A HEURISTIC APPROACH TO THE CHANCE CONSTRAINED MINIMUM SPANNING *k*-CORE PROBLEM

By

AMEYA ABASAHEB DHAYGUDE

Bachelor of Engineering

University of Mumbai

Mumbai, India

2006

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 2010

A HEURISTIC APPROACH TO

THE CHANCE CONSTRAINED

MINIMUM SPANNING *k*-CORE PROBLEM

Thesis Approved:

Dr. Balabhaskar Balasundaram

Thesis Adviser

Dr. Tieming Liu

Dr. Manjunath Kamath

Dr. Mark E. Payton

Dean of the Graduate College

.

ACKNOWLEDGEMENTS

I am deeply indebted to my advisor Dr. Balabhaskar (Baski) Balasundaram for his endless patience, support and encouragement throughout this thesis. It has been an honor to be one of his first M.S. thesis students. Dr. Baski a perfectionist individual and obsessive researcher, is a precious guide in my professional and personal development. I appreciate all his contributions of time, ideas, advices and funding to make my M.S. experience productive and invigorating.

I would like to express my sincere thanks to my committee members Dr. Tieming Liu and Dr. Manjunath Kamath for serving on my graduate committee. Their valuable comments and insightful questions during the proposal defense contributed to the research directions and activities. The IEM department at Oklahoma State University provided me with wonderful learning atmosphere that helped me to develop required sets of skills in academia.

I am grateful to Foad Mahdavi for patiently helping me with C++, Xpress-MP and other conceptual doubts throughout the thesis. Also, I would like to express my special thanks to Kedar Vilankar to solve my countless MATLAB issues. My special thanks to Amey Phadke for helping me with several Linux concerns.

I would like to specially thank Amol Bhave and Sameer Managalvedhe for being wonderful colleagues, and everlasting friends. They made my life in Stillwater, a memorable one.

My time at Oklahoma State University was made enjoyable by many friends that became a part of my life. I would like to thank Kunal Divekar, Abhishek Dhuri and Alok Dange, for their lasting friendships. It would be difficult for me to name others who have helped me personally and professionally. My sincere thanks are due to all of them.

Although I have worked many late nights and weekends, I think the ones who deserve the most recognition is my family. I am deeply indebted to my father Abasaheb Dhaygude for always giving me more than I deserved and more than they could afford. I am greatly obliged to my mother Rajani Dhaygude for providing endless love, encouragement and support. I am glad for the love and support, I received from Atit and Manali. Also, thankful for the support of my cousin Rohan Kulkarni for last two of my M.S. years. Thank you.

Ameya Dhaygude
Oklahoma State University
December, 2010

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

TABLE Page

LIST OF FIGURES

CHAPTER I

INTRODUCTION

In mathematics, a *graph* is defined by a pair (V, E) where V is the vertex set and E is the edge set. Figure 1 illustrates a simple graph with 4 vertices and 6 edges. Graphs can be effective tools to represent many real-life situations, where the vertex set is representative of entities and edges indicates the presence or absence of specific relationships between pairs of these entities. We use the terms "graph" and "network" interchangeably. Users of all real-life networks wish to transfer some entity like electricity, material, final product, information, vehicle etc. from one vertex to another vertex through edges. Users prefer a cost effective and well connected network. This thesis is about a combinatorial optimization problem that fulfills these design requirements.

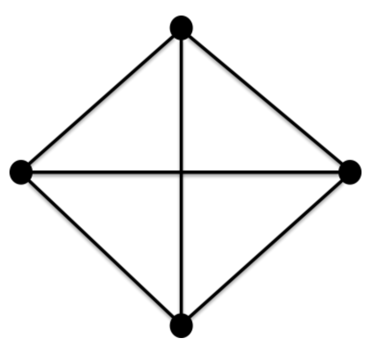


Figure 1. A simple graph

In this thesis, we always consider finite, simple and undirected graphs. An undirected graph denoted by $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$ is a vertex set with $|V| = n$ and E is an edge set with $|E| = m$ that consists of unordered pairs of vertices called as edges [i.e. Edge $(i, j) =$ Edge (j, i)]. An edge $(i, j) \in E$ is said to be incident at the vertices $i, j \in V$ and i, j are called the endpoints of edge (i, j). Graph G is called a complete graph if for all pairs of vertices $i, j \in V$ there exists an edge $(i, j) \in E$. Figure 1 shows a complete graph on four vertices. Real values such as costs and capacities can be assigned to the edges and vertices of a graph. If $\exists\ (i, j) \in E$ then vertices i and j are called adjacent to each other and are said to be neighbors. If $V' \subseteq V$ and $E' \subseteq E$ then graph $G' = (V', E')$ is called as *subgraph* of graph $G = (V, E)$. $G' = (V', E')$ is an *induced subgraph* of $G = (V, E)$ if, E' contains each edge of E with both endpoints in V'. $G' = (V', E')$ is a *spanning subgraph* of $G = (V, E)$ if, $V' = V$ and $E' \subseteq E$. The *degree* of a vertex v denoted by $D(v)$ is the number of edges incident at it. $d_G(i, j)$ denotes the length of a shortest path in terms of number of edges between vertices i and j in G, and $diam(G) = \max d_G(i, j), \forall\ (i, j) \in V$ is the *diameter* of graph G. For example, in Figure 2 degree of vertex 1 is 3 and diameter of graph is 2.

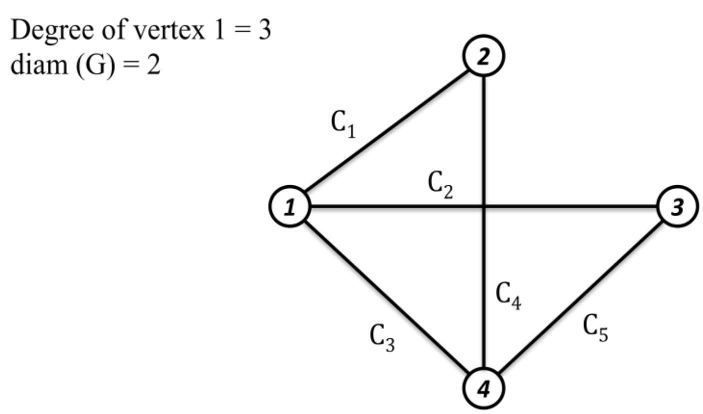


Figure 2. Degree and diameter of graph

The *vertex connectivity* of graph G denoted by $\kappa(G)$ is the minimum number of vertices whose removal from the graph results in a disconnected or trivial graph. The *edge connectivity* of graph

G denoted by $Ҡ'(G)$ is the minimum number of edges whose removal from the graph results in a disconnected or trivial graph. These graph connectivity parameters obey the inequality $Ҡ(G) \leq Ҡ'(G)$. Readers are referred to the texts by West [31] and Diestel [10] for an introduction to graph theory.

I.1. Research Overview

This thesis discusses a combinatorial optimization problem called as *the minimum spanning k-core problem* introduced by Balasundaram in [4] and studies it under deterministic settings as well as probabilistic settings. The minimum spanning *k*-core problem is motivated by hub network design problems, where a set of designated hubs need to be connected in a reliable manner. The hubs can represent airports, warehouses or distribution centers. In our model, the design element is the edge set, as focus of our model will be on designing underlying network to meet required conditions on connectivity and diameter. To achieve structural specifications we are using properties of a graph theoretic structure called ***k*-core,** a graph is said to be a *k*-core if $D(v) \geq k, \forall v \in V$. Given an undirected graph $G=(V,E)$ and a fixed positive integer *k*, the minimum spanning *k*-core problem is to identify a minimum cost set of edges E^*, so that the resulting n vertex graph $G^*=(V, E^*)$ is a *k*–core. By an appropriate choice of parameter *k*, one can ensure that the designed network is robust under node or edge failure and it has diameter 2. Briefly, our model aims to balance the network design objectives of robustness (high vertex connectivity), reachability (navigation between vertices in fewer steps) and cost effectiveness by using the notion of *k*-cores.

I.2. Applications

Recall, the three design objectives reachability, robustness and cost effectiveness of a minimum spanning *k*-core that we have discussed in the previous section. Based on these three objectives minimum spanning k-core problem is applicable in many situations. For example, transportation networks, telecommunication networks, electrical and power distribution networks etc. To get meaningful insight about applicability of the minimum spanning k-core problem we will discuss an example of a general transportation network.

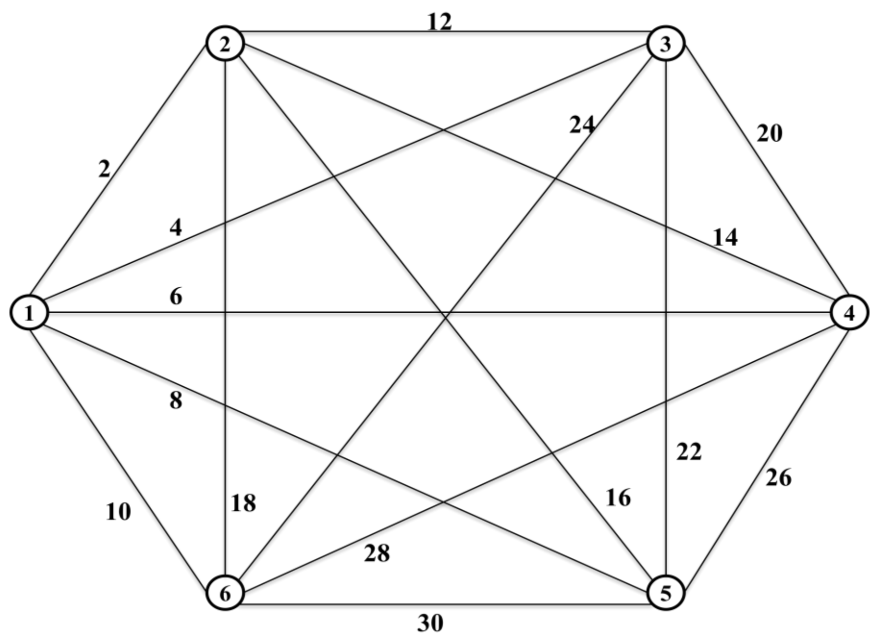


Figure 3. Example graph
Sum of all edge weights = 240; |E|=15

Let us consider an undirected and complete graph G = (V, E) as shown in Figure 3, where G represents a general transportation network with $|V|= 6$, $|E| =15$, $k = |V|/2 = 3$ and respective edge weights. Detailed discussion over selecting an appropriate value of k is given in Chapter II. Vertices could represent airports, warehouses or cities and edges represent routes connecting these vertices. Edge weights represent the cost of transportation between vertices. Solving the minimum spanning k-core problem on the given graph G will identify a subset E* of edges so that the resulting subgraph $G^* = (V, E^*)$ is a minimum spanning k-core as shown in Figure 4.

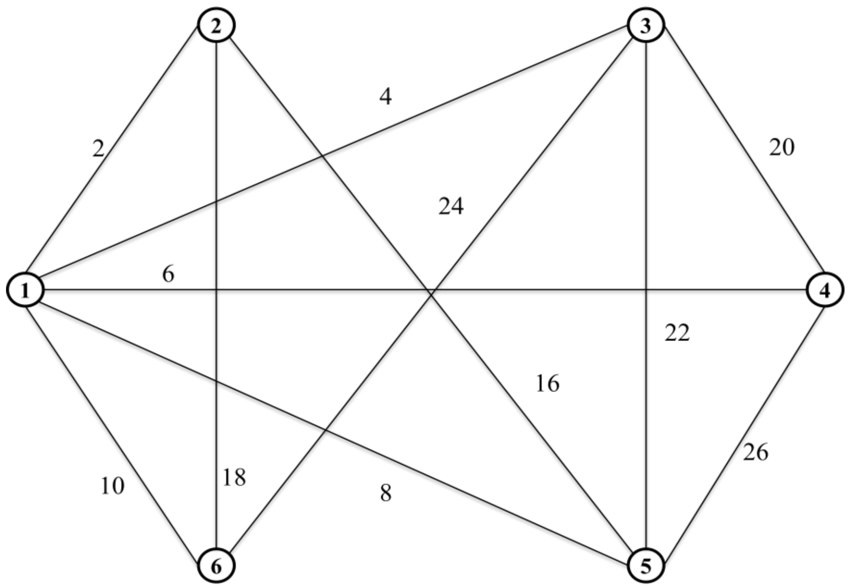


Figure 4. Example design
Total cost of network = 130; /E/=11*

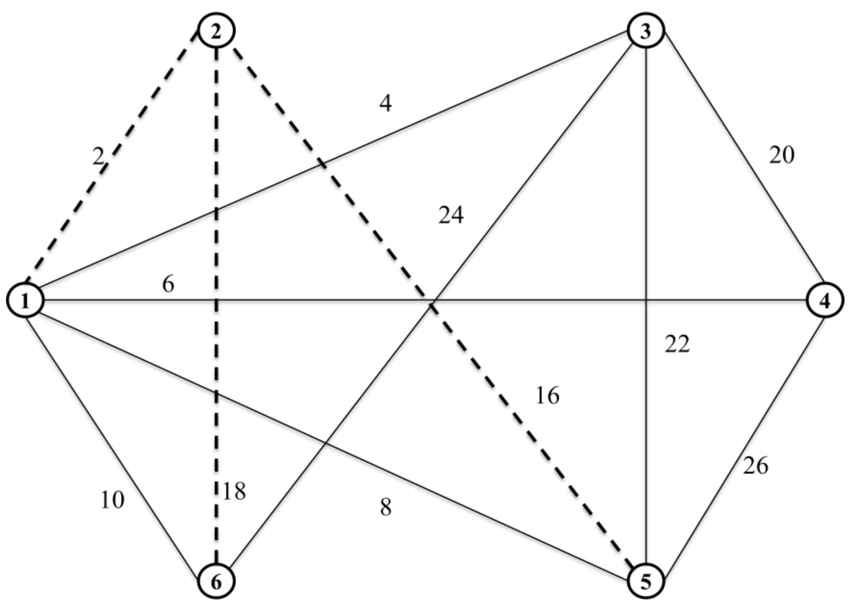


Figure 5. Illustration of robustness

The minimum spanning *k*–core, G* satisfies all desired properties such as reachability, robustness and cost effectiveness as explained in the following text.

- Every node is reachable in 2 or fewer steps from every other node, i.e. $\mathrm{diam}\,(G^*) \leq 2$.
- Cost of the design is 130 whereas a complete point-to-point design costs 240.
- Robustness/vertex connectivity of network design is illustrated with the help of Figure 5. Suppose there is breakdown of vertex 2, we can still travel between other vertices in less than or equal to 2 steps. Also, note that the breakdown of any specific vertex will result in failure of edges incident at that vertex.

So far we have discussed the problem under deterministic settings, where we have identified a subset E* of the edges such that $G^*=(V, E^*)$ satisfies reachability, robustness and cost effectiveness objectives. Let's consider probabilistic settings where each edge exists with a probability p_e along with a cost c_e and is subject to probabilistic failure. We consider two types of edges with probabilities p_1 and p_2 in the graph such that $p_1 > p_2$. The degree of a node becomes a result of the sum of all incident independent p_1 and p_2 "trials" which makes the degree a random variable. The goal of the probabilistic version is to select sufficient number of edges incident at every node such that the probability of a particular node's degree being greater than or equal to k is above a prescribed probability level. Chapter II explains the probabilistic version in more detail.

I.3. Research Objective and Contributions

The overall goal of this thesis is to develop metaheuristic algorithms that solve the deterministic and probabilistic versions of minimum spanning k-core problem on large scale instances. Greedy Randomized Adaptive Search Procedure (GRASP), a metaheuristic introduced by Feo and Resende [15] has been successfully applied to various combinatorial optimization problems, such as set covering, location problems, flow shop scheduling, routing problems and production planning [26]. Festa and Resende [16, 17] provided an annotated bibliography of the GRASP literature from 1989 to 2008. Since this is the first application of GRASP to the minimum spanning k-core problem, this thesis can provide guidance on efficiency of GRASP in solving the minimum spanning k-core problem.

Contributions: Broadly, this thesis makes the following contributions. The mathematical formulations for both deterministic and probabilistic versions of the problem are studied and the formulation of the deterministic version is implemented in Xpress. We have developed GRASP algorithms to solve both versions of the minimum spanning k-core problem. Effective local search phases for both versions are developed by designing appropriate neighborhood definitions. GRASP algorithms for both versions of problem have been implemented in the C++ programming language and extensive computational experiments carried out on the implementation to study the performance of the developed algorithms. We have identified techniques to improve overall algorithmic implementation by designing appropriate data structures. Large test bed of instances for C++ and Xpress implementations are created using MATLAB.

I.4. Thesis Organization

The rest of this thesis is organized as follows. Chapter II formally describes the research problem that includes the selection of network design parameter k and mathematical programming formulations of the problem. Chapter II also describes the relationship of the research problem to the classical matching problem. Chapter II concludes with the explanation over the need for metaheuristics. Details of algorithmic approach are given in Chapter III, which includes a thorough description of GRASP algorithm for both the versions of minimum spanning k-core problem. Results of computational experiments performed on test-bed are given in Chapter IV. Finally, we conclude this thesis with Chapter V which presents a brief summary of the research, important conclusions and future research directions.

CHAPTER II

THE MINIMUM SPANNING *k*-CORE PROBLEM

Network design is an important problem in designing robust transportation and distribution systems. Some important properties to consider while designing such networks that have a high impact on its efficiency and robustness are:

(a) Reachability, i.e. transportation between vertices in fewer steps.

(b) Cost of transportation between vertices.

(c) Robustness of network structure i.e., removal of few vertices from the network should not disconnect the network.

(d) Survivability of the network structure under probabilistic failure of edges or vertices.

The k-core model was introduced by Seidman [28] as a measure of "network cohesion" in social network analysis [30]. This model aims to detect a robust cluster with specified structural properties such as vertex connectivity (a measure of robustness) and diameter (a measure of reachability). The problem studied in this thesis is an extension of k-core based network model called *the minimum spanning k-core problem* introduced by Balasundaram in [4].

II.1. *k*-Cores

A graph is called a *k*-core if every vertex has at least *k* neighbors. In other words, the minimum degree of G is at least *k*. *k*-Cores were introduced by Seidman in 1983 [28] as a model for simplifying interconnections of the graph elements to aid in analysis. Seidman's goal was to identify regions of the social network containing "tightly knit" social subgroups.

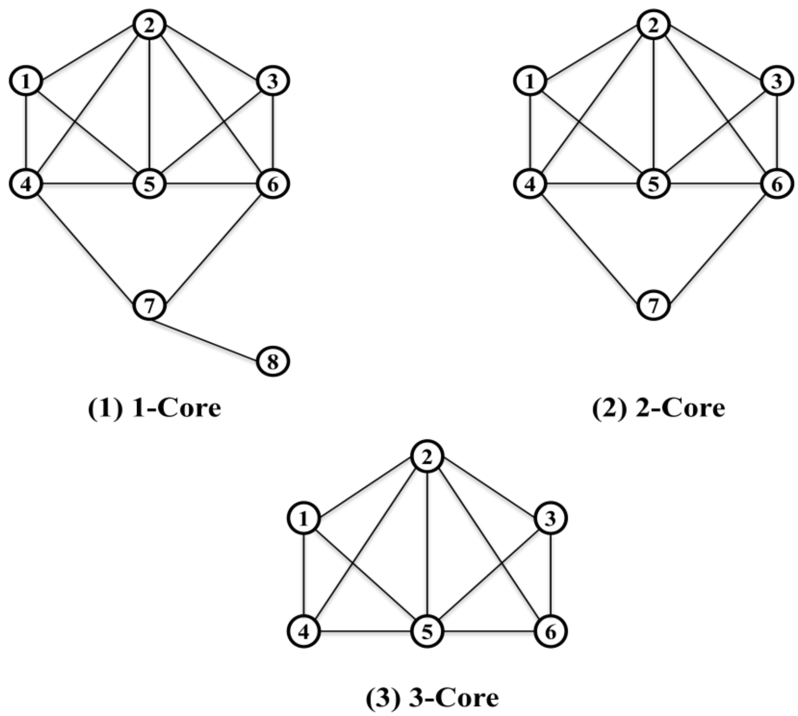


Figure 6. An illustrative example of *k*–cores

Figure 6 illustrates an example that can help readers to understand concept of *k*-cores in graph theory. Figure 6-(1) is a simple undirected graph G with minimum degree of 1 at node 8; hence graph G is a 1-core. In Figure 6-(2) node 8 is deleted, which results in a graph which is a 2-core with minimum degree of 2 at node 7. Finally, deletion of node 7 results in graph which is a 3-core with minimum degree of 3.

II.1.1. Choosing Appropriate k

While designing a network to be minimum spanning k–core, two properties of interest to us are vertex connectivity and diameter of graph. Following propositions derived by Seidman in [28] are important to choose an appropriate k and to design a k-core on n vertices with prescribed connectivity and diameter. In general, solving the minimum spanning k-core problem on graph G is to identify a set of edges E* so that the graph G* = (V, E*) satisfies the vertex degree requirement and the total cost of edges created is minimized for some "appropriately" chosen k.

Proposition 1 [Seidman, 1983]:

Let $G = (V, E)$ be a k-core on n vertices. If $k \geq \max\,[r, \frac{n+r-2}{2}]$, then $Ꝁ(G) \geq r$.

Proposition 2 [Seidman, 1983]:

Let $G = (V, E)$ be a k-core on n vertices. If $k > \frac{n-2}{2}$ then $diam\,(G) \leq 2$.

Proposition 3 [Seidman, 1983]:

Let $G = (V, E)$ be a k-core on n vertices with $Ꝁ(G) = r$ with $1 \leq r \leq k < n$.

If $k \leq \frac{n-2}{2}$ then $diam\,(G) \leq 3\left\lfloor \frac{n-2k-2}{\beta} \right\rfloor + b\,(n, k, r) + 3$ where $\beta = \max\{k+1, 3r\}$ and,

$$b\,(n,k,r) = \begin{cases} 0;\ if\ n-2k-2 (mod\ \beta) < r \\ 1;\ if\ r \leq n-2k-2 (mod\ \beta) < 2r \\ 2;\ if\ 2r \leq n-2k-2 (mod\ \beta) \end{cases}$$

Based on the propositions, if we require $Ꝁ'(G^*) \geq Ꝁ(G^*) \geq 2$ and $diam(G^*) \leq 2$, we can choose $k = \left\lceil \frac{n}{2} \right\rceil$. Furthermore, if $r \geq 2$ and $k = \left\lceil \frac{n+r-2}{2} \right\rceil$ then,

(1) $Ꝁ(G^*) \geq r$ and $diam(G^*) \leq 2$;

(2) G*-v is a (k-1)-core for any $v \in V$;

(3) $Ꝁ(G^*\text{-}v) \geq r-1$ and $diam(G^*\text{-}v) \leq 2$;

II.2. The Minimum Spanning k-Core Problem (MSkC Problem)

Given the vertices of network to be designed as $V = \{1, \ldots, n\}$, an appropriately chosen fixed positive integer k, set of candidate edges E and the cost c_e of creating an edge $e \in E$, the minimum spanning k-core problem is to identify a subset E* of edges, so that the resulting n-vertex graph $G^* = (V, E^*)$ is a k-core and the total cost of edges included in G* is minimized. The following is a binary integer programming (IP) formulation for the MSkC problem.

Decision Variables: Binary Variables: x_e for every edge $e \in E$
$x_e = 1$, if edge e is selected to be in the subgraph G*
$x_e = 0$, otherwise

$$\min \sum_{e \in E} c_e . x_e$$

$$S.T.$$

$$\sum_{e \in \partial(v)} x_e \geq k, \forall v \in V$$

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

Where, $\partial(v)$ is the set of edges incident at node v. This formulation ensures that every node has at least k incident edges/neighbors, while the overall cost of network is minimized.

II.2.1. The Maximum Weighted *b*-Matching Problem

It is necessary to understand the relationship between the maximum weighted b-matching problem and MSkC problem. Given a simple undirected graph $G=(V, E)$ and a vector b, a b-matching is a subset M of the edges such that every vertex $v \in V$ is incident with at most $b(v)$ edges in M. Hence, the maximum weighted b-matching problem defined on G with edge weights c_e for all $e \in E$ is to find a b-matching 'M', such that the total cost of edges added to M is a maximum. The following is a binary IP formulation for the maximum weighted *b*-matching problem.

$$\max \sum_{e \in E} c_e . x_e$$

$$S.T.$$

$$\sum_{e \in \partial(v)} x_e \leq b(v), \forall v \in V$$

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

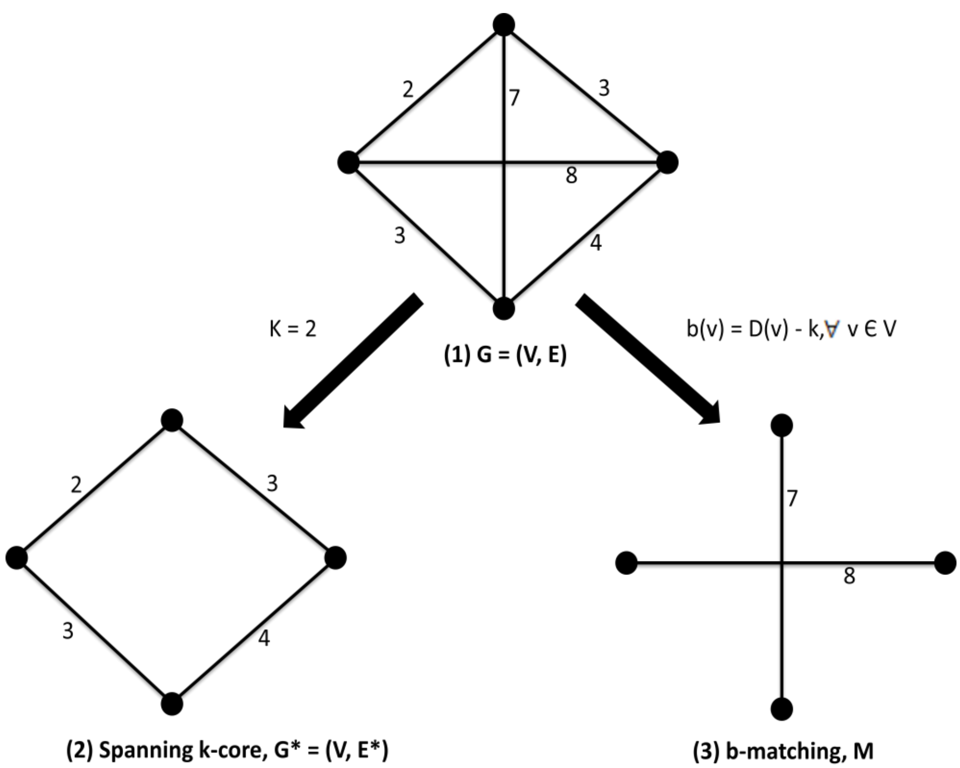


Figure 7. Spanning *k*-core to *b*-matching reduction

It was shown by Balasundaram [4] that the MSkC problem is polynomial-time solvable by reduction to a special case of the maximum weighted *b*-matching problem [9]. This reduction follows from the observation that by solving maximum weighted *b*-matching on G for a given vector b, identifies those edges that must be excluded in MSkC problem. In other words, the edges which are excluded in a maximum weighted b-matching will be the edges of a minimum spanning k-core. Figure 7 illustrates the reduction of the MSkC problem to the maximum weighted b-matching problem. Note that by solving the maximum weighted b–matching problem on a given G (refer Figure 7-(1)) for a given vector b = {1, 1, 1, 1} identified edges with costs 7 and 8 (refer Figure 7-(3)) that are excluded in MSkC (refer Figure 7-(2)). In other words, the edges with costs 2, 3, 3 and 4 which are excluded in the maximum weighted b-matching are the edges of MSkC. The reduction of the MSkC problem to the generalized matching problem is, when every node has at most (D(v)-k) incident edges. More specifically, in the binary IP formulation of the maximum weighted b-matching problem discussed in the previous section, b(v) is equal to (D(v)-k) as shown in the following formulation.

$$\max \sum_{e \in E} c_e . x_e$$

$$S.T.$$

$$\sum_{e \in \partial(v)} x_e \leq (D(v) - k), \forall v \in V$$

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

Edmonds and Pulleyblank [9, 11] have provided a pseudo-polynomial algorithm for solving the maximum weighted b-matching. Also, Anstee [3] has provided a strongly polynomial time algorithm for solving the maximum weighted b-matching problem. Hence, following the reduction, the MSkC problem is also polynomial-time solvable by using the maximum weighted b-matching algorithm.

II.3. Chance Constrained Minimum Spanning *k*-Core Problem (CCMSkC Problem)

So far we have discussed the deterministic version of the problem, where we wish to identify a MSkC in a given graph. Now consider the case in which one or more structural components (edges or nodes) of the obtained MSkC will fail due to some reasons. For example, in an airline network emergency breakdown of important hubs will result in lack of connectivity with adjacent airports. Such failure of structural components will violate the desired properties of vertex connectivity (reachability) and diameter (robustness) of MSkC. We focus our attention on probabilistic edge failures in this thesis. Consider the Erdős–Rényi model [12-14] which is denoted by G(n,p), where every possible edge is present independently with uniform probability p. To be more specific, the presence or absence of an edge between two vertices is independent of the presence or absence of any other edge i.e. each edge occurs independently with probability p. Recall that the number of edges incident at a vertex is called the degree D(v) of that vertex, and has a binomial probability distribution given as follows:

$$\Pr\left(\mathrm{D(v)} = k\right) = \binom{\mathrm{n}-1}{k} * \mathrm{p}^{k} * (1-\mathrm{p})^{\mathrm{n}-1-k} \qquad \textbf{(2.1)}$$

where,

n-1 = Maximum possible number of incident edges in G (n, p)

D(v) = Degree of node v, v ∈ V

II.3.1. Randomness of Vertex Degree

Suppose we are solving the MSkC problem on a complete graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ with $|\mathrm{V}| = \mathrm{n}$ and $|\mathrm{E}| = \binom{\mathrm{n}}{2} = \mathrm{m}$. Given that, $\mathrm{x} \in \{0, 1\}^{\mathrm{m}}$ is a decision vector of the edges to be included in a solution to the MSkC problem, that is:

$$\mathrm{x_e} = \begin{cases} 1; \text{ if edge e is in the solution} \\ 0; \text{ if edge e is not in the solution} \end{cases}$$

Now suppose, set E is random, and each $e \in E$ exists with probability p and an indicator random variable Y_e such that:

$$Y_e = \begin{cases} 1; \text{ if edge e exists} \\ 0; \text{ if edge e is doesn't exist} \end{cases}$$

Indicator random variable Y_e forms the components of the random vector $Y \in \Omega$, where $\Omega \in \{0, 1\}^m$ is the sample space corresponding to all possible graphs on n vertices. Hence, given the decision vector x and the realization vector Y of set E, the realized network solution has the edge e if and only if, $x_e Y_e = 1$ which makes the degree a random variable. Recall from Section II.2, the degree constraint of the deterministic formulation. We required that the degree D(v) of every vertex v must be greater than or equal to k,

$$D(v) = \sum_{e \in \partial(v)} x_e \geq k, \forall v \in V$$

In the probabilistic version as each edge e has a probability of existence p in terms of indicator random parameter Y_e. The degree constraint changes to,

$$D(v) = \sum_{e \in \partial(v)} x_e . Y_e \geq k, \forall v \in V$$

This change in the degree constraint converts D(v) into a random variable and one cannot guarantee $D(v) \geq k, \forall v \in V$, in every realization of the random vector. The probability that D(v) is greater than or equal to k can be calculated by using binomial probability distribution as follows:

$$\Pr(D(v) \geq k) = \sum_{i=k}^{T} \binom{T}{i} * p^i * (1-p)^{T-i} \qquad \textbf{(2.2)}$$

where, $T = \sum_{e \in \partial(v)} x_e =$ the number of edges incident at vertex v in the solution x.

II.3.2. Chance Constrained Programming

Chance constrained programming is one of the approaches available to deal with uncertainty in optimization problems. Chance constrained programming is applicable to models where (optimal) decisions have to be taken prior to realizing random effects. The constraints involving random parameters can be violated due to uncertainty and it becomes difficult to find a feasible decision which would certainly eliminate constraint violation caused by unexpected events. Under this framework, we can rewrite the degree constraint as,

$$\Pr(\mathrm{D}(\mathrm{v}) \geq k) \geq \beta \,, \forall \ \mathrm{v} \in \mathrm{V} \tag{2.3}$$

Here, $\mathrm{D}(\mathrm{v})$ is the degree of a vertex v which is a binomial random variable as discussed in the previous section. The value $\beta \in [0, 1]$ is the prescribed probability level which is selected as per the safety requirements of the system. It is intuitive that, higher values of β can result in fewer and higher cost feasible solutions to the problem. Equation 2.3 represents $|\mathrm{V}|$ individual chance constraints and can be calculated using Equation 2.2. Also, this constraint can be modeled as a joint chance constraints as shown in Equation 2.4.

$$\Pr\left[\wedge_{\mathrm{v} \in \mathrm{V}}(\mathrm{D}(\mathrm{v}) \ \geq \ k)\right] \geq \beta \tag{2.4}$$

In this thesis we are using individual chance constraints as we have not found an efficient way to handle dependence and calculate the joint probability. This is an important topic for future research. The binary nonlinear IP formulation with individual chance constraints is given by:

$$\min \sum_{e \in E} c_e . x_e$$

$$S.T.$$

$$\sum_{e \in \partial(v)} x_e \geq k, \forall v \in V$$

$$\Pr(D(v) \geq k) \geq \beta, \forall v \in V$$

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

II.3.3. Transforming CCMSkC Problem To MSkC Problem for $G(n, p)$

Suppose we consider the uniform random graph model $G(n, p)$ and $x \in \{0, 1\}^m$ satisfies,

$$\sum_{e \in \partial(v)} x_e . \geq k, \forall v \in V$$

Let $T = \sum_{e \in \partial(v)} x_e$,then

$$\Pr(D(v) \geq k) = \sum_{i=k}^{T} \binom{T}{i} * p^{i} * (1-p)^{T-i}$$

Hence, there either exists a t such that $\Pr(D(v) \geq k) \geq \beta$ for $T = t$ or no such t exists. Satisfying the chance constraint only depends on the number of edges added and not on which edges are added as they are all equally likely to fail. Hence, the CCMSkC problem reduces to MSkC problem with $k = t$, where $\sum_{i=k}^{t} \binom{t}{i} * p^{i} * (1-p)^{t-i} \geq \beta$ whenever such a t exists. The problem is infeasible otherwise. We now introduce a model where some edges can fail with a higher probability than others.

II.3.4. $G(n,p_1,p_2)$ Model

In this model we consider two types of edges in network, where some edges exist with lower probability p_2 and the remaining with higher probability p_1 ($p_1>p_2$). We are assuming that a greater fraction of the network edges will be higher probability edges. The edge probabilities are again assumed independent and the degree of a node becomes the sum of independent p_1 trials and p_2 trials. The probability of a node's degree being greater than or equal to k can be calculated by using Equation 2.6.

$$\Pr(D(v) \geq k) = \sum_{t=k}^{T} \Pr(D(v) = t) \tag{2.6}$$

where, $T = \sum_{e \in \partial(v)} x_e$

Let, $T_1(v)$ be the number of higher probability edges and $T_2(v)$ be the number of lower probability edges incident at v in the solution x, then

$\Pr(D(v) = t) =$

$$\sum_{\substack{0 \le a \le T_1(v) \\ 0 \le b \le T_2(v) \\ a+b=t}} \left[\binom{T_1(v)}{a} . p_1^{\,a} . (1-p_1)^{T_1(v)-a} * \binom{T_2(v)}{b} . p_2^{\,b} . (1-p_2)^{T_2(v)-b} \right] \tag{2.7}$$

Let us consider an example (refer Figure 8) where, we are calculating the probability that the degree of node 6 being at least k using Equation 2.6.

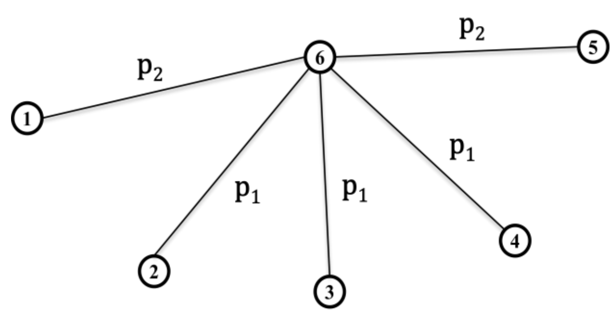


Figure 8. $G(n,p_1,p_2)$ formula example

$T_1(v)$ = Number of p_1 trials = 3

$T_2(v)$ = Number of p_2 trials = 2

T = Total number of trials = $\sum_{e \in \partial(v)} x_e$ = P1trials + P2trials = 3 + 2 = 5

Assume, $k = 3$, $p_1 = 0.9$, $p_2 = 0.3$, substituting in Equation 2.6,

$$\Pr(D(6) \ge 3) = \sum_{t=3}^{5} \Pr(D(6) = t)$$

$$\sum_{t=3}^{5} \Pr(D(6)=t) = \Pr(D(6) = 3) + \Pr(D(6) = 4) + \Pr(D(6) = 5)$$

By using Equation 2.7, we can calculate $\Pr(D(6) = 3) = 0.10206$.

II.4. Need for Metaheuristics

Various approaches have been developed to solve combinatorial optimization problems, which can either be exact or heuristic. An exact algorithm guarantees an optimal solution to the problem in a finite number of steps, whereas a heuristic algorithm does not guarantee an optimal solution but tries to provide a good solution in a reasonable amount of time. Metaheuristics form a class of algorithms that intelligently embed basic heuristic algorithms in sophisticated algorithmic frameworks that explore and exploit the search space more effectively [25]. In case of NP-hard problems, exact algorithms may take exponential computational time in worst-case. In real-world conditions we can accept solutions that are good enough for implementation and produced in a reasonable amount of time, which can be achieved through sophisticated metaheuristics algorithms.

As discussed in Section II.2.1, the MSkC problem is a polynomial-time solvable by using $O(n^4)$ complexity of maximum weighted *b*-matching problem. $O(n^4)$ running time is prohibitive for large size problems and the exact algorithm might take high computational time in worst-cases. No polynomial-time algorithm is presently available for the probabilistic version of minimum spanning *k*-core problem. Note that the formulation is a nonlinear IP. Hence, to get better solutions in reduced amount of time, metaheuristics can be a good approach to solve the MSkC and CCMSkC problems.

CHAPTER III

GREEDY RANDOMIZED ADAPTIVE SEARCH PROCEDURE

The Greedy Randomized Adaptive Search Procedure introduced by Feo and Resende [15] is an iterative, multi-start, heuristic procedure, where each iteration consists of two phases, the greedy randomized construction phase and the local search phase. In the greedy randomized construction phase, an initial feasible solution is constructed by randomly choosing elements from the *Restricted Candidate List* (RCL). RCL consists of only best elements of the candidate list selected by a greedy function. The second phase of GRASP is the local search phase that is applied for further improvement of the solution generated by the GRASP construction, as the solution obtained by the construction phase is not guaranteed to be the local optimum. At the end of each GRASP iteration best solution is updated and a final solution is obtained when GRASP completes a fixed number of iterations. This chapter explains in detail the overall procedure of GRASP for the minimum spanning k-core problem in deterministic and probabilistic settings. The overall procedure of the GRASP is shown in Algorithm 1 which is similar for both deterministic and probabilistic versions. The difference in the procedure is at the construction and local search phase of the algorithm.

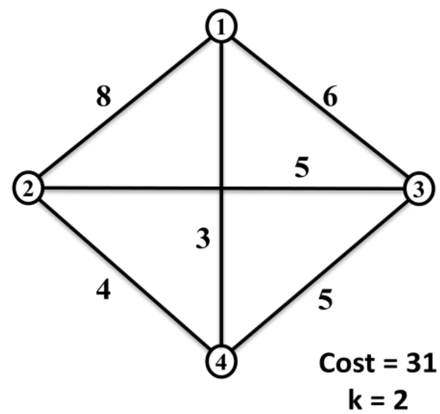
GRASP construction phase builds an initial feasible solution based upon the structural properties of the minimum spanning k-core problem. This initial feasible solution is further improved by investigating appropriate neighborhoods of the feasible solution in the local search phase. Finally, the best solution among the iterations will be return by GRASP as feasible solution. The stopping criterion for GRASP is number of iterations. Larger number of GRASP iterations increases computational time, but increases the possibility of finding better quality solution.

```
Procedure GRASP
    for i = 0, . . . . , Number of GRASP Iterations do
        Randomly generate 'α'
        StartSolution = GreedyRandomizedConstruction (α)
        LocalOptimal = LocalSearch (StartSolution)

        if Local Optimal is Better Than Best Solution then
            BestSolution = LocalOptimal
        end
    end

    return BestSolution
end GRASP
```

Algorithm 1. GRASP framework

III.1. Construction Phase

Initially, the solution is an empty set and construction phase adds candidate edges to the solution. Construction phase terminates when the solution achieves the desirable structural properties of the problem under consideration (i.e., becomes feasible). We now illustrate this approach on an example.



Edge Index	1	2	3	4	5	6
Edges	(1, 2)	(1, 3)	(1, 4)	(2, 3)	(2, 4)	(3, 4)
Costs	8	6	3	5	4	5

Figure 9. Input graph and data for construction phase

Step 1 Building a candidate list: The algorithm ranks candidate edges by using a greedy rule. As our objective is to minimize overall cost of network, algorithm sorts all edges in ascending order of costs as shown in Table 1.

Edge Index	3	5	4	6	2	1
Edges	(1, 4)	(2, 4)	(2, 3)	(3, 4)	(1, 3)	(1, 2)
Costs	3	4	5	5	6	8

Table 1. Candidate list

Step 2 Building a restricted candidate list (RCL): In this step, the algorithm acts greedy by constructing RCL which consists of only best quality candidate edges from candidate list. RCL is associated with a 'Threshold Parameter' α ($0 \leq \alpha \leq 1$), generated randomly at the start of each GRASP iteration. A threshold value is then calculated as $C_{min} + \alpha\ (C_{max} - C_{min})$, where C_{min} and C_{max} are the minimum and maximum edge costs in the candidate list respectively. Based on the threshold value, a candidate edge is selected from the RCL at random to add to the solution.

Step 3 Updating Candidate List: Edge added to the solution in step 2 is removed from the candidate list and the algorithm reselects minimum cost (C_{min}) and maximum cost (C_{max}) edges.

Algorithm repeats Step 2 and Step 3 until degrees of all vertices become at least k. In Step 2, if α is equal to 0 then the behavior of the algorithm will be purely greedy and the algorithm always selects a minimum cost edge from the candidate list. On the other hand, if α is equal to 1 then the algorithm will randomly select edge from the candidate list. Following sections describe construction phase using purely greedy (α = 0), and purely randomized (α = 1) and greedy-randomized ($0 \leq \alpha \leq 1$) approaches for the graph given in Figure 9.

III.1.1. Greedy Construction Phase

Consider a construction phase iteration with α = 0. This results in a threshold value of C_{min}. Initially, candidate list consists of all the edges as shown in Table 1. As α = 0, in step 2 the algorithm constructs an RCL that consists of edges with cost less than or equal to C_{min}. Algorithm makes a greedy choice and selects a minimum cost edge to add into solution. Finally, in step 3 candidate list is updated by removing the edge added in step 2 and algorithm reselects C_{min} and C_{max}. Steps 2 and 3 repeat until degrees of all the vertices of the solution become at least k.

Candidate List	C_{min}	C_{max}	Threshold Value	RCL	Solution
{1,2,3,4,5,6}	3	8	3	{3}	{3}
{1,2,4,5,6}	4	8	4	{5}	{3,5}
{1,2,4,6}	5	8	5	{4,6}	{3,5,4}
{1,2,6}	5	8	5	{6}	{3,5,4,6}
{1,2}	6	8	6	{2}	{3,5,4,6,2}

Table 2. Greedy construction phase solution

The details of the steps carried out in the purely greedy construction phase to construct an initial feasible solution for the graph in Figure 9 are given in Table 2 where RCL, Solution and

Candidate List are in terms of edge index. Greedy construction phase for the minimum spanning *k*-core problem returns an initial feasible solution. Then we drop edges starting heaviest first until the solution is minimal (i.e., at least one end point has degree = *k*), i.e. solution set $S = \{3, 5, 4, 2\}$ with minimized cost of 18 as shown in Figure 10.

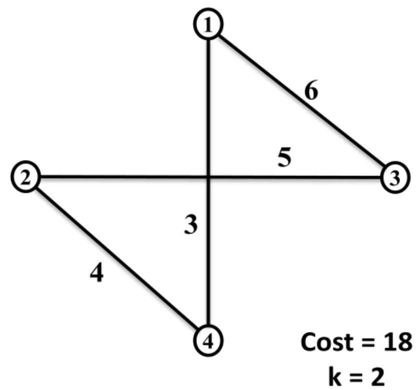


Figure 10. Greedily constructed solution

III.1.2. Randomized Construction Phase

Consider a construction phase iteration with $\alpha = 1$. This results in a threshold value of C_{max} and RCL is the original candidate list. Algorithm's behavior will be completely random as it selects any candidate edge to add into solution. Steps 2 and 3 repeat until degrees of all the vertices of the solution become at least *k*.

Candidate List	C_{min}	C_{max}	Threshold Value	RCL	Solution
{1,2,3,4,5,6}	3	8	8	{1,2,3,4,5,6}	{1}
{2,3,4,5,6}	3	6	6	{2,3,4,5,6}	{1,4}
{2,3,5,6}	3	6	6	{2,3,5,6}	{1,4,6}
{2,3,5}	3	6	6	{2,3,5}	{1,4,6,2}
{3,5}	3	4	4	{3,5}	{1,4,6,2,3}

Table 3. Randomized construction phase solution

The details of the steps carried out in the purely randomized construction phase to construct an initial feasible solution for Figure 9 graph are given in Table 3. Randomized construction phase for minimum spanning k-core problem returns an initial feasible solution, which can then be made minimal by greedily dropping edges, i.e. solution set $S = \{1, 4, 6, 3\}$ with minimized cost of 21 as shown in Figure 11.

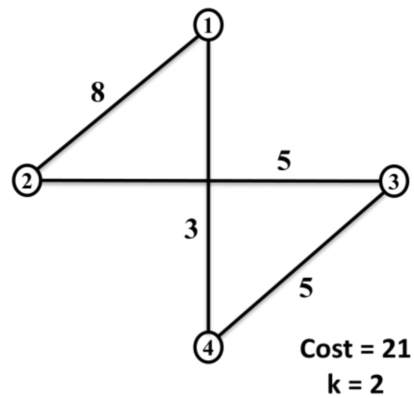


Figure 11. Randomly constructed solution

III.1.3. Greedy Randomized Construction Phase

Consider a construction phase iteration with α = 0.5. Initially, candidate list consists of all the edges as shown in Table 1. As α = 0.5, in step 2 algorithm construct *RCL* that consists of edges with cost less than or equal to $C_{min} + \frac{1}{2}(C_{max} - C_{min})$. Now, the algorithm will randomly select an edge with cost less than or equal to threshold value to add into the *solution*. Finally, in step 3 the *candidate list* is updated by removing the edge added in step 2 and the algorithm reselects C_{min} and C_{max}. Steps 2 and 3 repeat until degrees of all the vertices of the solution become at least k.

Candidate List	C_{min}	C_{max}	Threshold Value	RCL	Solution
{1,2,4,5,6}	3	8	6	{2,3,4,5,6}	{3}
{1,2,5,6}	4	8	6	{2,4,5,6}	{3,4}
{1,2,6}	4	8	6	{2,5,6}	{3,4,5}
{1,6}	5	8	7	{2,6}	{3,4,5,2}

Table 4. Greedy-randomized construction phase solution

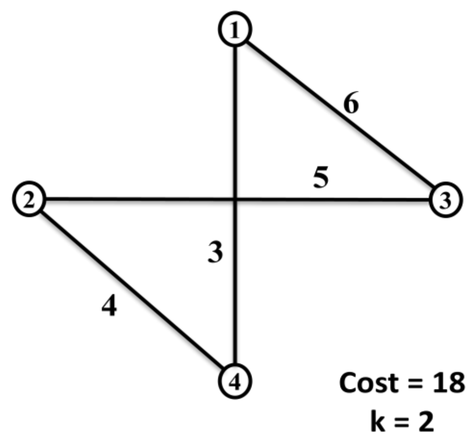


Figure 12. Greedy-randomized solution

The details of the steps carried out in the greedy-randomized construction phase to construct an initial feasible solution for the graph in Figure 9 are given in Table 4. Greedy randomized construction phase for the minimum spanning *k*-core problem returns an initial feasible solution, i.e. Solution set S = {3, 4, 5, 2} with minimized cost of 18 as shown in Figure 12.

In case of purely greedy approach, RCL is restricted to only the minimum cost edges that can result in addition of extra edges to satisfy structural properties of problem especially for the CCMSkC problem. Also, completely randomized approach may not include good quality edges and result in inferior solutions to the problem. On the other hand, greedy-randomized approach allows the algorithm to balance both cost minimization and degree requirements of minimum spanning *k*-core problem. In other words, greedy-randomized approach can potentially provide superior solutions as compare to completely greedy or randomized approaches. This is the reason behind using greedy randomized construction approach in GRASP for the construction phase. Note that GRASP is a multi-start heuristic and for each iteration, α is generated randomly between 0 and 1.

Algorithm 2 and Algorithm 3 are the detailed greedy randomized construction phases of the MSkC and CCMSkC problems respectively. The input data for both algorithms is an undirected graph $G = (V, E)$. The output of Algorithm 2 is a *minimal* spanning k-core that ensures degree of every vertex $v \in V$ is at least k and there is no edge in the solution with both end points greater than k. Furthermore, the output of Algorithm 3 is a *minimal* spanning k-core that ensures every vertex $v \in V$ satisfies the chance constraint in addition to the degree constraint.

```
Procedure GreedyRandomizedConstruction (α)
    Initial SolutionSet = ∅
    Initial candidate list, C_L= E
    while MinDeg < k do
        C_min = min{ C_e | e ∈ C_L }
        C_max = min{ C_e | e ∈ C_L }
        RCL = { e ∈ C_L | C_e ≤ C_min + α (C_max – C_min)}
        Select an edge s from the RCL at random
        SolutionSet = SolutionSet ∪ {s}
        Increment degrees of both endpoints of edge s by 1 in the degree list
        MinDeg = Minimum degree in the degree list
    end
    Find minimal spanning k-core by deleting all edges with both endpoints greater than k
    return SolutionSet
end GreedyRandomizedConstruction
```

Algorithm 2. Construction phase for the MSkC problem

Initially, the construction phase builds a candidate list that includes all the edges of the given graph $G = (V, E)$. In the next step algorithm constructs RCL which consists only of lower cost edges from the candidate list. RCL is built using a threshold parameter α which is generated randomly in the range of 0 to 1. A threshold value is calculated as $C_{min} + \alpha\ (C_{max} - C_{min})$, where C_{min} and C_{max} are the minimum and maximum edge costs from the candidate list respectively. An edge is randomly selected from RCL to include in the solution. Once the selected edge is added to the solution, the algorithm updates the candidate list for the next iteration. Iterations terminate when degrees of all the nodes become greater than or equal to *k*. The steps explained up to this point are similar for both Algorithm 2 and the first phase of Algorithm 3. In the last step, Algorithm 2 deletes all the edges with both endpoints greater than *k* and identifies a *minimal* spanning *k*-core.

```
Procedure GreedyRandomizedConstruction (α)
    SolutionSet = Algorithm 2 solution; excluding step of finding minimal spanning k-core
      for i=0, . . . , Number of Nodes do
              Calculate Pr(D(i) ≥ k)
              β = Prescribed probability level
              while Pr(D(i) ≥ k) < β and ∂(i)\SolutionSet ≠ ∅ do
                    j = argmin{c_e| e ∈ ∂(i)\SolutionSet}
                    SolutionSet = SolutionSet ∪ {j}
              end

              if Pr(D(i) ≥ k) < β do
                 terminate GRASP
                 return "infeasible"
              end
      end

      Find minimal spanning k-core by deleting excess edges if,
      chance constraints of both endpoints of edges are not violated
    return SolutionSet
end GreedyRandomizedConstruction
```

Algorithm 3. Construction phase for the CCMSkC problem

Algorithm 3 is divided into two phases. The first phase is similar to Algorithm 2, excluding the step of finding a *minimal* spanning *k*-core. In the second phase Algorithm 3 satisfies all the individual chance constraints by adding sufficient lower cost edges at every vertex $v \in V$. Algorithm 3 terminates the GRASP iterations, if the individual chance constraint of a particular node is violated and all incident edges have been added. In the last step, Algorithm 3 deletes excess edges and identifies a *minimal* spanning k-core. An edge is deleted only if both endpoints still satisfy chance constraints upon deletion of the edge.

III.2. Local Search Phase

Local search phase starts with the solution from the GRASP construction phase and iteratively improves the current solution by exploring solutions in the local neighborhood. Neighborhood of a solution is a function defined on search space S (S is the set of all feasible solutions) that assigns a set of neighbors $N(s) \subseteq S$ for each $s \in S$. Set $N(s)$ is called the neighborhood of s. Solution $s^* \in S$ is called a local minimum if $f(s^*) \leq f(s), \forall\ s \in N(s^*)$ and it is called a global minimum if $f(s^*) \leq f(s), \forall\ s \in S$. Given a graph $G = (V, E)$ as input, the GRASP construction phase builds an initial feasible solution $G' = (V, E'), E' \subseteq E$. Since the solution generated by the construction phase is not necessarily local/global optimum, it is helpful to further improve the solution in the local search phase. (1,1)-exchange neighborhood is to delete an edge from the current solution and add an edge (not present in the current solution) resulting in another feasible solution. Similarly, (1,2)-exchange neighborhood is to delete an edge from the current solution and add two edges (not present in the current solution) resulting in another feasible solution. We use (1,1)-exchange and (1, 2)-exchange neighborhoods for both the versions of the problem.

III.2.1. Local Search Phase for The MSkC Problem

Let $E^0 \subseteq E$ be a feasible solution, i.e., $G^0 = (V, E^0)$ is a spanning k-core.

$N_{1,1}(E^0) = \{ E' \subseteq E \mid E' = E^0 \cup \{w\} \setminus \{u\}$, where $w \notin E^0$, $u \in E^0$ and (V, E') is a spanning k-core$\}$

$N_{1,2}(E^0) = \{ E' \subseteq E \mid E' = E^0 \cup \{v, w\} \setminus \{u\}$, where $w \neq v$, $u \in E^0$, $v,w \notin E^0$ and (V, E') is a spanning k-core$\}$

The local search neighborhood used in GRASP for the MSkC problem is $N_{1,1}(E^0) \cup N_{1,2}(E^0)$. Note that $E^0 \cup \{w\} \backslash \{u\} \in N_{1,1}(E^0)$ will correspond to a minimal solution if u,w edges are incident at the same node which has degree exactly k in the solution E^0. Similarly, $E^0 \cup \{v, w\} \backslash \{u\} \in N_{1,2}(E^0)$ will be a minimal solution if v is incident at one endpoint of u and w is incident at the other endpoint of u, where both endpoints are at degree exactly k in E^0. Such a minimal neighboring solution is an improving solution if the cost of the added edges is less than the cost of the deleted edge. Algorithm 4 is the local search phase of the MSkC problem.

Procedure LocalSearch(E^0)

while there exists $(u,w) \in N_{1,1}(E^0)$ such that $c_w < c_u$ **do**
$E^0 = E^0 \cup \{w\} \backslash \{u\}$
end

while there exists $(u,v,w) \in N_{1,2}(E^0)$ such that $(c_w + c_v) < c_u$ **do**
$E^0 = E^0 \cup \{w,v\} \backslash \{u\}$
end

return E^0
end LocalSearch

Algorithm 4. Local search phase for the MSkC problem

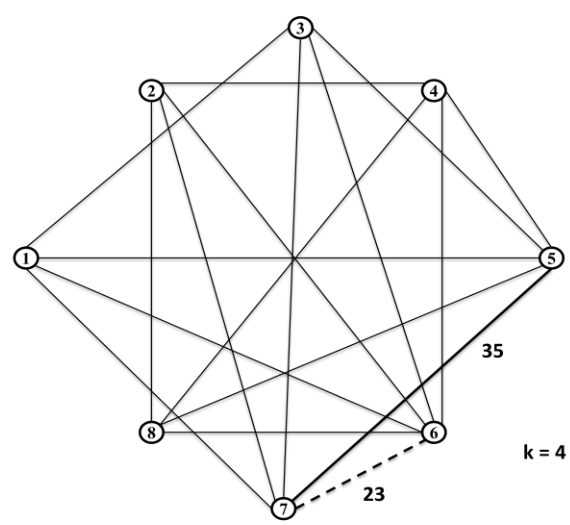


Figure 13. A feasible solution after (1,1)-exchange for the MSkC problem

Consider Figure 13, in which candidate edge to delete is $(5, 7)$ with degree of node 5 greater than k and degree of node 7 equal to k. An improving solution after (1, 1)-exchange can be found by replacing edge $(5,7)$ with the edge $(6, 7)$. Edge $(6, 7)$ is incident at node 7, not present in the current solution and the cost of edge $(6, 7)$ is less than the cost of edge $(5, 7)$. Let's assume that after this step (1, 1)-exchange terminates and feasible solution is further improved in (1, 2)-exchange neighborhood.

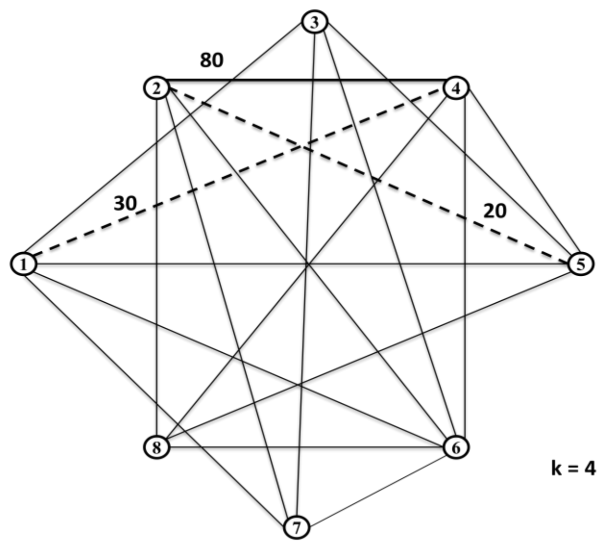


Figure 14. A feasible solution after (1,2)-exchange for the MSkC problem

An improving solution after (1, 2)-exchange is shown in Figure 14, where $(2, 4)$ is the edge to delete with degree of both endpoints equal to k. The edge $(2, 4)$ is replaced with edges $(1, 4)$ and $(2, 5)$. Both $(1, 4)$ and $(2, 5)$ edges are incident at respective endpoints of $(2, 4)$ edge, not present in current solution and sum of the costs of $(1, 4)$ and $(2, 5)$ edges is less than the cost of edge $(2, 4)$.

III.2.2. Local Search Phase for The CCMSkC Problem

Let $E^0 \subseteq E$ be a feasible solution, i.e., $G^0 = (V, E^0)$ is a spanning k-core. We use $N_{1,1}(E^0)$ and $N_{1,2}(E^0)$ as defined in Section III.2.1. A minimal neighboring solution is an improving solution if the cost of the added edges is less than the cost of the deleted edge and the probabilities of the added edges is greater than or equal to the probability of the deleted edge. Algorithm 5 is the local search phase of the CCMSkC problem.

Procedure LocalSearch(E^0)

  while there exists $(u,w) \in N_{1,1}(E^0)$ such that $c_w < c_u$, $p_w \geq p_u$ **do**
    $E^0 = E^0 \cup \{w\} \backslash \{u\}$
  end

  while there exists $(u,v,w) \in N_{1,2}(E^0)$ such that $(c_w + c_v) < c_u$, $p_w \geq p_u$, $p_v \geq p_u$ **do**
    $E^0 = E^0 \cup \{w,v\} \backslash \{u\}$
  end

  return E^0
end LocalSearch

Algorithm 5. Local search phase for the CCMSkC problem

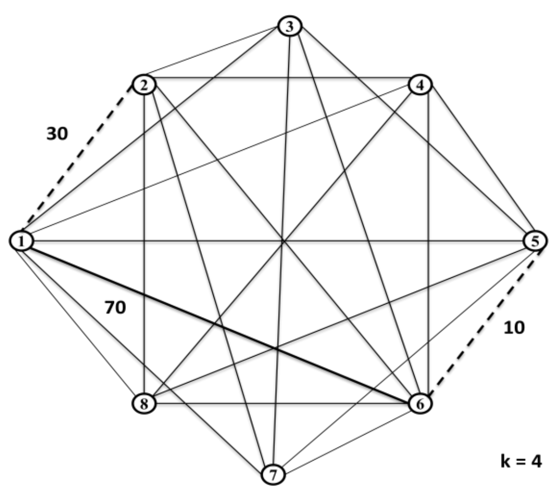


Figure 15. A feasible solution after (1,2)-exchange for the CCMSkC problem

Consider Figure 15, in which edge to delete is (1,6) with the degree of both endpoints greater than k. An improving solution after (1,2)-exchange can be found by replacing edge (1,6) with the edge (1,2) and (5,6) edges, which are incident at respective endpoints of edge (1,6). Edges (1,2) and (5,6) are not present in current solution and sum of the costs of (1, 2) and (5,6) is less than the cost of edge (1, 6). Also, consider that the probabilities of both (1,2) and (5,6) edges are at least probability of the edge (1,6).

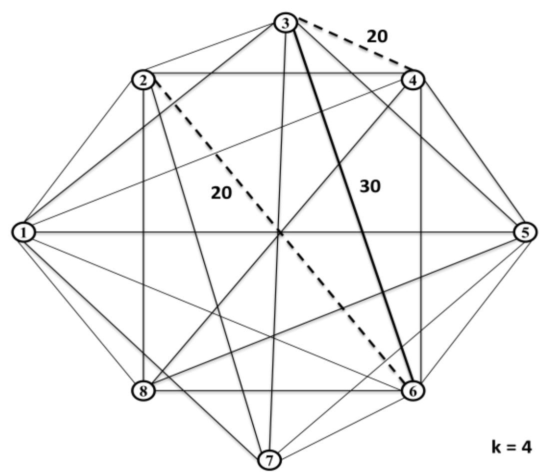


Figure 16. A feasible solution after (1,1)-exchange for the CCMSkC problem

Consider Figure 16, in which (3,6) is the edge to be deleted with the degree of both endpoints greater than k. Higher probability edges (2,6) and (3,4) incident at respective endpoints of the edge (3,6) are selected. Current solution cannot improve in an (1,2)-exchange, as sum of the costs of edges (2,6) and (3,4) is greater than the cost of edge (3,6). However an improving solution after (1,1)-exchange can be found by replacing edge (3,6) with either (2,6) or (3,4). The algorithm deletes edge (3,6) and adds edge (3,4) if chance constraint at node 6 is not violated upon deletion of the edge (3,6). In the next chapter, we discuss the computational experiment and numerical results from solving the MSkC and CCMSkC problems using the GRASP algorithm developed in this chapter.

CHAPTER IV

COMPUTATIONAL EXPERIMENTS

This chapter presents computational experiments on the GRASP algorithms developed in Chapter III. Extensive experimentation on the algorithms for the deterministic and probabilistic versions of the problem is carried out on a test-bed of instances. In Section 4.1, we describe the implementation details and in Section 4.2 we describe the instances used in testing. Section 4.3 describes the experimental design and introduces the statistics collected during the experiments. Finally, we conclude this chapter by presenting numerical results and observations.

IV.1. General Implementation Details

GRASP algorithm was implemented in the C++ programming language. All numerical experiments were conducted on Dell Precision T3500 computers with Intel Xeon W3550, 3.07 GHz processor and 3GB RAM.

A binary IP formulation for the deterministic version of problem discussed in Section II.2 is implemented in Xpress (Xpress Optimization Suite 7.0). Xpress results are important to assess the solution quality of deterministic GRASP as Xpress provides the optimal solution for the deterministic version of problem. Detailed Xpress model can be found in Section A.5 of Appendix A.

Preliminary experiments demonstrated that the updating candidate list was the most time consuming step in the construction phase. During each iteration of construction phase, we were updating candidate list and the restricted candidate list by searching for the minimum and maximum cost edges in the candidate list. To avoid searching for minimum and maximum cost edges, we designed a double dimensional array specifically to update candidate list.

Edge Index	0	1	2	3	4
Edge Cost	15	7	35	10	5

Table 5. Double dimensional array

Double dimensional array stores edge id in the first dimension and respective edge cost in the second dimension as shown in Table 5.Once this array is initialized, we applied the bubble sort algorithm to sort all the edges in ascending order of cost along with their ids as shown in Table 6.

Edge Index	4	1	3	0	2
Edge Cost	5	7	10	15	35

Table 6. Double dimensional array after bubble sort

Now algorithm directly selects minimum and maximum cost edges from double dimension array’s first and last positions respectively. If the edge at any particular array’s position is not in the candidate list, the algorithm moves to the next position of the array to find an addable edge. Table 7 shows runtime improvement due to double dimension array implementation on deterministic GRASP algorithm with 10 iterations.

Instance	Previous Runtime (sec)	Current Runtime (sec)
150 Nodes	17	9.3
200 Nodes	61	30
250 Nodes	158	78

Table 7. Double dimensional array deterministic GRASP runtime improvement

IV.2. Description of Test Bed

The test-bed of instances consisting of graphs of various sizes was generated by using MATLAB Appendix A, Section A.1. MATLAB generator produces test instances for the minimum spanning k-core problem with specified number of vertices, edges, edge costs and probabilities. The number of vertices in the generated graphs was selected as 30, 50, 70, 100, 150, 200 and 250. The number of edges for all instances was calculated as $\frac{n(n-1)}{2}$, as we are assuming input to the problem is a complete graph. The costs for edges were generated randomly and uniformly from specified ranges. We have selected the edge cost ranges as [100,500], [500,1000], [500,1500] and [1000,2000] for every test instance. Table 8 presents information regarding the 28 test instances of $G(n,p_1,p_2)$ graphs, used in our experiments for both deterministic and probabilistic algorithms. The name of an instance provides information about the number of nodes and edge cost range for that instance, for e.g. "Kcore-30-100-500.txt" is a 30-node instance and costs assigned to the edges are distributed uniformly in the range of [100,500]. Furthermore, each edge was randomly assigned either p_1 or p_2 type of probability and all test instances consists of larger proportion of p_1 type edges. In every instance, MATLAB uses a predefined fraction which controls the proportion of generating p_1 type of edges amongst the graph edges. We have used p_1 = 90%, p_2 = 30% and p_1 fraction = 80% for all the test instances. Therefore 80% of the edges are p_1 type edges and the remaining 20% are p_2 type edges. GRASP for the MSkC is tested on all test instances ignoring probability. Another MATLAB generator Appendix A, Section A.2 was used to convert all test instances into a format readable by Xpress. An instance's data file for C++ implementation and its conversion to readable Xpress-MP format by using respective MATLAB generators is given in Appendix A, Section A.3 and Section A.4.

Instance	n	m	Total Cost
Kcore-30-100-500.txt	30	435	130646
Kcore-30-500-1000.txt			327814
Kcore-30-500-1500.txt			431391
Kcore-30-1000-2000.txt			651884
Kcore-50-100-500.txt	50	1225	369059
Kcore-50-500-1000.txt			918390
Kcore-50-500-1500.txt			1231380
Kcore-50-1000-2000.txt			1854680
Kcore-70-100-500.txt	70	2415	720557
Kcore-70-500-1000.txt			1804450
Kcore-70-500-1500.txt			2413580
Kcore-70-1000-2000.txt			3625380
Kcore-100-100-500.txt	100	4950	1480550
Kcore-100-500-1000.txt			3700920
Kcore-100-500-1500.txt			4940630
Kcore-100-1000-2000.txt			7399500
Kcore-150-100-500.txt	150	11175	3370700
Kcore-150-500-1000.txt			8388520
Kcore-150-500-1500.txt			11197400
Kcore-150-1000-2000.txt			16752500
Kcore-200-100-500.txt	200	19900	5983010
Kcore-200-500-1000.txt			14934600
Kcore-200-500-1500.txt			19821800
Kcore-200-1000-2000.txt			29863100
Kcore-250-100-500.txt	250	31125	9359000
Kcore-250-500-1000.txt			23308700
Kcore-250-500-1500.txt			31161400
Kcore-250-1000-2000.txt			46725600

Table 8. Test instances

IV.3. Design of Experiments

This section discusses the experimental setup used to test the performance of both GRASP algorithms. As discussed earlier, the termination criterion for GRASP is number of iterations.

Larger number of GRASP iterations increases computational time, but increases the possibility of finding better quality solutions. After extensive experimentations we decided upon 10 GRASP iterations within which good quality solutions to both the versions of minimum spanning k-core were observed. Furthermore, to demonstrate algorithmic performance in terms of quality of solution, we have executed 1000 iterations for 30, 50, 70 and 100 node instances for both versions of GRASP. Following statistical data is collected during the computational experiments.

- *GRASP Solution*: The best solution returned by GRASP for a given instance.
- *Edges Included*: The number of edges $|E^*|$ included in the final solution given by GRASP.
- *Construction Time*: Total time spent in construction phase.
- *Local Search Time*: Total time spent in local search phase.
- *GRASP Time* = Total GRASP time.
- *LS Hit Rate*: Number of times local search improved the initial feasible solution provided by GRASP construction phase.
- LSAvgPerDec = The percentage improvement by local search averaged over all the iterations in which improvement in initial feasiwas observed.
- *Xpress Objective*: The optimal cost returned by Xpress for a given instance.
- *Optimality Gap*: Represents percentage gap between the best solution by deterministic GRASP and Xpress optimal solution for a given graph instance. The following formula has been used to compute optimality gap:

$$\text{Optimality Gap} = \frac{(\text{Best Solution} - \text{Optimal Solution})}{\text{Optimal Solution}}$$

IV.4. Numerical Results: GRASP for The MSkC Problem

Optimality gap for all test instances at 10 GRASP iterations are detailed in Appendix B, Table B.1 and vary within the range of 1% to 5%. Lesser optimality gap corresponds to the good quality GRASP solutions. Considering 1% to 5 % range of optimality gap for 10 GRASP iterations, we can conclude that the GRASP for the MSkC problem provides good quality solutions. Furthermore, to increase possibility of getting good quality solutions experiments are executed with 1000 GRASP iterations for 30, 50, 70 and 100 node instances and optimality gaps are detailed in Appendix B, Table B.2. It's noteworthy to observe reduction in the optimality gap range to 0%-3%.At larger iterations GRASP returns high-quality solutions to the problem, which is exemplified with the instances that have 0.0%, 0.1%, 0.4% and 0.6% optimality gaps. This implies algorithm can return superior quality solutions at higher number of iterations for any given instance of graph. Figure 17 shows the optimality gap for all 30, 50, 70 and 100 node instances at 10 and 1000 GRASP iterations.

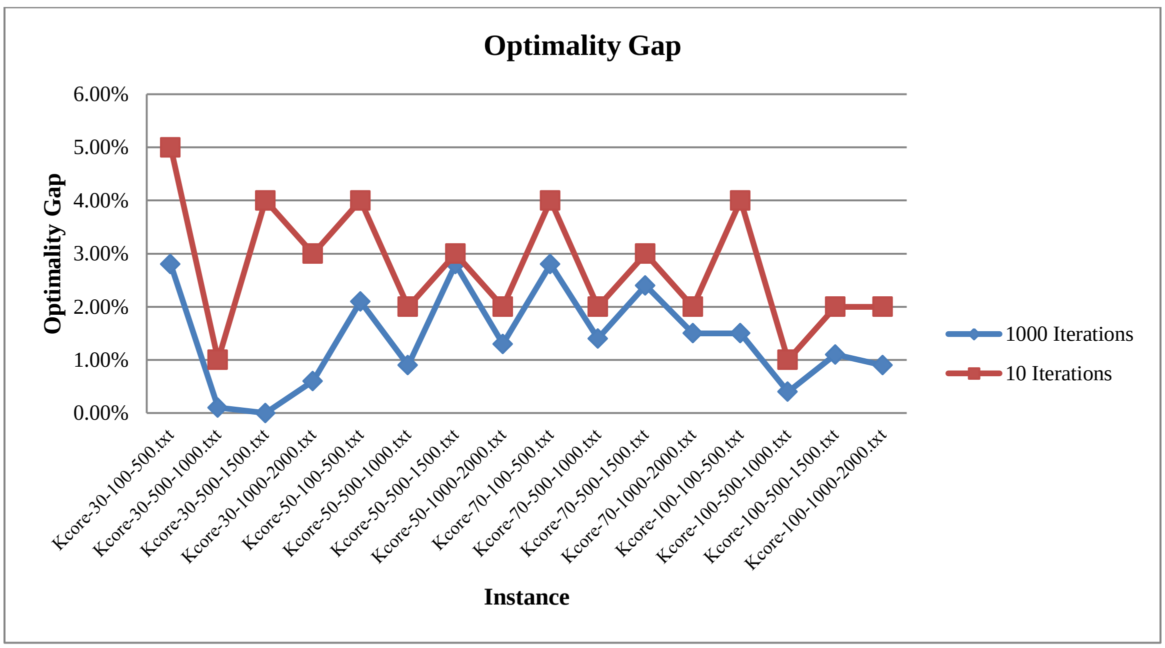


Figure 17. Optimality gap comparison

Appendix B provides statistics mentioned in Section IV.3 for GRASP on the MSkC problem, where Table B.1 presents statistics for 10 iterations for all instances. Table B.2 presents statistics for 1000 iterations for 30, 50, 70 and100 node instance. GRASP is further extended to solve the CCMSkC problem. GRASP's efficiency is first established on the MSkC problem as it allows us to compare optimal solutions from Xpress. For the CCMSkC problem we need to solve a mixed integer non linear program (MINLP), which is not easy given the current commercial optimization packages. So, we evaluate GRASP performance in terms of its running time. The running time comparison of GRASP for MSkC and CCMSkC problems is given in the following section.

IV.5. Numerical Results: GRASP for The CCMSkC Problem

Experimental results of GRASP for CCMSkC problem given in this chapter and Appendix B are based upon the parameters given in Table 9.

Parameter	Value
p_1	90%
p_2	30%
β	60%

Table 9. Parameters of GRASP for the CCMSkC problem

Results of GRASP for the CCMSkC problem at 10 iterations for all the test instances are given in Appendix B, Table B.3, Table B.5. Table B.5 presents the difference in edge sets and objective values of CCMSkC and MSkC problems. Table B.3 provides statistics mentioned in Section IV.3 for GRASP on the CCMSkC problem. To increase possibility of getting good quality solutions experiments were executed with 1000 GRASP iterations for 30, 50, 70 and 100 node instances and detailed in Appendix B, Table B.4. Table 10 presents the difference in solutions returned by GRASP at 1000 and 10 iterations. Difference in solution implies that

GRASP for the CCMSkC problem can return high-quality solutions at higher number of iterations.

Instance	GRASP Solution at 10 Iterations	GRASP Solution at 1000 Iterations	Difference
Kcore-30-100-500.txt	76816	76166	650
Kcore-30-500-1000.txt	212783	210801	1982
Kcore-30-500-1500.txt	271522	269127	2395
Kcore-30-1000-2000.txt	424785	422686	2099
Kcore-50-100-500.txt	196153	195362	791
Kcore-50-500-1000.txt	553641	552678	963
Kcore-50-500-1500.txt	702588	699887	2701
Kcore-50-1000-2000.txt	1109320	1107420	1900
Kcore-70-100-500.txt	366200	365926	274
Kcore-70-500-1000.txt	1055430	1052020	3410
Kcore-70-500-1500.txt	1318970	1317060	1910
Kcore-70-1000-2000.txt	2120300	2111270	9030
Kcore-100-100-500.txt	739676	738665	1011
Kcore-100-500-1000.txt	2123340	2123090	250
Kcore-100-500-1500.txt	2679520	2675020	4500
Kcore-100-1000-2000.txt	4266990	4261880	5110

Table 10. CCMSkC objective improvement at 1000 iterations

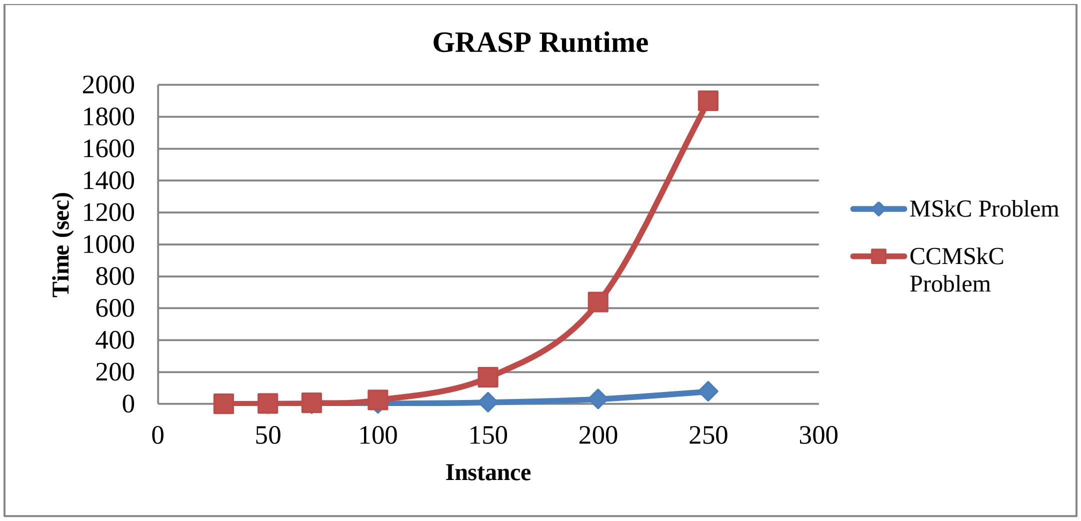


Figure 18. GRASP runtime at 10 iterations

Figure 18 represents running time comparison of GRASP for the MSkC and CCMSkC problems at 10 iterations. Observing performance of GRASP for the CCMSkC problem, we conclude that GRASP for the CCMSkC problem provides good quality solutions in less amount of time.

Recall from Chapter II, chance constraints ensures that the probability of a node's degree being at least k is greater than or equal to a prescribed probability level β, $0 \leq \beta \leq 1$. Higher values of β results in either infeasible solutions or higher cost feasible solutions to the problem. Probability of getting infeasible solutions increases as β increases. The effect of different β values on the CCMSkC problem is shown in Appendix B, Table B.6, where we note the increase in objective values and edge set sizes of instances as β increases.

CHAPTER V

CONCLUSION AND FUTURE WORK

In this thesis we studied the minimum spanning k-core problem introduced by Balasundaram in [4]. k-Cores were originally proposed by Seidman [22] in the social network analysis literature. Minimum spanning k-core problem uses the notion of classical k-cores and explicitly controls minimum degree and by proper choice of minimum degree, implicitly controls diameter and connectivity of the network design.

The main contribution of this thesis is a GRASP metaheuristic to solve the MSkC and CCMSkC problems. GRASP for the MSkC problem was first developed, benchmarked, and then extended for the CCMSkC problem. Developing GRASP for the MSkC problem before extending it to the CCMSkC problem allowed us to assess the performance of GRASP on the MSkC problem by comparing optimal results from Xpress. These results helped to establish that the GRASP for the MSkC problem returns a good feasible solution for all the test instances. Following this we extended GRASP to solve the CCMSkC problem. In this thesis we have used individual chance constraints as we have not found an efficient way to handle dependence and calculate the joint probability. Finding an expression to calculate joint probability and employ the existing model with a joint chance constraint is an interesting topic for future research.

Furthermore, the preliminary experiments on the GRASP for CCMSkC problem leads us to an interesting conclusion that the CCMSkC problem on $G(n,p)$ uniform random graph instances can be reduced to a special case of MSkC problem. Therefore, we proposed a new $G(n,p_1,p_2)$ model for the CCMSkC problem and developed GRASP algorithm for the same. Results shown in Appendix B, Table B.6 suggest that the GRASP for CCMSkC problem provides good quality solutions in reduced amount of time. In the future, it would be interesting to derive an efficient expression that can determine probability of a vertex degree being at least k for $G(n, p_1, p_2, \ldots, p_m)$ graph model, where every edge exists with a possibly different probability. Additionally, a problem for study in the immediate future is to use the Conditional Value at Risk (CVaR) based approach for the probabilistic version of minimum spanning *k*-core problem. CVaR is a downside financial risk measure that has recently been adopted to the network optimization under uncertainty [6, 27].

Finally, considering superior solution qualities of developed GRASP algorithms and above mentioned future extensions, we conclude that the GRASP approach applied to the minimum spanning *k*-cores in this thesis is a successful approach. The complexity of CCMSkC problem is an open question to be addressed.

REFERENCES

1. Abello, J., Pardalos, P., & Resende, M. (1999). On Maximum Clique Problems In Very Large Graphs. In J. Abello, & V. J., *External Memory Algorithms and Visualization* (pp. 110-130). Boston: Americal Mathematical Society.

2. Ahuja, R., Magnanti, T., & Orlin, J. (1993). *Network Flows:Theory, Algorithms, and Applications.* Prentice Hall.

3. Anstee , R. (1987). A Polynomial Algorithm for b-Matchings: An Alternative Approach . *Information Processing Letters , 24* (3), 153-157 .

4. Balasundaram, B. (2007). *Graph Theoretic Generalizations of Clique: Optimization and Extensions.* College Station: Texas A&M University.

5. Blum , C., & Roli. , A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv. , 35* (3), 268-308.

6. Boginski, V. L., Commander, C. W., & Timofey, T. (2009). Polynomial-time identification of robust network flows under uncertain arc failures. *Optimization Letters* , 461-473.

7. Boginski, V., Butenko, S., & Pardalos, P. (2003). On Structural Properties of Market Graph. In A. Nagurney, *innovation in Financial and Economic Networks.* London: Edward Elgar publishers.

8. Broido, A., & Claffy, K. (2001). Internet Topology: Connectivity of IP Graphs. In F. S., & P. K., *Scalability and Traffic Control in IP Networks* (pp. 172-187). Bellingham, WA: SPIE Publications.

9. Cook, W., & Pulleyblank, W. (1987). Linear Systems for Constrained Matching Problems. *Mathematics of Operations Research , 12* (1), 97-120 .

10. Diestel, R. (1997). *Graph Theory.* Berlin : Springer-Verlag.

11. Edmonds, J. (1965). Maximum Matching and A Polyhedron with 0, 1 Vertices. *Journal of Research of the National Bureau of Standards , B 69B*, 125-130 .

12. Erdős, P., & Rényi, A. (1959). On Random Graphs . *Publ. Math. , 6*, 290-297.

13. Erdős, P., & Rényi, A. (1969). On The Evolution of Random Graphs . *Publications of the Mathematical Institute of the Hungarian Academy of Sciences , 5*, 17-61.

14. Erdős, P., & Rényi, A. (1961). On the Strength of Connectedness of a Random Graph . *Acta Mathematica Sscientia , 12*, 261-267.

15. Feo , T., & Resende, M. (1995). Greedy Randomized Adaptive Search Procedures . *Journal of Global Optimization , 6*, 109-133.

16. Festa , P., & Rensende, M. (2000). *GRASP: An annotated bibliography*. Floham Park, NJ: AT&T Labs Research.

17. Festa, P., & Resende, M. C. (2009). An annotated bibliography of GRASP–Part II: Applications. *International Transactions in Operational Research , 16* (2), 131–172.

18. Glover, F. (1997). Tabu Search. In *Metaheuristic procedures for training neural networks* (p. 53). Boston: Springer.

19. Goldberg, D. (1989). Genetic algorithms in search and optimization. *Machine Learning* .

20. Henrion, R. (2004). *Stochastic Programming Community Home Page.* Retrieved from http://stoprog.org.

21. Kirkpatrick, S. (1984). Optimization by simulated annealing: Quantitative studies. *Journal of Statistical Physics* , 975-986.

22. Lap Chi, L., Singh, M., Mohammad, R., & Naor, J. (2007). Survivable Network Design with Degree and Order Constraints. *Proceedings of the Thirty-Ninth Annual ACM Symposium On Theory of Computing.*

23. Minoux, M. (1989). Network synthesis and optimum network design problems: Models, solution methods, and applications. *Networks , 19*, 313-360.

24. Pulleyblank, W. (1973). *Faces of matching polyhedra.* . Canada : University of Waterloo.

25. Reeves, C. (1993). Modern Heuristic Techniques for Combinatorial Problems . *Blackwell Scientific Publishing* .

26. Resende , M., & Ribeiro , C. (2003). Greedy Randomized Adaptive Search Procedures. In G. Kochenberger, & F. Glover, *Handbook of Metaheuristics* (pp. 219–249). Kluwer Academic Publishers.

27. Rockafellar, T. R., & Uryasev, S. (2000). Optimization of Conditional Value-at-Risk. *The journal of risk , 2*.

28. Seidman, S. (1983). Network structure and minimum degree. *Social Networks , 5*, 269-287.

29. Steiglitz, K., Weiner, P., & Kleitman., D. (1969). The Design of Minimum-Cost Survivable Networks. *IEEE Transactions on Circuit Theory , 16* (4), 455–460.

30. Wasserman, S., & Faust, K. (1994). *Social Network Analysis.* New York: Cambridge University Press.

31. West, D. (2001). *Introduction to Graph Theory* . NJ : Prentice-Hall .

APPPENDIX A

MATLAB INSTANCE GENERATORS AND

XPRESS MODEL IMPLEMENTATION

Appendix A presents two MATLAB generators we have used to generate test bed of the instances. Section A.1 is MATLAB generator which generates instance data files for C++ implementation. Format of an instance data file for C++ is shown in Section A.3, which further converted into Xpress readable format (Section A.4) by using MATLAB generator given in Section A.2.Finally, this appendix provide Xpress mosel language code for the MSkC problem.

A.1. MATLAB instance generator for C++ implementation

```
n = 6;
IMIN = 10;
IMAX = 50;

vert = n;
edge = n*(n-1)/2;
P1_percent = 80;
name =
strcat('Kcore','-',num2str(vert),'-',num2str(IMIN),'-
',num2str(IMAX),'.txt')
fid = fopen(name,'w');

     fprintf(fid, 'c ');
     fprintf(fid, name);
     fprintf(fid, '\n');
     fprintf(fid, 'p ');
     fprintf(fid, 'nodes ');
     fprintf(fid, '%d' , vert);
     fprintf(fid, '\n');

     id = 0;
     P1 = 0.9;
     P2 = 0.3;
     p1_edges = floor(edge * P1_percent/100);
     probability_column = zeros(edge,1);
     probability_column(1:p1_edges) = P1;
     probability_column(p1_edges+1:end) = P2;
     probability_column = randsample(probability_column,edge);

     for t = 1:n-1
        for h = t+1:n
            fprintf(fid,'e ');
            fprintf(fid,'%d',id);       %Printing edge id
            fprintf(fid, ' ');
            fprintf(fid,'%d',t-1);      %Printing tail node
            fprintf(fid, ' ');
            fprintf(fid,'%d',h-1);      %Printing head node
            fprintf(fid,' ');
            costs = randi([IMIN IMAX],[1 1]);
            fprintf(fid,'%d',costs);    %Printing cost
            %Printing probability
            fprintf(fid,' %1.2f',probability_column(id+1));
            fprintf(fid,'\n');
            id = id + 1;
        end
     end
    status = fclose(fid);
```

A.2. MATLAB instance generator for Xpress implementation

```
function c2xpress(name)
% Read Instances from C++ data file.
      ipname = strcat(name,'.txt');
      opname = strcat(name,'.dat');
      display(strcat(['Converting ' ipname ' to ' opname '...']));
      fid = fopen(ipname,'r');
      discard = fscanf(fid,'%s',[1 1]);
      while (discard =='c')
          tline = fgets(fid);
          discard = fscanf(fid,'%s',[1 1]);
      end
      discard = fgets(fid,6);
      G = fscanf(fid,'%d',[1 1]);
      N = G(1); % no. of vertices.
      E = (N*(N-1))/2; % no. of edges.
      k = N/2;
      COST = zeros(E,1);
      PROB = zeros(E,1);
      discard = fscanf(fid,'%s',[1 1]);
      while (discard =='e')
          temp =  fscanf(fid,'%d %d',[5]);
          COST(temp(1)+1)=temp(4);
          discard = fscanf(fid,'%s',[1 1]);
          discard = fscanf(fid,'%s',[1 1]);
      end
      status = fclose(fid);

% Write Instances in XPRESS Format.
    fidw = fopen(opname,'w');
    fprintf(fidw,'%s','NODEMAX: ');
    fprintf(fidw,'%d',N-1);
    fprintf(fidw,'\n');
    fprintf(fidw,'\n');
    fprintf(fidw,'%s','k: ');
    fprintf(fidw,'%d',k);
    fprintf(fidw,'\n');
    fprintf(fidw,'\n');
    fprintf(fidw,'%s','ARCS: [');
    fprintf(fidw,'\n');
    id = 0;
      for t = 1:N-1
        for h = t+1:N
            fprintf(fidw,'%s','(');
            fprintf(fidw,'%d',id);        %Printing edge id
            id = id + 1;
            fprintf(fidw,'%s', ' ');
            fprintf(fidw,'%d',1);
            fprintf(fidw,'%s',') ');
            fprintf(fidw,'%d',t-1);       %Printing tail node
            fprintf(fidw, ' ');
            fprintf(fidw,'%d',h-1);       %Printing head node
            fprintf(fidw,'\n');
        end
    end
```

```
    fprintf(fidw,']\n\n');
    if (id~=E)
        display('messed up');
    end
    fprintf(fidw,'%s','COST: [');fprintf(fidw, '\n');
    for i=1:E
            fprintf(fidw,'%s','(');
             fprintf(fidw,'%d',i-1);
             fprintf(fidw,'%s',') ');
             fprintf(fidw,'%d',COST(i));
             fprintf(fidw,'\n');
    end
    fprintf(fidw, ']\n\n');
    if (id~=E)
        display('messed up');
    end
    display('Run Complete.');
    status = fclose(fidw);
clear;
```

A.3. An instance data file generated for C++ implementation

```
c Kcore-6-10-50.txt
p nodes 6
e 0 0 1 15 0.90
e 1 0 2 27 0.90
e 2 0 3 47 0.90
e 3 0 4 42 0.90
e 4 0 5 49 0.30
e 5 1 2 36 0.90
e 6 1 3 11 0.90
e 7 1 4 44 0.30
e 8 1 5 48 0.90
e 9 2 3 37 0.90
e 10 2 4 41 0.90
e 11 2 5 40 0.30
e 12 3 4 26 0.90
e 13 3 5 36 0.90
e 14 4 5 17 0.90
```

A.4. An instance data file generated for Xpress implementation

```
NODEMAX: 5

k: 3

ARCS: [(0 1) 0 1 (1 1) 0 2 (2 1) 0 3 (3 1) 0 4 (4 1) 0 5 (5 1) 1 2 (6
1) 1 3 (7 1) 1 4 (8 1) 1 5 (9 1) 2 3 (10 1) 2 4 (11 1) 2 5 (12 1) 3 4
(13 1) 3 5 (14 1) 4 5]

COST: [(0) 15 (1) 27 (2) 47 (3) 42 (4) 49 (5) 36 (6) 11 (7) 44 (8) 48
(9) 37 (10) 41 (11) 40 (12) 26 (13) 36 (14) 17]
```

A.5. Xpress model for deterministic binary IP formulation

```
model "Minimum Spanning k-core Network"
uses "mmxprs","mmsystem"

!! DATA & PARAMETERS
      parameters
            DATAFILE= "Kcore-6-100-500.dat"
            e = 0
      end-parameters

      declarations
            NODEMAX, k: integer
      end-declarations

      initializations from DATAFILE
            NODEMAX k
      end-initializations

      declarations
            NODES = 0…NODEMAX
            ARCS: array (ARCID: range, 1..2) of integer
            COST: array (ARCID) of integer
            starttime, runtime : real
      end-declarations

      initializations from DATAFILE
            ARCS COST
      end-initializations

      finalize(ARCID)

      declarations
            x: array (ARCID) of mpvar     !1 if Arc is selected, 0
      otherwise
      end-declarations

!!OBJECTIVE FUNCTION
      NetworkCost: = sum (a in ARCID) COST(a)*x(a)

!!CONSTRAINT-1: DESIRABLE VERTEX CONNECTIVITY CHECK
      forall (i in NODES) do
      sum (a in ARCID | ARCS (a,2)=i) x(a) + sum (a in ARCID | ARCS
      (a,1)=i) x(a) >= k
      end-do

!!CONSTRAINT-2: BINARY VARIABLE x
      forall(a in ARCID) x(a) is_binary

!!SOLVING PROBLEM
      starttime:=gettime
      minimize(NetworkCost)
      runtime:=gettime- starttime

!!PROBLEM STATUS
      declarations
```

```
      status:array({XPRS_OPT,XPRS_UNF,XPRS_INF,XPRS_UNB,XPRS_OTH}) of
      string
      end-declarations
      status::([XPRS_OPT,XPRS_UNF,XPRS_INF,XPRS_UNB,XPRS_OTH])[
      "Optimum found","Unfinished","Infeasible","Unbounded","Failed"]

!! PRINTING SOLUTION
      writeln ("SOLUTION:")
      writeln ("Status: ",status(getprobstat))
      writeln ("Running Time (excluding data operations): ",runtime)
      writeln ("Objective Value: ", getobjval)
            forall(a in ARCID)
            if(getsol(x(a)) > 0) then
            e := e+1
            end-if
      writeln ("Number of edges in solution:",e)
end-model
```

APPENDIX B

RESULTS OF GRASP ALGORITHM FOR MSkC AND CCMSkC PROBLEMS

Appendix B present necessary statistics collected during the computational experiments on the deterministic and probabilistic versions of GRASP algorithm. Tables B.1, B.2 are statistics of GRASP for the MSkC problem. However, Tables B.3, B.4 and B.5 are statistics of GRASP for the CCMSkC problem. At the end, Table B.6 illustrates the effect of different β values on three instances of the CCMSkC problem.

Instance	Xpress Solution	Xpress Time (Sec)	GRASP Solution	GRASP Time (Sec)	Construction Time (sec)	Local Search Time (sec)	Optimality Gap	LS Hit Rate	LSAvgPerDec
Kcore-30-100-500.txt	46835	0.032	49064	0.00	0.00	0.00	5%	1	0.8%
Kcore-30-500-1000.txt	142816	0.016	144952	0.02	0.00	0.02	1%	0.9	0.2%
Kcore-30-500-1500.txt	173428	0.016	180292	0.00	0.00	0.00	4%	0.8	0.5%
Kcore-30-1000-2000.txt	284664	0.015	292989	0.02	0.02	0.00	3%	0.9	0.5%
Kcore-50-100-500.txt	129271	0.032	134837	0.09	0.06	0.03	4%	1	0.9%
Kcore-50-500-1000.txt	396438	0.032	405629	0.11	0.09	0.02	2%	0.9	0.1%
Kcore-50-500-1500.txt	484134	0.031	500888	0.11	0.06	0.05	3%	0.9	0.1%
Kcore-50-1000-2000.txt	797718	0.031	811665	0.11	0.08	0.03	2%	0.9	0.2%
Kcore-70-100-500.txt	245924	0.031	255813	0.42	0.34	0.08	4%	1	0.5%
Kcore-70-500-1000.txt	771378	0.078	786296	0.45	0.28	0.17	2%	1	0.1%
Kcore-70-500-1500.txt	928686	0.063	957826	0.45	0.36	0.09	3%	1	0.0%
Kcore-70-1000-2000.txt	1543413	0.047	1576950	0.44	0.36	0.08	2%	1	0.1%
Kcore-100-100-500.txt	503100	0.093	521969	1.70	1.31	0.39	4%	1	0.3%
Kcore-100-500-1000.txt	1562196	0.093	1582140	1.81	1.38	0.44	1%	1	0.0%
Kcore-100-500-1500.txt	1890662	0.094	1931690	1.59	1.19	0.41	2%	1	0.2%
Kcore-100-1000-2000.txt	3144100	0.109	3195850	1.77	1.31	0.45	2%	1	0.1%
Kcore-150-100-500.txt	1146864	0.25	1186470	8.56	7.55	1.02	3%	1	0.5%
Kcore-150-500-1000.txt	3532914	0.204	3584760	9.31	6.53	2.78	1%	0.9	0.1%
Kcore-150-500-1500.txt	4255775	0.203	4350770	9.44	5.72	3.72	2%	1	0.2%
Kcore-150-1000-2000.txt	7048505	0.203	7151640	9.91	6.63	3.28	1%	0.8	0.4%

Table B.1. GRASP results for the MSkC problem at 10 iterations

Instance	Xpress Solution	Xpress Time (Sec)	GRASP Solution	GRASP Time (Sec)	Construction Time (sec)	Local Search Time (sec)	Optimality Gap	LS Hit Rate	LSAvgPerDec
Kcore-200-100-500.txt	2017840	0.359	2081920	28.08	21.99	6.09	3%	1	0.5%
Kcore-200-500-1000.txt	6267888	0.422	6342090	31.84	22.58	9.27	1%	1	0.0%
Kcore-200-500-1500.txt	7502256	0.407	7657740	29.02	27.22	1.80	2%	1	0.1%
Kcore-200-1000-2000.txt	12561163	0.39	12718700	30.80	23.33	7.47	1%	0.8	0.0%
Kcore-250-100-500.txt	3155259	0.547	3243390	73.25	50.16	23.09	3%	1	0.8%
Kcore-250-500-1000.txt	9775016	0.531	9881630	80.20	62.27	17.94	1%	1	0.2%
Kcore-250-500-1500.txt	11786021	0.516	11986000	74.83	60.67	14.16	2%	1	0.5%
Kcore-250-1000-2000.txt	19570171	0.547	19822600	82.78	54.27	28.52	1%	1	0.8%

Table B.1. Continued

Instance	GRASP Solution	GRASP Time (Sec)	Construction Time (sec)	Local Search Time (sec)	Optimality Gap	LS Hit Rate	LSAvgPerDec	Edges Included
Kcore-30-100-500.txt	48152	1.33	0.98	0.35	2.8%	1	0.5%	225
Kcore-30-500-1000.txt	142952	0.63	1.01	0.38	***0.1%***	0.974	0.2%	226
Kcore-30-500-1500.txt	173455	1.44	1.02	0.42	***0.0%***	1	0.2%	226
Kcore-30-1000-2000.txt	286236	0.58	0.99	0.40	***0.6%***	0.895	0.1%	225
Kcore-50-100-500.txt	132037	10.72	8.13	2.59	2.1%	1	0.7%	628
Kcore-50-500-1000.txt	400060	11.38	8.27	3.11	***0.9%***	0.886	0.1%	626
Kcore-50-500-1500.txt	497504	11.09	8.63	2.47	2.8%	0.975	0.2%	627
Kcore-50-1000-2000.txt	807757	10.24	7.85	2.39	1.3%	0.972	0.2%	626
Kcore-70-100-500.txt	252932	40.01	31.22	8.79	2.8%	1	0.7%	1232
Kcore-70-500-1000.txt	782121	43.14	31.57	11.58	1.4%	0.994	0.1%	1227
Kcore-70-500-1500.txt	950526	43.47	30.87	12.60	2.4%	0.951	0.2%	1227
Kcore-70-1000-2000.txt	1566770	42.91	31.11	11.79	1.5%	0.997	0.1%	1226
Kcore-100-100-500.txt	510839	169.16	130.27	38.89	1.5%	1	0.5%	2519
Kcore-100-500-1000.txt	1569120	180.90	128.29	52.62	***0.4%***	0.978	0.1%	2501
Kcore-100-500-1500.txt	1911340	167.81	129.08	38.73	1.1%	1	0.2%	2502
Kcore-100-1000-2000.txt	3171750	172.69	130.49	42.20	***0.9%***	0.999	0.1%	2501

Table B.2. GRASP results for the MSkC problem at 1000 iterations

Instance	GRASP Solution	Construction Time (sec)	Local Search Time (sec)	GRASP Time (sec)	LS Hit Rate	LSAvgPerDec
Kcore-30-100-500.txt	76816	0.08	0.05	0.12	1	1.4%
Kcore-30-500-1000.txt	212783	0.09	0.05	0.14	0.8	1.0%
Kcore-30-500-1500.txt	271522	0.11	0.03	0.14	1	0.8%
Kcore-30-1000-2000.txt	424785	0.06	0.08	0.14	1	0.7%
Kcore-50-100-500.txt	196153	0.66	0.36	1.02	0.9	2.0%
Kcore-50-500-1000.txt	553641	0.69	0.50	1.19	1	1.0%
Kcore-50-500-1500.txt	702588	0.69	0.45	1.14	0.7	2.4%
Kcore-50-1000-2000.txt	1109320	0.70	0.47	1.17	1	0.7%
Kcore-70-100-500.txt	366200	2.83	1.70	4.53	1	2.9%
Kcore-70-500-1000.txt	1055430	2.88	1.86	4.74	0.8	0.7%
Kcore-70-500-1500.txt	1318970	2.84	1.97	4.81	1	1.7%
Kcore-70-1000-2000.txt	2120300	2.92	1.91	4.83	0.9	0.8%
Kcore-100-100-500.txt	739676	14.66	8.75	23.41	1	1.5%
Kcore-100-500-1000.txt	2123340	14.64	9.54	24.19	1	0.6%
Kcore-100-500-1500.txt	2679520	14.91	9.89	24.80	0.9	0.7%
Kcore-100-1000-2000.txt	4266990	14.72	8.91	23.63	1	0.5%
Kcore-150-100-500.txt	1682610	98.98	57.51	156.50	1	1.8%
Kcore-150-500-1000.txt	4821910	101.89	72.91	174.80	1	0.6%
Kcore-150-500-1500.txt	6019560	100.42	65.83	166.25	1	1.6%
Kcore-150-1000-2000.txt	9561310	99.34	69.06	168.40	1	0.5%
Kcore-200-100-500.txt	2977240	400.71	231.79	632.51	0.8	1.0%
Kcore-200-500-1000.txt	8516920	390.81	245.75	636.56	1	0.6%
Kcore-200-500-1500.txt	10578000	393.98	239.31	633.29	1	2.4%
Kcore-200-1000-2000.txt	17041700	393.73	257.50	651.23	1	0.6%
Kcore-250-100-500.txt	4646430	1160.44	621.23	1781.66	1	0.5%
Kcore-250-500-1000.txt	13313000	1183.88	781.87	1965.75	1	1.3%
Kcore-250-500-1500.txt	16616800	1180.52	727.25	1907.77	0.9	0.7%
Kcore-250-1000-2000.txt	26677100	1186.47	767.53	1954.00	1	0.8%

Table B.3. GRASP results for the CCMSkC problem at 10 iterations

Instance	GRASP Solution	Edges Included	Construction Time (sec)	Local Search Time (sec)	GRASP Time (sec)	LS Hit Rate	LSAvgPerDec (%)
Kcore-30-100-500.txt	76166	303	8.00	4.61	12.61	1	1.5%
Kcore-30-500-1000.txt	210801	302	7.84	4.91	12.75	0.9	0.7%
Kcore-30-500-1500.txt	269127	304	8.04	5.45	13.49	1	1.1%
Kcore-30-1000-2000.txt	422686	306	8.35	4.87	13.22	1	0.8%
Kcore-50-100-500.txt	195362	809	65.65	35.52	101.17	1	2.1%
Kcore-50-500-1000.txt	552678	813	67.26	46.05	113.31	1	1.0%
Kcore-50-500-1500.txt	699887	811	65.29	44.39	109.69	0.8	1.5%
Kcore-50-1000-2000.txt	1107420	808	65.61	42.37	107.98	1	0.9%
Kcore-70-100-500.txt	365926	1574	285.96	168.44	454.40	0.9	1.9%
Kcore-70-500-1000.txt	1052020	1570	286.38	193.13	479.51	1	0.8%
Kcore-70-500-1500.txt	1317060	1559	284.91	188.38	473.29	0.9	1.0%
Kcore-70-1000-2000.txt	2111270	1569	289.19	189.76	478.95	1	0.6%
Kcore-100-100-500.txt	738665	3193	1453.87	818.72	2272.59	1	1.3%
Kcore-100-500-1000.txt	2123090	3192	1478.77	994.03	2472.80	1	0.7%
Kcore-100-500-1500.txt	2675020	3206	1499.03	996.76	2495.78	1	1.0%
Kcore-100-1000-2000.txt	4261880	3192	1487.53	948.43	2435.97	1	1.7%

Table B.4. GRASP results for the CCMSkC problem at 1000 iterations

Instance	CCMSkC Edge Set Size	GRASP for CCMSkC Solution Cost	MSkC Edge Set Size	GRASP for MSkC Solution Cost
Kcore-30-100-500.txt	305	76816	226	49064
Kcore-30-500-1000.txt	306	212783	225	144952
Kcore-30-500-1500.txt	308	271522	226	180292
Kcore-30-1000-2000.txt	310	424785	226	292989
Kcore-50-100-500.txt	814	196153	631	134837
Kcore-50-500-1000.txt	813	553641	625	405629
Kcore-50-500-1500.txt	817	702588	628	500888
Kcore-50-1000-2000.txt	810	1109320	627	811665
Kcore-70-100-500.txt	1574	366200	1235	255813
Kcore-70-500-1000.txt	1574	1055430	1232	786296
Kcore-70-500-1500.txt	1568	1318970	1227	957826
Kcore-70-1000-2000.txt	1574	2120300	1228	1576950
Kcore-100-100-500.txt	3194	739676	2521	521969
Kcore-100-500-1000.txt	3193	2123340	2503	1582140
Kcore-100-500-1500.txt	3209	2679520	2508	1931690
Kcore-100-1000-	3199	4266990	2506	3195850
Kcore-150-100-500.txt	7206	1682610	5670	1186470
Kcore-150-500-1000.txt	7231	4821910	5636	3584760
Kcore-150-500-1500.txt	7220	6019560	5635	4350770
Kcore-150-1000-	7190	9561310	5633	7151640
Kcore-200-100-500.txt	12834	2977240	10053	2081920
Kcore-200-500-1000.txt	12816	8516920	10008	6342090
Kcore-200-500-1500.txt	12831	10578000	10016	7657740
Kcore-200-1000-	12816	17041700	10009	12718700
Kcore-250-100-500.txt	20077	4646430	15740	3243390
Kcore-250-500-1000.txt	20083	13313000	15628	9881630
Kcore-250-500-1500.txt	20061	16616800	15640	11986000
Kcore-250-1000-	20070	26677100	15632	19822600

Table B.5. The CCMSkC problem objective functions and edge sets at 10 iterations

Instance \ β	0.0	0.1	0.2	0.3	0.4	0.5
Kcore-30-500-1000.txt	146182	179892	187614	194649	196900	204287
Edges Set	227	266	276	284	286	295
Kcore-50-500-1000.txt	405703	495253	506164	520740	533741	543407
Edges Set	629	746	756	773	789	803
Kcore-70-500-1000.txt	786768	951185	976499	993467	1019260	1037130
Edges Set	1236	1445	1478	1501	1533	1553

Instance \ β	0.6	0.7	0.8	0.9	1
Kcore-30-500-1000.txt	211518	226536	243504	No feasible solution found	No feasible solution found
Edges Set	303	318	336		
Kcore-50-500-1000.txt	555257	567982	590436	623637	No feasible solution found
Edges Set	815	830	859	895	
Kcore-70-500-1000.txt	1054260	1082480	1111040	1152250	No feasible solution found
Edges Set	1571	1606	1637	1687	

Table B.6. Effect of different β values on the CCMSkC problem at 10 iterations

VITA

Ameya Abasaheb Dhaygude

Candidate for the Degree of

Master of Science

Thesis: A HEURISTIC APPROACH TO THE CHANCE CONSTRAINED MINIMUM SPANNING k-CORE PROBLEM

Major Field: Industrial Engineering and Management

Biographical:

Education:

December, 2010
Master of Science in Industrial Engineering and Management
Oklahoma State University, Stillwater, Oklahoma, USA

June, 2006
Bachelor of Science in Mechanical Engineering
University of Mumbai, Mumbai, Maharashtra, India

Experience:

Research Assistant (January, 2009 – December, 2010)
Oklahoma State University, Stillwater, Oklahoma, USA

Project Development Executive (September, 2007 – February, 2008)
StructArch Consultants, Mumbai, Maharashtra, India

Engineer Trainee (March, 2007 – August, 2007)
NMSEZ Pvt. Ltd., Mumbai, Maharashtra, India

Engineer Trainee (November, 2006 – February, 2007)
Hercules Hoists Ltd., Mumbai, Maharashtra, India

Professional Membership: Alpha Pi Mu,
Society for Industrial and Applied Mathematics,
Institute for Operations Research and the
Management Sciences

Name: Ameya Abasaheb Dhaygude Date of Degree: December, 2010

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: A HEURISTIC APPROACH TO THE CHANCE CONSTRAINED MINIMUM SPANNING *k*-CORE PROBLEM

Pages in Study: 61 Candidate for the Degree of Master of Science

Major Field: Industrial Engineering and Management

This thesis presents metaheuristic approaches to solve a novel network design problem under uncertainty. The problem is an extension of the classical *k*-core based network model called as the *minimum spanning k–core problem*. The minimum spanning *k*-core problem aims to balance the network design objectives of robustness, reachability and cost effectiveness. The problem is further extended to a probabilistic version called as, the *chance constrained minimum spanning k-core problem*. The minimum spanning *k*-core problem can be used to design underlying transportation networks, telecommunication networks, electrical and power distribution networks etc. in robust manner.

In this thesis, Greedy Randomized Adaptive Search Procedure (GRASP), a metaheuristic approach is developed to solve both versions of the minimum spanning *k*-core problem. Computational experiments are performed to study the effectiveness of GRASP on specially designed test instances. Computational results conclude that GRASP provides good quality feasible solutions and efficiently solve both versions of the minimum spanning *k*-core problem.

ADVISER'S APPROVAL: Dr. Balabhaskar Balasundaram