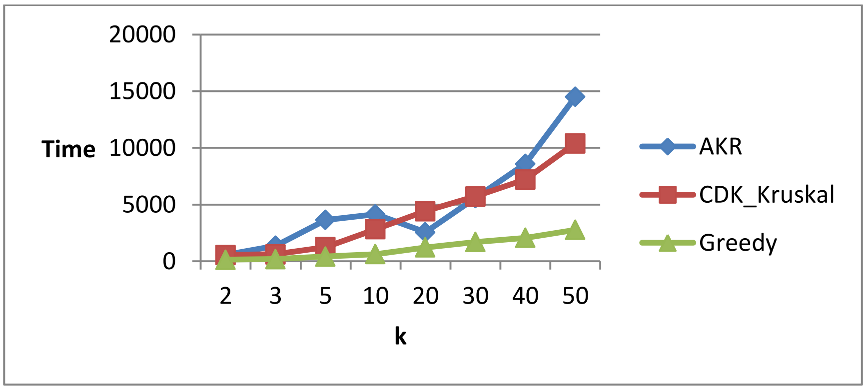

Figure 5.3.10: Running time comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Random weight (wi=1-10000) for $p = 0.8$

Table 5.3.6: Computational results on Random Graphs with Random demand-Random weight (wi=1-10000) for $p = 1$

Random Weight (wi=1-10000)						
Random Demand (di=1-5) and M=5						
Probability that there is an edge between a pair of nodes = 1						
k	RESULT			RUNNING TIME (ms)		
	AKR	CDK_Kruskal	Greedy	AKR	CDK_Kruskal	Greedy
2	815	835	685	593	624	156
3	462	462	462	203	406	234
5	2100	2405	1860	1654	1560	312
10	2627	2767	2422	2153	2418	686
20	5990	6440	5463	4103	5132	1451
30	5626	9380	8536	15615	6911	1903
40	8812	9397	5965	11123	8923	2824
50	10417	11212	11405	8984	12212	3057

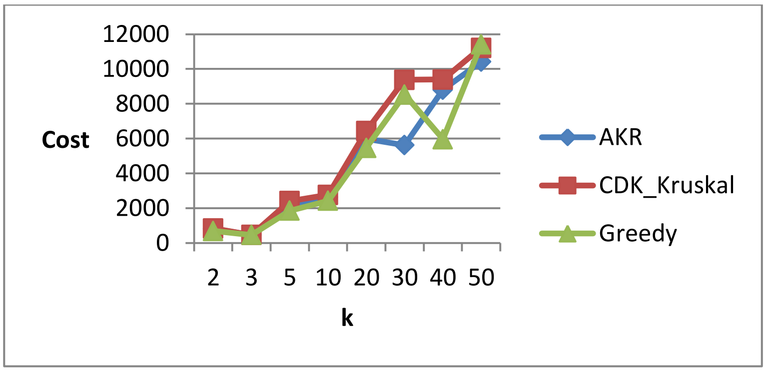

Figure 5.3.11: Cost comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Random weight (wi=1-10000) for $p = 1$

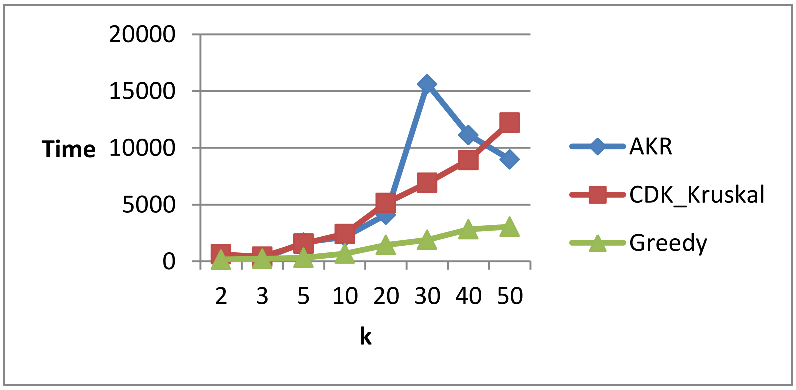

Figure 5.3.12: Running time comparison of AKR, Greedy and ÇDK-Kruskal on Random Graph with Random demand-Random weight (wi=1-10000) for $p = 1$

The experiments over the generated random graphs show that AKR gives slightly better cost results while weights of all edges and demands of terminal pairs are randomly assigned. The differences between the results become more significant as the probability of having an edge between two nodes increases. Thus, Greedy has better running time compared to ÇDK-Kruskal since ÇDK-Kruskal is computing an adjunct graph which takes much time when it is compared to finding shortest path.

One of the main results of this experiment is that when edge weights span a wide range, AKR runs slower than ÇDK-Kruskal and Greedy, especially when the graph is sparse since there is an excessive number of edge weight updates represented by moats grown in the algorithm. However, when the graph becomes denser, e.g. complete graph with $p = 1$, the number of edge weight updates decreases for AKR since there is a direct edge between any terminal pair. Hence, its running time gets closer to ÇDK-Kruskal.

5.4 EXPERIMENTS ON TSP URUGUAY GRAPH WITH RANDOM DEMANDS-

In this experiment, AKR, Greedy and ÇDK-Kruskal are run on a real world graph with 734 nodes and the edges which are determined by a value $m * y$ where $m = 5747$ is the maximum distance between two cities in Uruguay and $y = \{0.1, 02, 0.4, 0.6, 0.8, 1\}$. If the distance between two cities is smaller than or equal to $5746 * y$, the edge corresponding to this distance is included into the graph.

We run AKR, Greedy and ÇDK-Kruskal on TSP Uruguay Graph for each $k = \{2, 3, 5, 10, 20, 30, 40, 50\}$ in which the demands d_i for each terminal pair are randomly assigned integer between 1 and 5 and $M = 5$. Thus, in the sampling step, every terminal pair is picking with a probability of $d_i/5$. The computational results for each y are given as follows:

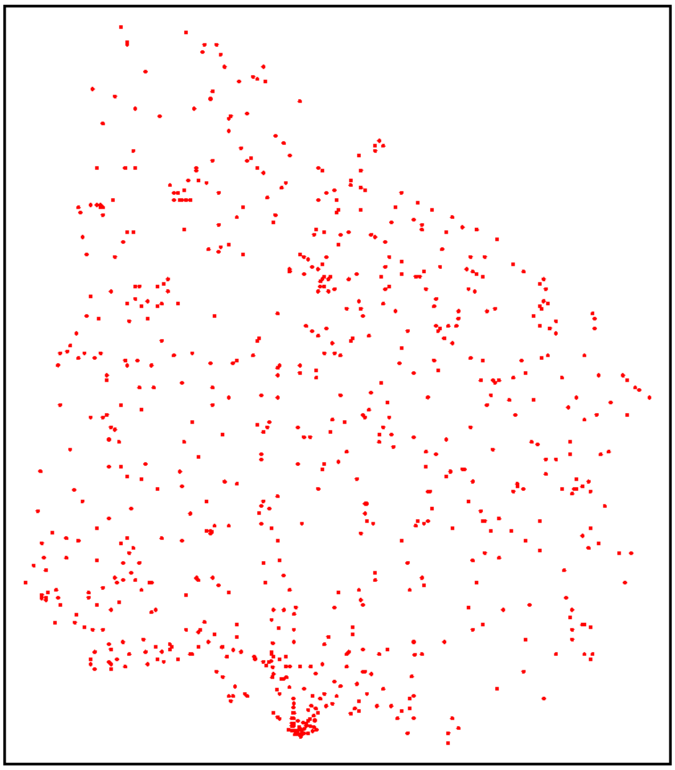

Figure 5.4.1: The point set derived from the National Imagery and Mapping Agency Database of Geographic Feature Names [34]

Figure 5.4.2: Map of Uruguay from CIA World Factbook [35]

Table 5.4.1: Computational results on TSP Uruguay Graph with random demands for $y = 0.1$

Random Demand (di=1-5) and M=5						
For the edges of weight less than 0,1 x Maximum weight (5746)						
k	RESULT			RUNNING TIME (ms)		
	AKR	CDK_Kruskal	Greedy	AKR	CDK_Kruskal	Greedy
2	20905	20905	24690	2715	62	156
3	20919	20919	20501	3806	47	31
5	38623	38623	39153	3260	63	31
10	56660	56660	64386	3182	94	47
20	78341	78595	89087	3588	249	63
30	94300	94450	108301	3900	343	94
40	106652	106439	121884	2792	484	125
50	127397	127771	137445	3713	609	171

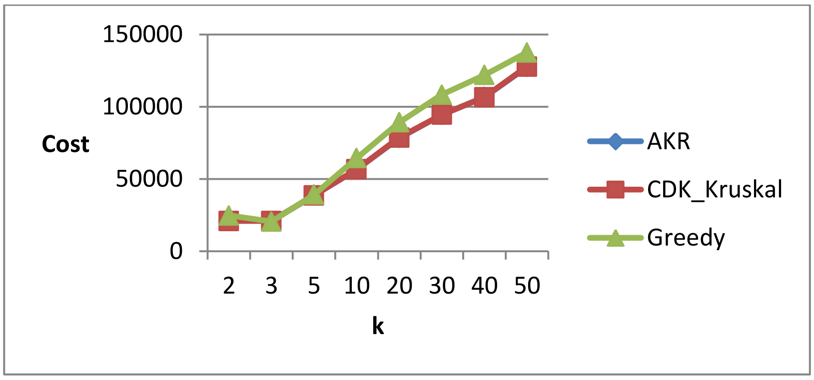

Figure 5.4.3: Cost comparison of AKR, Greedy and ÇDK-Kruskal on TSP Uruguay Graph with random demands for $y = 0.1$

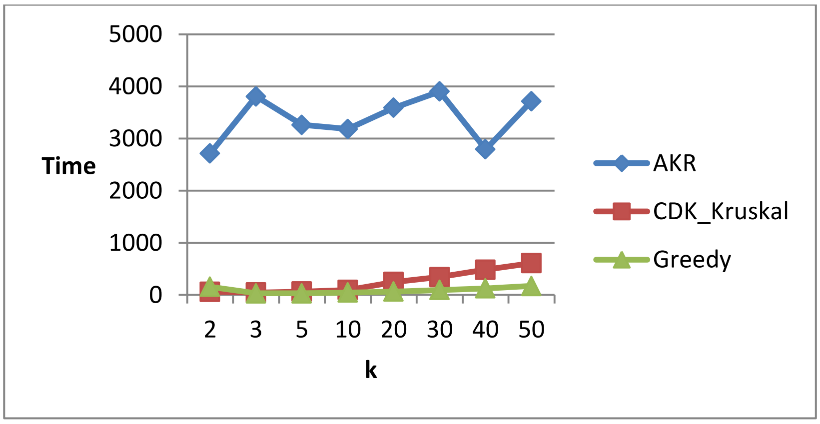

Figure 5.4.4: Running time comparison of AKR, Greedy and ÇDK-Kruskal on TSP Uruguay Graph with random demands - for $y = 0.1$

Table 5.4.2: Computational results on TSP Uruguay Graph with random demands for $y = 0.2$

Random Demand (di=1-5) and M=5						
For the edges of weight less than 0,2 x Maximum weight (5746)						
k	RESULT			RUNNING TIME (ms)		
	AKR	CDK_Kruskal	Greedy	AKR	CDK_Kruskal	Greedy
2	24280	24280	24280	18486	63	31
3	14239	14239	14239	1919	78	47
5	38807	38810	38725	26286	172	78
10	58365	59257	66911	19468	531	140
20	83408	83408	96991	21840	1170	265
30	103848	104182	118563	16442	1295	374
40	120615	121027	152622	20389	1170	328
50	115420	115939	146363	16099	2434	639

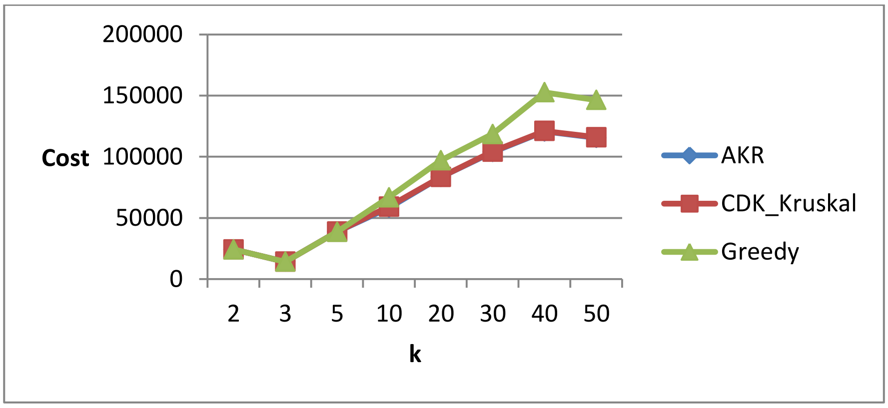

Figure 5.4.5: Cost Comparison of AKR, Greedy and ÇDK-Kruskal on TSP Uruguay Graph with random demands for y=0.2

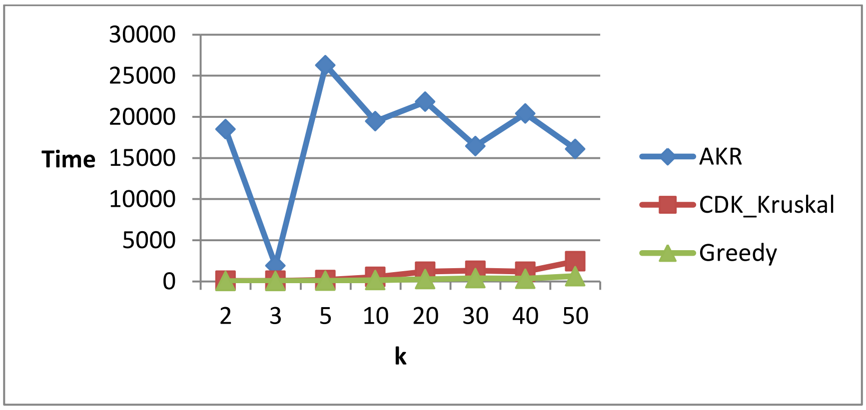

Figure 5.4.6: Running time comparison of AKR, Greedy and ÇDK-Kruskal on TSP Uruguay Graph with random demands - for $y = 0.2$

Table 5.4.3: Computational results on TSP Uruguay Graph with random demands for $y = 0.4$

Random Demand (di=1-5) and M=5						
For the edges of weight less than 0,4 x Maximum weight (5746)						
k	RESULT			RUNNING TIME (ms)		
	AKR	CDK_Kruskal	Greedy	AKR	CDK_Kruskal	Greedy
2	13250	13250	13250	19141	203	93
3	13445	13495	13495	48376	359	140
5	54045	54409	55445	104988	733	203
10	49633	49633	57008	96689	1294	328
20	94813	95019	105224	71417	2387	671
30	97987	97821	122524	89715	2746	1295
40	109474	109912	142335	83943	4228	1529
50	127912	128206	166226	50809	3073	1638

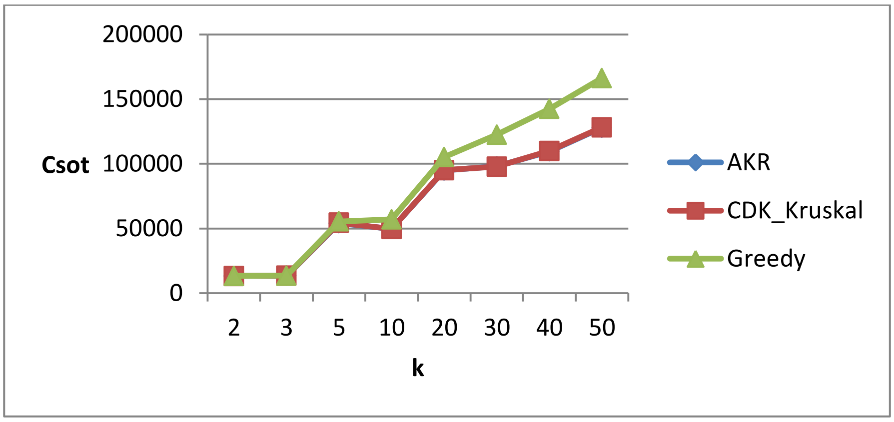

Figure 5.4.7: Cost comparison of AKR, Greedy and ÇDK-Kruskal on TSP Uruguay Graph with random demands for $y = 0.4$

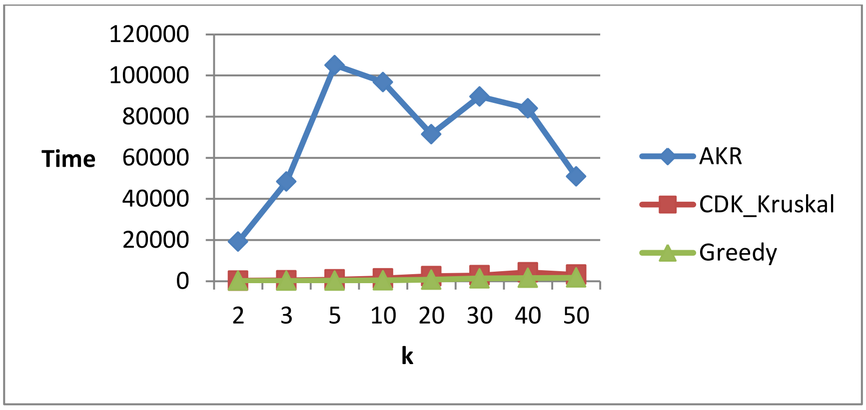

Figure 5.4.8: Running time comparison of AKR, Greedy and ÇDK-Kruskal on TSP Uruguay Graph with random demands - for $y = 0.4$

Table 5.4.4: Computational results on TSP Uruguay Graph with random demands for $y = 0.6$

Random Demand (di=1-5) and M=5						
For the edges of weight less than 0,6 x Maximum weight (5746)						
k	RESULT			RUNNING TIME (ms)		
	AKR	CDK_Kruskal	Greedy	AKR	CDK_Kruskal	Greedy
2	10350	10350	10350	10935	297	125
3	34950	34950	37390	144815	998	203
5	48336	48336	56050	186170	1466	328
10	52578	52578	62188	96127	3136	624
20	74261	75003	78298	158184	2043	1295
30	110278	110494	148778	86455	3416	1966
40	103725	103725	136375	132678	9641	2371
50	127048	127060	160933	73850	5023	3261

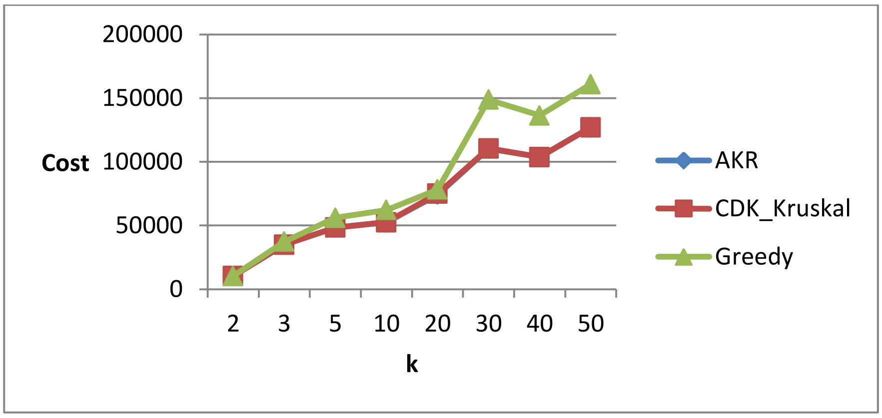

Figure 5.4.9: Cost comparison of AKR, Greedy and ÇDK-Kruskal on TSP Uruguay Graph with random demands for $y = 0.6$

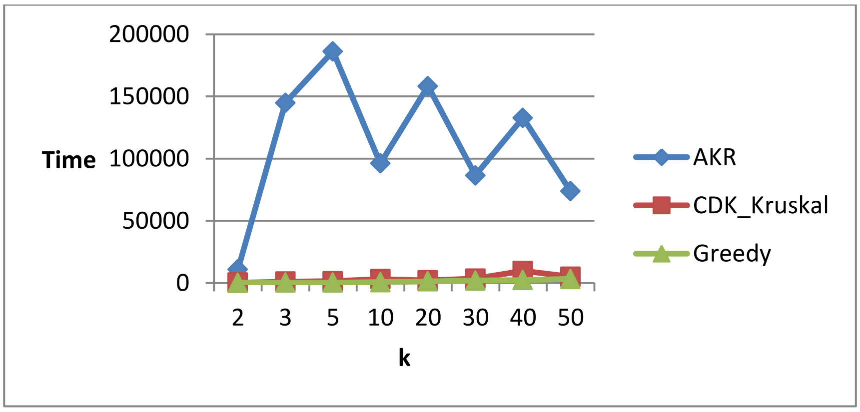

Figure 5.4.10: Running time comparison of AKR, Greedy and ÇDK-Kruskal on TSP Uruguay Graph with random demands - for $y = 0.6$

Table 5.4.5: Computational results on TSP Uruguay Graph with random demands for $y = 0.8$

Random Demand (di=1-5) and M=5						
For the edges of weight less than 0,8 x Maximum weight (5746)						
k	RESULT			RUNNING TIME (ms)		
	AKR	CDK_Kruskal	Greedy	AKR	CDK_Kruskal	Greedy
2	3436	3436	3436	125	93	110
3	21412	21412	21312	32448	811	234
5	30803	30803	32738	112071	1045	328
10	63890	63890	74391	114348	3073	718
20	86275	56847	113775	102835	2496	1451
30	104813	105116	126085	103584	5928	1513
40	111628	112162	134230	117468	11841	2917
50	126788	127185	182165	89169	12418	3744

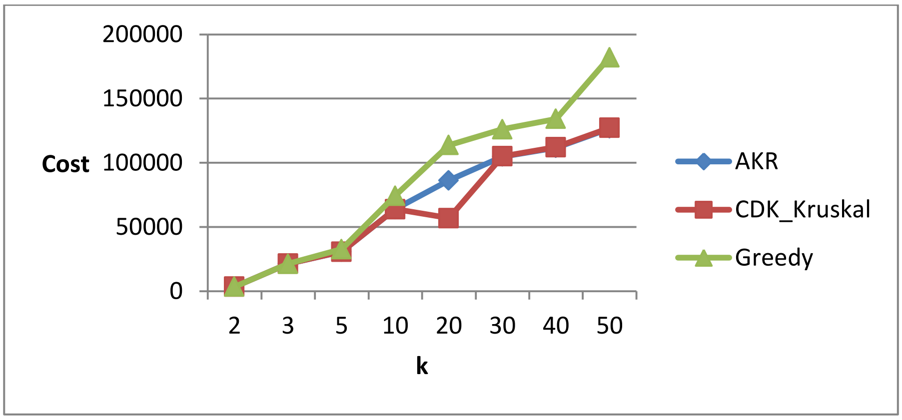

Figure 5.4.11: Cost comparison of AKR, Greedy and ÇDK-Kruskal on TSP Uruguay Graph with random demands for $y = 0.8$

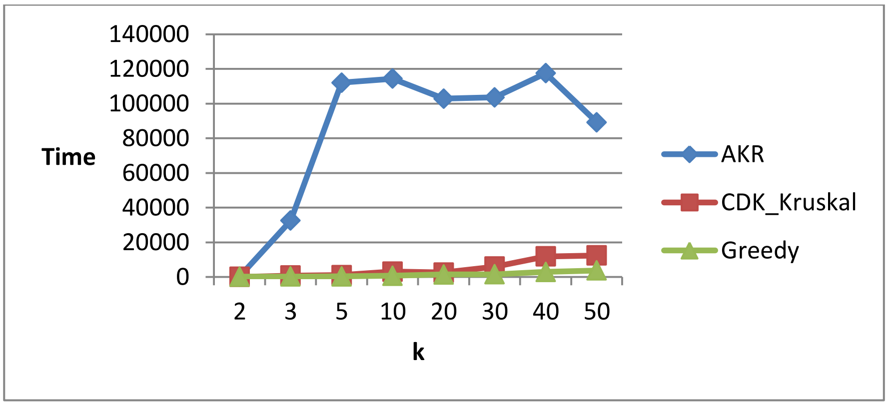

Figure 5.4.12: Running time comparison of AKR, Greedy and ÇDK-Kruskal on TSP Uruguay Graph with random demands - for $y = 0.8$

Table 5.4.6: Computational results on TSP Uruguay Graph with random demands for $y = 1$

Random Demand (di=1-5) and M=5						
For the edges of weight less than 1 x Maximum weight (5746)						
k	RESULT			RUNNING TIME (ms)		
	AKR	CDK_Kruskal	Greedy	AKR	CDK_Kruskal	Greedy
2	8735	8735	8735	35069	359	156
3	20151	20151	20151	119018	374	328
5	37937	37985	38739	321976	687	280
10	53773	54313	58254	63208	1544	452
20	77844	78049	88494	144730	4071	1342
30	87204	88033	102970	179272	4664	1810
40	104174	104236	126566	81773	11451	2745
50	104151	104226	140009	94863	11841	3728

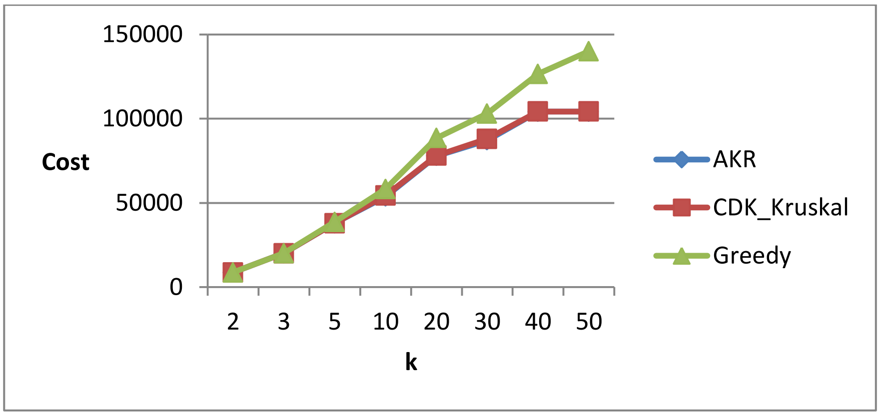

Figure 5.4.13: Cost comparison of AKR, Greedy and ÇDK-Kruskal on TSP Uruguay Graph with random demands for $y = 1$

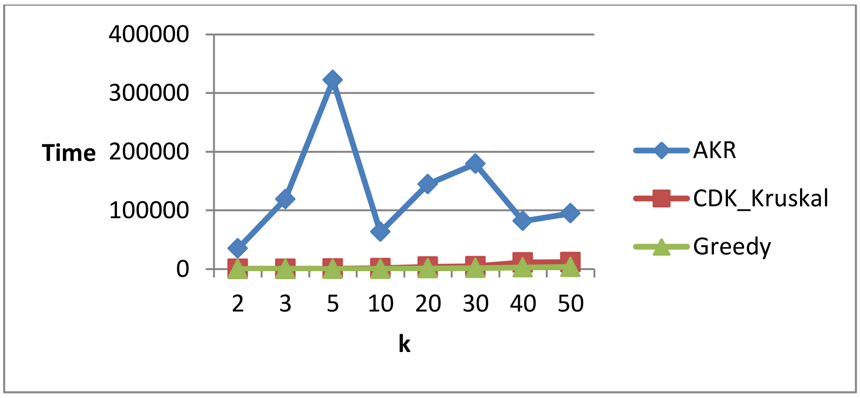

Figure 5.4.14: Running time comparison of AKR, Greedy and ÇDK-Kruskal on TSP Uruguay Graph with random demands- for $y = 1$

The experiments over the TSP Uruguay Graph show that AKR and ÇDK-Kruskal gives better cost results than Greedy. And in many cases AKR gives slightly better results than ÇDK-Kruskal. The differences between the results become more significant as the edges between cities which are determined by the value $m * y$ increases. Thus, Greedy has better running time compared to ÇDK-Kruskal since ÇDK-Kruskal is computing an adjunct graph which takes much time when it is compared to finding shortest path. We would like to note that, we have run these algorithms on various TSP data and eventually, very similar to the ones we have observed for TSP Uruguay.

One of the main results of this experiment is that since edge weights span a wide range in real world data and it causes excessive number of edge weight updates represented by moat grown in the AKR algorithm, it runs much slower than ÇDK-Kruskal and Greedy. However, when the graph becomes denser, e.g. complete graph with $p = 1$, the number of edge weight updates decreases for AKR since there is a direct edge between any terminal pair. Hence, its running time gets closer to ÇDK-Kruskal. In addition, ÇDK-Kruskal turns out to be a very good algorithm for real world graphs since its running time is closer to Greedy, but its cost results are similar to AKR.

5.5 EXPERIMENTS ON A GEOMETRIC RANDOM GRAPH WITH RANDOM DEMAND

In this experiment, AKR, Greedy and ÇDK-Kruskal are run on a geometric random graph with 1000 nodes and the edges which are determined by a value $m * y$ where $m = 1368$ is the maximum distance between two nodes in graph and $y = \{0.1, 02, 0.4, 0.6, 0.8, 1\}$. If the distance between two nodes is smaller than or equal to $1368 * y$, the edge corresponding to this distance is included into the graph.

We run AKR, Greedy and ÇDK-Kruskal on generated Geometric Random Graph for each $k = \{2, 3, 5, 10, 20, 30, 40, 50\}$ in which the demands d_i for each terminal pair are randomly assigned integer between 1 and 5 and $M = 5$. Thus, in the sampling step, every terminal pair is picking with a probability of $d_i/5$. The computational results for each y are given as follows:

Table 5.5.1: Computational results on Geometric Random Graph with random demands for $y = 0.1$

Random Demand (di=1-5) and M=5						
For the edges of weight less than 0,1 x Maximum weight (1368)						
k	RESULT			RUNNING TIME (ms)		
	AKR	CDK_Kruskal	Greedy	AKR	CDK_Kruskal	Greedy
2	4385	4385	4500	1373	88	587
3	3582	3582	3582	117	17	72
5	10547	10547	10314	1691	459	63
10	11837	11837	11030	1611	265	93
20	20003	20051	22867	2000	605	194
30	21552	21559	25812	1911	531	199
40	26734	26769	31192	1534	661	221
50	29838	29878	35552	1359	892	335

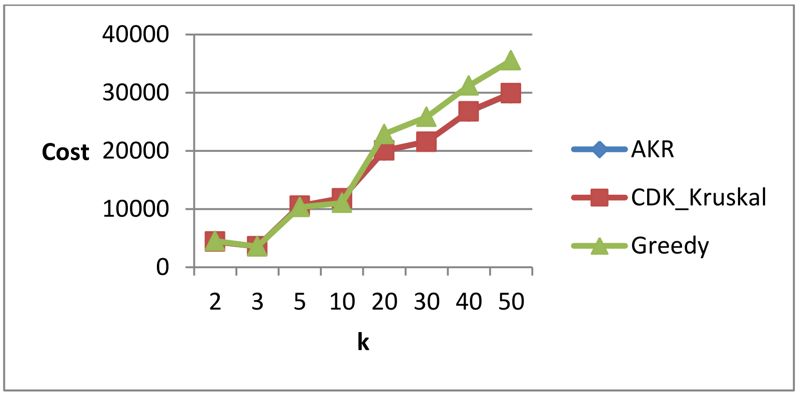

Figure 5.5.1: Cost comparison of AKR, Greedy and ÇDK-Kruskal on Geometric Random Graph with random demands for $y = 0.1$

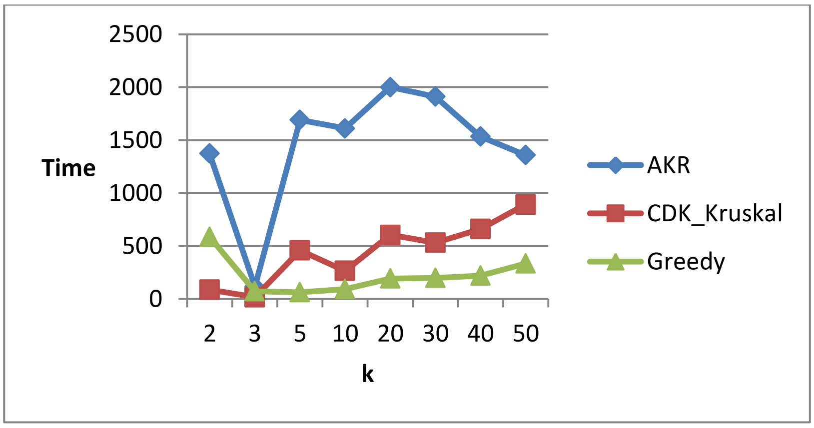

Figure 5.5.2: Running time comparison of AKR, Greedy and ÇDK-Kruskal on Geometric Random Graph with random demands - for $y = 0.1$

Table 5.5.2: Computational results on Geometric Random Graph with random demands for $y = 0.2$

Random Demand (di=1-5) and M=5						
For the edges of weight less than 0,2 x Maximum weight (1368)						
k	RESULT			RUNNING TIME (ms)		
	AKR	CDK_Kruskal	Greedy	AKR	CDK_Kruskal	Greedy
2	7065	7065	7045	15838	259	74
3	3733	3733	3733	5444	114	84
5	5708	5718	5718	4270	288	118
10	14060	14180	14898	8391	680	236
20	21359	21433	24864	13585	1337	356
30	23323	23315	28121	17454	2233	666
40	27330	27573	32515	8522	2396	875
50	31030	31193	37241	9146	3121	1340

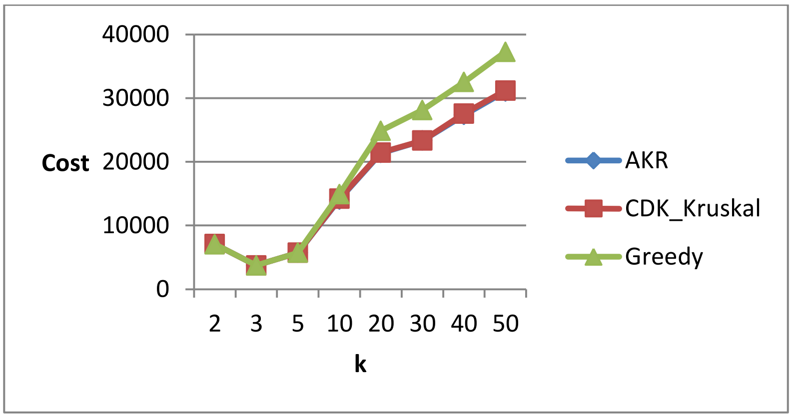

Figure 5.5.3: Cost Comparison of AKR, Greedy and ÇDK-Kruskal on Geometric Random Graph with random demands for y=0.2

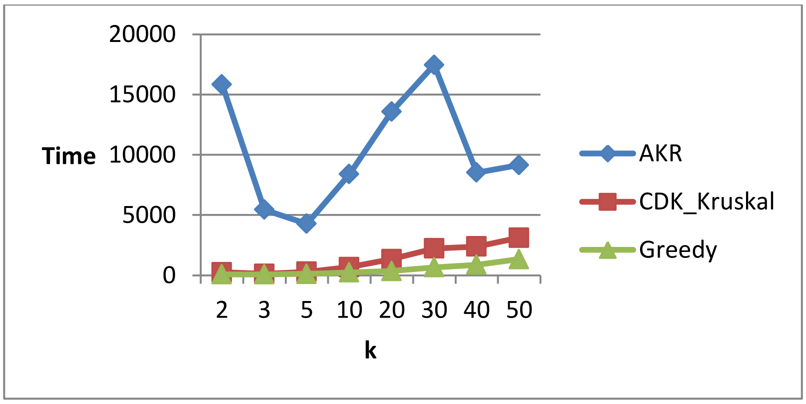

Figure 5.5.4: Running time comparison of AKR, Greedy and ÇDK-Kruskal on Geometric Random Graph with random demands - for $y = 0.2$

Table 5.5.3: Computational results on Geometric Random Graph with random demands for $y = 0.4$

Random Demand (di=1-5) and M=5						
For the edges of weight less than 0,4 x Maximum weight (1368)						
k	RESULT			RUNNING TIME (ms)		
	AKR	CDK_Kruskal	Greedy	AKR	CDK_Kruskal	Greedy
2	2184	2184	2184	117	72	298
3	5448	5563	5563	50886	326	415
5	7808	7836	8180	19922	1322	421
10	11810	11883	14058	42376	2116	498
20	17480	20333	23086	17480	4101	1549
30	22556	22577	27523	43473	6873	1711
40	26286	26557	33152	24409	7591	2474
50	29972	29987	37306	28833	9655	2938

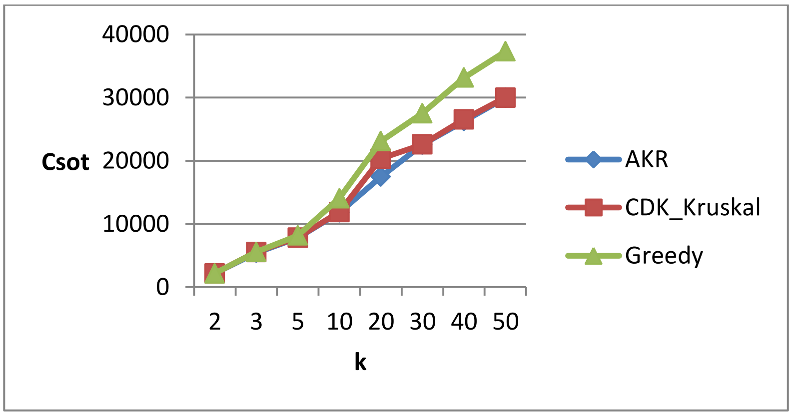

Figure 5.5.5: Cost comparison of AKR, Greedy and ÇDK-Kruskal on Geometric Random Graph with random demands for $y = 0.4$

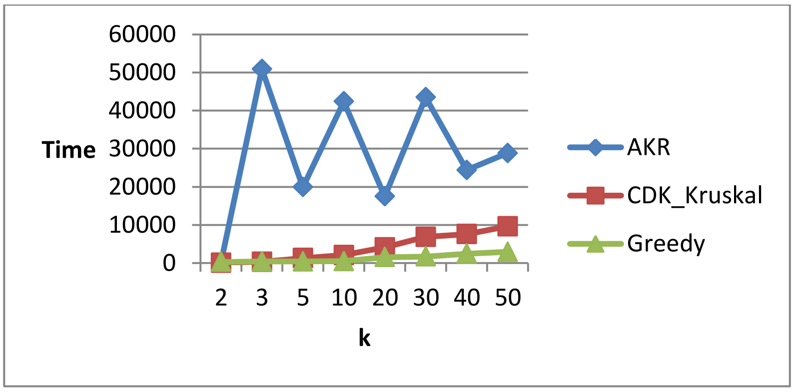

Figure 5.5.6: Running time comparison of AKR, Greedy and ÇDK-Kruskal on Geometric Random Graph with random demands - for $y = 0.4$

Table 5.5.4: Computational results on Geometric Random Graph with random demands for $y = 0.6$

Random Demand (di=1-5) and M=5						
For the edges of weight less than 0,6 x Maximum weight (1368)						
k	RESULT			RUNNING TIME (ms)		
	AKR	CDK_Kruskal	Greedy	AKR	CDK_Kruskal	Greedy
2	5435	5417	5417	84742	608	275
3	5338	5226	5226	70072	524	289
5	6867	6879	6879	77002	1511	547
10	12854	12854	15350	78500	4354	1215
20	19099	19290	22286	28835	6616	1940
30	23543	23544	28158	32437	12299	3059
40	26451	26575	32311	57064	14880	3905
50	31716	31789	39523	41520	13421	3391

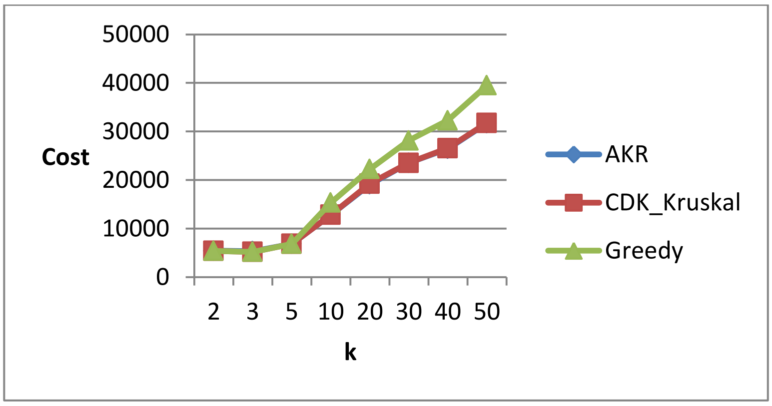

Figure 5.5.7: Cost comparison of AKR, Greedy and ÇDK-Kruskal on Geometric Random Graph with random demands for $y = 0.6$

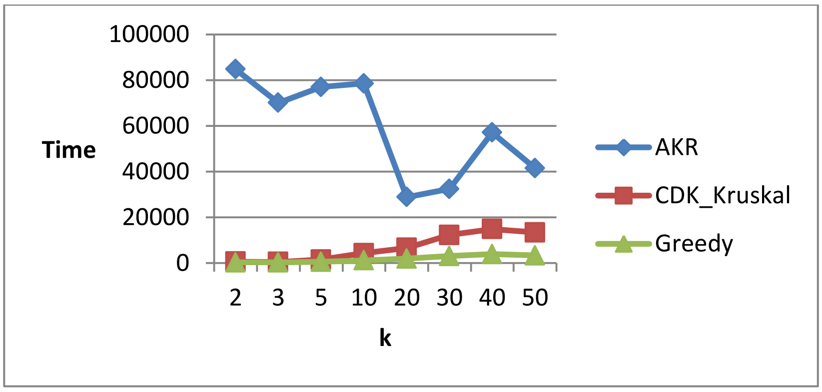

Figure 5.5.8: Running time comparison of AKR, Greedy and ÇDK-Kruskal on Geometric Random Graph with random demands - for $y = 0.6$

Table 5.5.5: Computational results on Geometric Random Graph with random demands for $y = 0.8$

Random Demand (di=1-5) and M=5						
For the edges of weight less than 0,8 x Maximum weight (1368)						
k	RESULT			RUNNING TIME (ms)		
	AKR	CDK_Kruskal	Greedy	AKR	CDK_Kruskal	Greedy
2	3625	3625	3625	46730	442	214
3	4607	4607	4607	56000	350	204
5	9295	9295	9237	68611	2201	483
10	11776	11776	14151	27701	3311	809
20	19693	19849	24131	55874	6646	1746
30	22666	22699	27806	37996	11975	2927
40	26538	26625	32508	47549	12865	3151
50	50150	29831	37518	50150	17459	5742

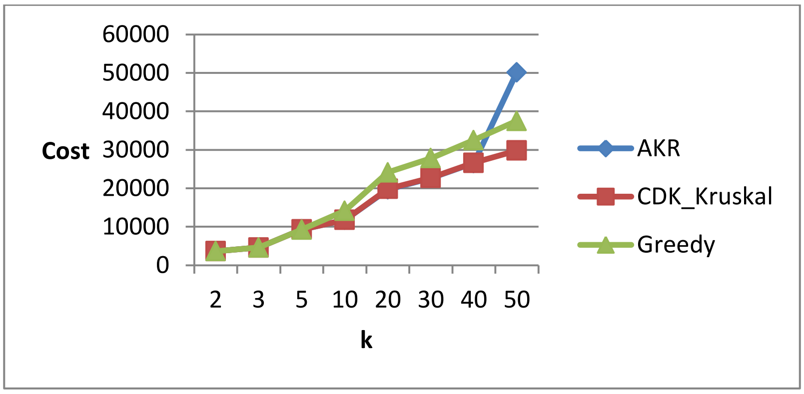

Figure 5.5.9: Cost comparison of AKR, Greedy and ÇDK-Kruskal on Geometric Random Graph with random demands for $y = 0.8$

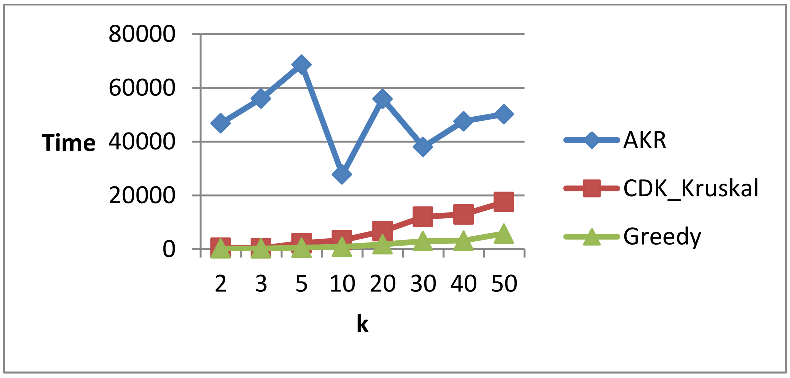

Figure 5.5.10: Running time comparison of AKR, Greedy and ÇDK-Kruskal on Geometric Random Graph with random demands - for $y = 0.8$

Table 5.5.6: Computational results on Geometric Random Graph with random demands for $y = 1$

Random Demand (di=1-5) and M=5						
For the edges of weight less than 1 x Maximum weight (1368)						
k	RESULT			RUNNING TIME (ms)		
	AKR	CDK_Kruskal	Greedy	AKR	CDK_Kruskal	Greedy
2	5323	5277	5277	70098	376	371
3	4811	4811	4811	2558	776	414
5	9359	9359	10027	150649	1453	627
10	16802	16818	19496	85523	3184	1603
20	20247	20345	23054	120418	5002	1874
30	22804	22864	27082	57020	13525	3432
40	27911	28054	33428	42646	11421	3122
50	28716	28901	35349	41288	14713	4607

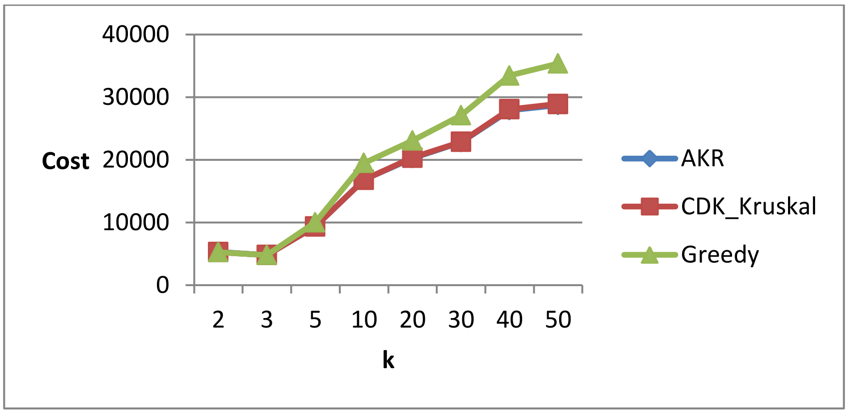

Figure 5.5.11: Cost comparison of AKR, Greedy and ÇDK-Kruskal on Geometric Random Graph with random demands for $y = 1$

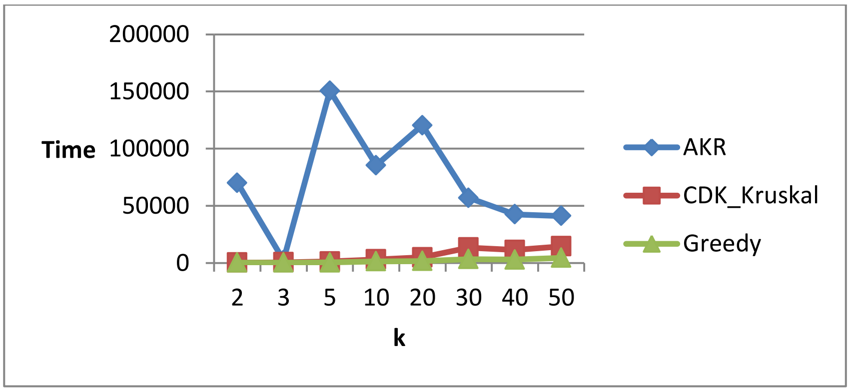

Figure 5.5.12: Running time comparison of AKR, Greedy and ÇDK-Kruskal on Geometric Random Graph with random demands- for $y = 1$

The experiments of this section are qualitatively the same as the previous section. Overall, ÇDK-Kruskal is a good alternative to AKR for random geometric graphs and real-world geometric graphs.

REFERENCES

[1] David P. Williamson and David B. Shmoys, *The Design of Approximation Algorithm*, Ithaca, New York, 2010.

[2] A. Gupta, A. Kumar, M. Pal, and T. Roughgarden, "Approximation via cost-sharing: A simple approximation algorithm for the multicommodity rent-or-buy problem", *Proceedings of IEEE Symposium on Foundations of Computer Science*, October 2003, pp. 606-615, Cambridge, England, 2003.

[3] M. Bern and P. Plassmann, "The Steiner problem with edge lengths 1 and 2", *Inform. Process. Lett.*, Vol. 32, pp. 171–176, 1981.

[4] B. Awerbuch and Y. Azar, "Buy-at-bulk network design", *In 38th FOCS*, pp: 542–547, 1997.

[5] Y. Bartal. "On approximating arbitrary metrics by tree metrics", *Inv 30th STOC*, pp: 161–168, 1998.

[6] A. Kumar, A. Gupta, and T. Roughgarden, "A constant-factor approximation algorithm for the multicommodity rent-or-buy problem", *In FOCS*, pp: 333–342, 2002.

[7] A.Gupta, A. Kumar, and T. Roughgarden, "Simpler and better approximation algorithms for network design", *in Proceedings of the 35^{th} Annual ACM Symposium on Theory of Computing*, STOC'03, pp. 365-372, 2009.

[8] A. Gupta, A. Kumar, M. Pal, and T. Roughgarden, "Approximation via cost-sharing: A simple approximation algorithm for the multicommodity rent-or-buy problem". *In Proceedings, IEEE Symposium on Foundations of Computer Science*, pp. 606-615, 2003.

[9] A. Gupta, A. Kumar, M. Pal, and T. Roughgarden, "Approximation via cost sharing: Simpler and better approximation algorithm for network design", *J. ACM*, voll.54, p.11, 2007.

[10] L. Fleischer, J. Könemann, S. Leonardi, and G. Schafer, "Strict cost sharing schemes for steiner forest", *SIAN J. Comput.*, vol. 39, pp. 3616-3632, 2010.

[11] A. Gupta and M. Pal, “Stochastic Steiner tree without a root”, *Proceedings of the 32nd International Conference on Automata, Languages and Programming*, ICALP’05, pp.1051-1063, 2005.

[12] A. Gupta, M. Pal, R. Ravi, and A. Sinha, “Boosted sampling: approximation algorithms for stochastic optimization”, STOC’04, pp. 417-426, 2004.

[13] L. Becchetti, J. Könemann, S. Leonardi, and M. Pal, “Sharing the cost more efficiently: Improved approximation for multicommodity rent-or-buy”, *Proceedings of ACM Transactions on Algorithms*, Fabruary 2007, vol. 3, pp. 1-22, 2007.

[14] L. Fleischer, J. Könemann, S. Leonardi, and G. Schaffer, *Strict cost share for Steiner Forest, SIAM J. Comput*., vol. 39 pp. 3616-3632, 2010.

[15] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, CA, 1979.

[16] M. Chlebik and J. Chlebikova, “The Steiner tree problem on graphs: Inapproximability results”, *Theor. Comput. Sci*., Vol. 406, pp. 207-214, 2008.

[17] H. Takahashi and A. Matsuyama, “An approximate solution for the Steiner problem in graphs”, *Math. Jap*., Vol. 24, pp. 573–577, 1980.

[18] A. Zelikovsky, “An 11/6-approximation algorithm for the network Steiner problem”, *Algorithmica*, Vol. 9, pp. 463–470, 1993.

[19] P. Berman and V. Ramaiyer, “Improved approximations for the Steiner tree problem”, *J. Algorithms*, Vol. 17, pp. 381–408, 1994.

[20] A. Zelikovsky, “Better Approximation Bounds for the Network and Euclidean Steiner Tree Problems”, *Tech. report CS-96-06*, University of Virginia, Charlottesville, VA, 1996.

[21] H. J. Prömel and A. Steger, “RNC-approximation algorithms for the Steiner problem”, *in Proceedings of the 14th Annual Symposium on Theoretical Aspects of Computer Science*, Springer, Berlin, pp. 559–570, 1997.

[22] M. Karpinski and A. Zelikovsky, “New approximation algorithms for the Steiner tree problem”, *J. Combin. Optim*., Vol. 1, pp. 47–65, 1997

[23] S. Hougardy, and H. J. Prömel, “A 1.598 approximation algorithm for the Steiner problem in graphs”, *in Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SIAM, Philadelphia, ACM, New York, pp. 448–453, 1999.

[24] G. Robins and A. Zelikovsky, “Tighter bounds for graph Steiner tree approximation”, *SIAM J. Discrete Math*., Vol. 19, pp. 122-134, 2005.

[25] J. Byrka, F. Grandoni, T. Rothvoss, and L. Sanita, “Steiner tree approximation via iterative randomized rounding”, *J. ACM*, Vol. 60, p. 6, 2013.

[26] A. Agrawal, P. N. Klein, and R. Ravi, “When trees collide: An approximation algorithm for the generalized steiner problem on networks”, *SIAM J. Comput*., Vol. 24, pp. 440-456, 1995.

[27] M. X. Goemans and D. P. Williamson, “A general approximation technique for constrained forest problems”, *SIAM J. Comput*., Vol. 24, pp. 296-317, 1995.

[28] J. Könemann, S. Leonardi, G. Schafer, and S. H. M. van Zwam, “A group-strategy proof cost sharing mechanism for the Steiner forest game”, *SIAM J. Comput.*, Vol. 37, pp. 1319-1341, 2008.

[29] J.B. Kruskal, “On the shortest spanning subtree of a graph and the traveling salesman problem”, *Proc. AMS*, Vol. 7, pp: 48-50, 1956.

[30] R. C. Prim, “Shortest connection networks and some generalizations”, *Bell Syst. Tech. Journal*, Vol. 36, pp: 122-134, 1957.

[31] J. Nesetril, E. Milkov, H. Nesetrilov, “Otakar boruvka on minimum spanning tree problem translation of both the 1926 papers, comments, history”., *Discrete Mathematics*, Vol. 233, pp: 1389-1401, 2001.

[32] T. H. Cormen, C.E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, (3. ed.), MIT Press, Cambridge, 2009.

[33] http://www.math.uwaterloo.ca/tsp/world/countries.html (Accessed on 30.06.2014)

[34] http://www.math.uwaterloo.ca/tsp/world/uypoints.html (Accessed on 30.06.2014)

[35] http://www.math.uwaterloo.ca/tsp/world/uymap.html (Accessed on 30.06.2014)