PATH SELECTION PROBLEM IN NETWORK DESIGN

by

Xueying Shen

M.Sc, Dalian University of Technology, 2011

B.Sc, Dalian University of Technology, 2009

A Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science

in the
Department of Mathematics
Faculty of Science

SIMON FRASER UNIVERSITY
Spring 2014

APPROVAL

Name: Xueying Shen

Degree: Master of Science

Title of Thesis: Path Selection Problem in Network Design

Examining Committee: Dr. Tamon Stephen
Chair

Dr. Abraham Punnen,
Professor, Supervisor

Dr. Snezana Mitrovic-Minic,
Adjunct Professor, External Examiner

Dr. Binay Bhattacharya,
Professor, Computer Science
Simon Fraser University
Supervisory Committee Member

Date Approved: Apr. 10, 2014

Partial Copyright Licence

Abstract

In this thesis we study several models for the Path Selection Problem associated with the construction of fibre optic networks. Four different variations of the problem are studied: Greedy Path Selection Problem (GPSP), Benevolent Path Selection Problem (BPSP), Discounted Path Selection Problem (DPSP) and Path Selection with capacity expansion (EPSP). All the variations are NP-hard and polynomially solvable special cases are identified for GPSP and BPSP. We also present detailed complexity analysis and integer programming formulations for these problems. Heuristic algorithms including greedy algorithm, semi-greedy algorithm and multi-start local search algorithm are developed for each problem. Extensive computational results are provided with the algorithm performance analysis. We also present some future directions for research.

Acknowledgments

I would like to thank my supervisor Dr. Abraham Punnen, you guide me through my research with tremendous patience and profound knowledge. I cannot express enough how grateful I am for being your student. Without your help and encouragement, I will never be who I am.

My deep gratitude to the thesis committee member Dr. Binay Bhattacharya and Dr. Snezana Mitrovic-Minic, for your precious time and valuable comments on this work. Many thanks to Dr. Zhaosong Lu, Dr. Tamon Stephen, Dr. Randall Pyke and Dr. Natalia Kouzniak for your continuous help.

I would like to express my appreciation to all the graduate fellows: Yong Zhang, Annie Zhang, Mahdieh Malekian, Piyashat Sripratak, Timothy James Yusun, Pooja Pandey and Xiaorui Li. Special thanks to Krishna Teja Malladi, Daniel Karapetyan and Brad Woods.

Finally, I would like to express my sincere gratitude to my parents and friends, you are the meaning of my life.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The primary focus of this thesis is to develop efficient heuristic algorithms for the *Path Selection Problem* (PSP) motivated by fibre optic network design. Let $G = (V, E)$ be a given graph, where the edge set $E = \{e_1, e_2, \ldots, e_m\}$ constitutes the union of n paths $\mathcal{F} = \{P_1, P_2, \ldots, P_n\}$. For each edge $e_i \in E$, a capacity b_i is prescribed. Also, for each edge e_i, a_{ij} is the fraction of the capacity used by path P_j from the capacity b_i of e_i, which is nonzero only when $e_i \in P_j$. Let $f : 2^{\mathcal{F}} \to \mathbb{R}$ be a real valued function that measures the value of a subset of paths. Then PSP is to choose a subset S of paths in $\mathcal{F}$ such that $f(S)$ is maximized while the paths in S obey appropriate capacity restrictions; i.e., $\sum_{P_j \in S} a_{ij} \leq b_i$ for all $e_i \in E$. Depending on the nature of f and/or additional restrictions, we have different variations of PSP. In particular, the variations that we consider are *Greedy Path Selection Problem* (GPSP), *Benevolent Path Selection Problem* (BPSP), *Discounted Path Selection Problem* (DPSP) and *Path Selection with capacity expansion* (EPSP). Formal definitions of these variations will be discussed in later chapters. PSP can be formulated as the following *0-1 integer programming problem*:

$$\begin{aligned} &\text{Maximize} \quad f(S(x)) \\ &\text{Subject to:} \\ &\qquad \sum_{P_j \in \mathcal{F}} a_{ij} x_j \leq b_i \quad \forall e_i \in E \\ &\qquad x_j \in \{0, 1\} \quad \forall P_j \in \mathcal{F} \end{aligned} \tag{1.1}$$

where $S(x) = \{P_j \in \mathcal{F} : x_j \neq 0\}$.

Throughout this thesis, we assume that for path P_j the capacity usage a_{ij} are equal for

all the edges in it and hence

$$a_{ij} = \begin{cases} a_j & \text{if } e_i \in P_j, \\ 0 & \text{otherwise.} \end{cases}$$

Let c_j be the value of path P_j. Then if we choose $f(S) = \sum_{P_j \in S} c_j$ in (1.1), we get an integer programming formulation of the GPSP. More details of the 0-1 integer programming formulations of GPSP will be discussed in Chapter 2. Let us now review some well studied problems closely related to GPSP.

1.1 Multi-dimensional Knapsack Problem

The *Multi-dimensional Knapsack Problem* (MDKP) [61] is the special 0-1 integer programming problem

$$\text{Maximize} \quad \sum_{j \in N} c_j x_j$$

Subject to:

$$\sum_{j \in N} a_{ij} x_j \leq b_i \quad \forall i \in M$$

$$x_j \in \{0, 1\} \quad \forall j \in N$$

Where $a_{ij} \geq 0$. When $a_{ij} = a_j$ for $a_{ij} \neq 0$, we call the resulting MDKP a *Column Restricted Multi-dimensional Knapsack Problem* (CMDKP).

We will show that CMDKP has the same form as the integer programming formulation of GPSP. Thus, GPSP is a special case of MDKP and can be solved using any algorithm to solve MDKP. However, exploiting the special structure of GPSP, more efficient algorithms may be possible and exploring this possibility is part of the discussions in Chapter 2.

MDKP is strongly NP-hard and it is NP-hard to obtain a fully polynomial approximation algorithm for the problem [32, 45]. However, polynomial time ϵ-approximation algorithms are available for this problem[15, 29, 50]. Exact algorithms for MDKP have been studied since 1960s based on dynamic programming, branch and bound or other enumeration techniques [33, 11, 10]. Heuristic algorithms for MDKP have been studied by various authors using different techniques such as greedy type algorithms [56], linear programming relaxation (LP relaxation) based algorithms [39], tabu search heuristics [22], genetic algorithm [20], simulated annealing heuristics [25] and others [34, 35]. For a comprehensive literature review on MDKP we refer to [20, 26, 40, 27].

1.2 Unsplittable Flow Problem

Another problem closely related to PSP (in particular, GPSP) is the *Unsplittable Flow Problem* (UFP) [6]. In UFP, we are given a graph $G = (V, E)$ with a *capacity* b_i defined on each edge $e_i \in E$ and a set of *requests* $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$. Each request R_j represents a combination $\{s_j, t_j, a_j, c_j\}$, where (s_j, t_j) is a pair of nodes in V, a_j is the *flow demand* and c_j is the *profit* realized. A request R_j is routed if we send a flow a_j along a single path in G from s_j to t_j, which brings profit c_j. The objective of UFP is to route as many requests as possible simultaneously so that the total profit is maximized while edge capacity restrictions are satisfied by the flow.

It can be verified that GPSP is essentially the UFP when the underlying graph is a tree. In this case, there is a unique path P_j connecting each node pair (s_j, t_j) and choosing a path to send flow is not an issue for the UFP on a tree. Therefore the objective of UFP becomes selecting a subset of paths such that the total profit is maximized without violating the capacity constraints. This problem is precisely a GPSP. Hence the algorithmic and complexity results available for UFP on a tree graph (UFP-T) are applicable for GPSP. In the following paragraph, we discuss some of the results for UFP-T and its special case UFP on a path graph (UFP-P).

When the underlying path is a single edge where each request contains identical copies of node pair and different profits and flow demands, then UFP-P reduces to the knapsack problem [51]. This implies that the UFP-P (hence the GPSP) is weakly NP-hard. Chrobak [19] proved that UFP-P is in fact strongly NP-hard even for UFP-P with $b_i = b_0$ for $\forall i$, $c_j = c_0$ for $\forall j$.

Note that, for a maximization problem, an algorithm $\mathcal{A}$ is called an ϵ-approximation algorithm if

$$\frac{z^*(\mathcal{P})}{z^{\mathcal{A}}(\mathcal{P})} \leq \epsilon$$

holds for every instance $\mathcal{P}$ of the problem, where $z^*(\mathcal{P})$ and $z^{\mathcal{A}}(\mathcal{P})$ respectively are the optimal objective function value and the objective function value of the solution obtained by algorithm $\mathcal{A}$. Phillips [54] presented the first constant factor (78.51) approximation algorithm for UFP-P with uniform capacity (i.e. $a_j = a_0$ for all j). The approximation ratio was improved by Bar-Noy [5] to 3, and further improved to $2 + \epsilon$ by Calinescu et al. [12]. For UFP-P with no-bottleneck assumption (NBA) where one assumes $\max a_j \leq \min b_i$, the first constant factor approximation algorithm was proposed by Chakrabarti et al. [13]. This

ratio was improved to $2 + \epsilon$ in 2003 [18]. A pseudo polynomial approximation algorithm for UFP-P was given in [3]. Bansal et al. [4] gave the first polynomial time $O(\log n)$-approximation algorithm for UFP-P, and Bonsma et al. [9] proposed the first polynomial time constant factor approximation algorithm for the problem yielding an approximation ratio of $7 + \epsilon$ where $\epsilon > 0$.

UFP-T is a generalization of UFP-P. By using randomized rounding, Chekuri et al. [17] proposed a constant factor approximation algorithm for UFP-T with NBA. Inspired by [4], Chekuri et al. [16] proposed poly-logarithmic approximation algorithms with approximation ratio $\log l$ for UFP-T with uniform profit and approximation ratio $O(l \min\{\log l, \log n\})$ for the weighted case where $l = |V|$.

Compard to a fairly large volume of works on approximation algorithms, study of heuristic algorithms for UFP and experimental analysis of such algorithms are relatively small. Based on the relationship between GPSP and UFP-T, the work presented in this thesis is also a contribution to the progress in solving some versions of UFP by heuristic algorithms.

1.3 Column Restricted Packing Integer Programming

Another problem closely related to GPSP is the *Column Restricted Packing Integer Programming* (CPIP) [43, 41]. Given $A = (a_{ij}) \in [0,1]^{m \times n}$, $b \in [1, \infty)^m$ and $c \in [0,1]^n$, a *Packing Integer Programming* (PIP) is defined as follows:

$$\begin{aligned} &\text{Maximize} && c^T x \\ &\text{Subject to:} && \\ & && Ax \leq b \\ & && x \in Z_+^n \end{aligned} \tag{PIP}$$

If all the non-zero entries in each column j have the same value a_j in PIP, we get CPIP. GPSP is essentially CPIP with binary variables and all involving coefficients are allowed to take any nonnegative value.

CPIP was first studied by Kolliopoulos and Stein [43, 41] when addressing *Disjoint Edge Paths Problem* (DEPP), which is a special case of UFP. A CPIP was constructed in their algorithm for DEPP to transform the fractional single-path routing into a final feasible approximate solution. They developed an approximation algorithm for CPIP. Baveja and Srinivasan [7] also studied CPIP and obtained results similar to that of [41] independently

by using a rounding scheme. A δ-bounded CPIP is a CPIP satisfying $\max_j a_{ij} \leq (1-\delta)b_i$ where $\delta \in (0,1)$. Chekuri et al. [16] proved that the integrality gap of a δ-bounded CPIP is at most $O(\log(1/\delta)/\delta^3)$ times the integral gap of its unit-demand version by using the method proposed in [41], where the unit-demand version is the CPIP with all right hand side values are 1.

As the duality relation between set packing problem and set covering problem, researchers also studied the corresponding Column Restricted Covering Integer Programming (CCIP) [57, 42, 5, 59, 44, 14].

1.4 Set Packing Problem

So far we observed that GPSP is a special case of MKDP and CPIP but overlaps with UFP. Also we point out that GPSP is a generalization of another well studied problem, the *Set Packing Problem* (SPP). Given a finite set $\mathcal{U}$ and a family of subsets F of $\mathcal{U}$, a weight is assigned to each element in F. The SPP is to find a subset $C \subseteq F$ with maximum total weights such that all the elements of C are pairwise disjoint, i.e., the intersection of any elements of C is empty. SPP can be formulated as an integer programming problem:

$$\text{Maximize} \quad \sum_{j \in N} c_j x_j$$

Subject to:

$$\sum_{j \in N_i} x_j \leq 1 \quad \forall i \in M$$

$$x_j \in \{0,1\} \quad \forall j \in N$$

If $a_{ij} \in \{0,1\}$ and $b_i = 1$, the GPSP reduces to SPP. SPP is one of the first studied NP-hard problems [31]. There is no constant factor polynomial time approximation algorithm available for SPP. The best known polynomial time approximation algorithm has a performance ratio $O(\sqrt{n})$ [48, 37]. A branch and cut algorithm proposed by Padberg [53] is one of the best performing exact algorithm for the problem. Alidaee et al. [2] formulated a SPP as an unconstrained binary quadratic programming. For other exact algorithms for SPP we refer to [62, 21, 46].

Although there are a considerable amount of works devoted to exact algorithm for SPP, we found relatively few heuristics for the problem. Lou and Goh [47] modeled the brokering problem as an SPP and an iterative greedy approximation algorithm was proposed to solve

it. *Greedy randomized adaptive search procedure* (GRASP) was applied to SPP by Delorme et al. [24]. The same researchers [30] also developed an *ant colony optimization* procedure to solve SPP. This algorithm was then used to solve a railway infrastructure management problem. Guo et al. [36] studied a bidding problem which was formulated as an SPP and solved it using simulating annealing algorithm. Merel et al. [52] proposed a hybrid algorithm combining Column Generation and ant colony optimization.

1.5 Practical motivation for Path Selection Problem

GPSP being generalization and/or specialization of some well studied optimization problem, its theoretical importance is explicit. However, our study of GPSP and other variations of PSP are primarily motivated by a practical application. The PSP appears in the fibre optic network design. We present below some details taken from [55] to illustrate practical implications of the model studied in this thesis.

a2b fibre Inc. [55] provides fibre optic network planning, design, construction and management. Revenue for the company includes both revenue generated by construction and expansion of the network and operation fee paid by the clients. Hence, the goal of the company is to build new fibre optic network connections for clients and enlarge the network it manages. To maximize revenue, the company introduced a strategy that clients who share the network segments share the construction cost to some extent, and it is called Participatory fibre Optic Network PFON^{TM} (Trade mark of a2b fibre Inc.) model.

Typically, a client is interested in building a fibre optic link between an origin-destination pair (O-D pair) and the company wishes to provide an optimal quote for the work. It is important for the company to provide an optimal quote, otherwise the client may not give the order or may choose a competitor. An O-D pair connection can be viewed as a path linking the origin and destination node in a network while passing through various specified intermediate nodes. The portion of path between two consecutive nodes is called an edge or a segment. For each segment, two crucial parameters are identified which are capacity and cost. Due to different structural properties of these segments, the capacities and costs varies. For example, they may need to be constructed afresh or there may exist unused water pipes or other structures that could sometimes be used as conduits. The company actively manages all such information in a database. By using the available information, a database manager will determine the "best path" to connect the O-D pair. Hence given

an O-D pair, choosing a path connecting them is not an issue for the company. Also the cost and the capacity for each edge in the path are known. A path belonging to a client connecting an O-D pair may share some edges with the other clients who (partially) own it. This makes the capacity restriction an important consideration. The construction cost of a path is the summation of its edge costs, which is called the *nominal cost*. The quote for a path is based on its nominal cost but not necessarily equal to it. Path selection becomes an issue when multiple paths are to be selected in present quote.

We consider Path Selection Problem from different perspectives. For example, the simplest scenario is that setting the quote for each path as its nominal cost. Then the problem becomes selecting clients so that the revenue is maximized and all the capacity constraints on edges are satisfied. This is the greedy path selection model. Other variations include discount based models, capacity expansion, or shared costs. Details of these will be discussed in the chapters to follow.

Chapter 2

Greedy PSP

Let $G = (V, E)$ be a graph, where the edge set $E = \{e_1, e_2, \ldots, e_m\}$ is the union of n given paths $P_1, P_2, \ldots, P_n$ in G. Note that two or more paths may share some common edges. For each edge $e_i \in E$, a *cost* w_i and a *capacity* b_i are prescribed. Also a_{ij} is the fraction of the capacity b_i of e_i used by path P_j, which is only nonzero when $e_i \in P_j$. For each path P_j, the nominal path cost $c_j = \sum_{e_i \in P_j} w_i$. Then the *Greedy Path Selection Problem* (GPSP) is to choose a subset of paths S such that $\sum_{P_j \in S} c_j$ is maximized while the paths in S obey appropriate capacity restrictions; i.e., $\sum_{P_j \in S} a_{ij} \leq b_i$ for all $e_i \in E$.

Let the *capacity of a path* P_j be a_j. Thus we assume that $a_{ij} = a_j$ for each $e_i \in P_j$. This assumption is consistent with the real life problem that initiated this study. Without loss of generality, we assume

$$a_j \leq b_i \quad \text{if } e_i \in P_j$$

Otherwise, the path P_j is redundant and can never be selected. Also, we assume that all the associated data are nonnegative which is consistent with the underlying application of the model.

According to the fibre optic network design problem, we call it Greedy Path Selection Problem due to the fact that the nominal cost of a path is charged to each client even though some of its segments (edges) may be shared by more than one client.

2.1 Integer programming formulation

Given a graph $G = (V, E)$ with the edge set $E = \{e_1, e_2, \ldots, e_m\}$ and n given paths $P_1, P_2, \ldots, P_n$ in G, we introduce following notations which are used throughout this thesis:

- $M = \{1, 2, \ldots, m\}$
- $N = \{1, 2, \ldots, n\}$
- $N_i = \{j : e_i \in P_j\}$ for $i \in M$
- $M_j = \{i : e_i \in P_j\}$ for $j \in N$
- $L = \sum_{j \in N} |M_j|$
- $\mathbb{E} = \mathbb{B}^{|N_1|} \times \mathbb{B}^{|N_2|} \times \ldots \times \mathbb{B}^{|N_m|}$

We now formulate GPSP as an integer programming problem. For each $j \in N$, define

$$x_j = \begin{cases} 1 & \text{if path } P_j \text{ is selected,} \\ 0 & \text{otherwise.} \end{cases} \tag{2.1}$$

Then it follows that the capacity restriction for each edge can be formulated as

$$\sum_{j \in N_i} a_j x_j \leq b_i \quad i \in M \tag{2.2}$$

We call constraints (2.2) as *edge capacity constraints*. Therefore, GPSP can be formulated as CMDKP with n variables and m constraints:

$$\begin{aligned} \text{GPSP1:} \quad \text{Maximize} \qquad & f_1(x) = \sum_{j \in N} c_j x_j \\ \text{Subject to:} \qquad & \\ & \sum_{j \in N_i} a_j x_j \leq b_i \quad i \in M \\ & x_j \in \{0, 1\} \quad \forall j \in N \end{aligned}$$

where

$$c_j = \sum_{i \in M_j} w_i. \tag{2.3}$$

Given a matrix A, let a_{ij} represent its (i,j)th entry. Consider the $m \times n$ matrix A defined as

$$a_{ij} = \begin{cases} a_j & \text{if } i \in M_j \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

Also let $b = (b_1, b_2, \ldots, b_m)^T$, $c^T = (c_1, c_2, \ldots, c_n)$, $x^T = (x_1, x_2, \ldots, x_n)$. Then the matrix form of the above CMDKP formulation can be written as

$$\begin{aligned} &\text{Maximize} && c^T x \\ &\text{Subject to:} && \\ & && Ax \leq b \\ & && x \in \{0,1\}^n \end{aligned} \tag{2.5}$$

and A is called the coefficient matrix. The formulation GPSP1 allows us to solve a GPSP of reasonable size using general purpose integer programming solvers such as CPLEX.

In the previous model, the decision variables represent paths. Let us now modify the model using variables representing edges. This formulation is useful in applications where additional constraints are added in relation to edges.

For $i \in M$, $j \in N_i$, define

$$y_{ij} = \begin{cases} 1 & \text{if edge } e_i \text{ is selected in path } j \\ 0 & \text{otherwise.} \end{cases} \tag{2.6}$$

If a path P_j is selected then all its edges should be selected as well, i.e., $y_{ij} = 1$ for $i \in M_j$. This can be guaranteed by the following constraints

$$\sum_{i \in M_j} y_{ij} = |M_j| x_j \quad \forall j \in N \tag{2.7}$$

We call constraints (2.7) the *path selection constraints*. Moreover, we can even relax the integrality restriction on y_{ij} into

$$0 \leq y_{ij} \leq 1 \tag{2.8}$$

This is because when $x_j = 0$, $y_{ij} = 0$ for $i \in M_j$ due to nonnegativity of y_{ij} and constraints (2.7). When $x_j = 1$, there are $y_{ij} = 1$, $j \in M_j$ due to the cardinality of the left side of (2.7)

is exactly $|M_j|$. As for edge capacity constraints, an edge e_i is used by path P_j if and only if $y_{ij} = 1$, and the corresponding capacity usage will be a_j. Hence we have

$$\sum_{j \in N_i} a_j y_{ij} \leq b_i \quad \forall i \in M \tag{2.9}$$

Thus we have another formulation of GPSP following as a *mixed integer programming problem*:

$$\text{GPSP2:} \quad \text{Maximize} \qquad f_2(y) = \sum_{j \in N} \sum_{i \in M_j} w_i y_{ij} \tag{2.10}$$

Subject to:

$$\sum_{j \in N_i} a_j y_{ij} \leq b_i \quad \forall i \in M \tag{2.11}$$

$$\sum_{i \in M_j} y_{ij} = |M_j| x_j \quad \forall j \in N \tag{2.12}$$

$$x_j \in \{0, 1\} \quad \forall j \in N \tag{2.13}$$

$$0 \leq y_{ij} \leq 1 \quad \forall i \in M, j \in N_i \tag{2.14}$$

The formulation GPSP2 is a variation of the formulation given in [55]. Note that L denotes the number of y_{ij} variables and $n \leq L \leq mn$. Thus GPSP2 has $n + L$ variables and $n + m$ constraints without considering the bound restrictions on variables.

Note that the path selection constraints are equivalent to

$$y_{ij} = x_j \quad j \in N, i \in M_j \tag{2.15}$$

Substituting (2.15) for (2.12) in GPSP2 leads to our next mixed integer programming formulation:

$$\text{GPSP3:} \quad \text{Maximize} \qquad f_2(y) = \sum_{j \in N} \sum_{i \in M_j} w_i y_{ij} \tag{2.16}$$

Subject to:

$$\sum_{j \in N_i} a_j y_{ij} \leq b_i \quad \forall i \in M \tag{2.17}$$

$$y_{ij} = x_j \quad j \in N, i \in M_j \tag{2.18}$$

$$x_j \in \{0, 1\} \quad \forall j \in N \tag{2.19}$$

$$0 \leq y_{ij} \leq 1 \quad \forall j \in N, i \in M_j \tag{2.20}$$

which has $n + L$ variables and $m + L$ constraints.

We introduce notation $\langle \cdot, \cdot \rangle$ to represent a pair of elements which may have different dimensions. Let x^0 be an optimal solution to the LP relaxation of GPSP1, $\langle x^*, y^* \rangle$ be an optimal solution to the LP relaxation of GPSP2 where $x \in \mathbb{B}^n$ and $y \in \mathbb{E}$, $\langle \bar{x} | \bar{y} \rangle$ be an optimal solution to the LP relaxation of GPSP3 where $\bar{x} \in \mathbb{B}^n$ and $\bar{y} \in \mathbb{E}$.

Theorem 2.1.1. $f_1(x^0) = f_2(\bar{y}) \leq f_2(y^*)$.

Proof. First we will show that the equality $f_1(x^0) = f_2(\bar{y})$ holds. Given x^0, define $y^0 = (y^0_{ij})$ as

$$y^0_{ij} = \begin{cases} x^0_j & i \in M_j \\ 0 & \text{otherwise.} \end{cases} \qquad j \in N$$

Then $\langle x^0, y^0 \rangle$ is a feasible solution to the LP relaxation of GPSP3. First path selection constraints are satisfied by construction of y^0 and edge capacity constraints are satisfied due to feasibility of x^0 to the LP relaxation of GPSP1. Moreover,

$$f_2(y^0) = \sum_{j \in N} \sum_{i \in M_j} w_i y^0_{ij} = \sum_{j \in N} \sum_{i \in M_j} w_i x^0_j = \sum_{j \in N} c_j x^0_j = f_1(x^0).$$

Thus x^0 and $\langle x^0, y^0 \rangle$ share the same objective function value. Since $f_2(\bar{y})$ is optimal for the LP relaxation of GPSP3, and $\langle x^0, y^0 \rangle$ is a feasible solution to the LP relaxation of GPSP3, we have

$$f_1(x^0) = f_2(y^0) \leq f_2(\bar{y}) \tag{2.21}$$

Similarly, given $\langle \bar{x}, \bar{y} \rangle$, $\bar{x}$ is a feasible solution to GPSP1 and

$$f_2(\bar{y}) = f_1(\bar{x}) \leq f_1(x^0) \tag{2.22}$$

The equality $f_1(x^0) = f_2(\bar{y})$ follows from (2.21) and (2.22).

Next, we will show $f_1(x^0) \leq f_2(y^*)$. $\langle x^0, y^0 \rangle$ is feasible for LP relaxation of GPSP2 since

$$\sum_{i \in M_j} y^0_{ij} = \sum_{i \in M_j} x^0_j = |M_j| x^0_j \quad j \in N$$

$$\sum_{j \in N_i} a_j y^0_{ij} = \sum_{j \in N_i} a_j x^0_j \leq b_i \quad i \in M$$

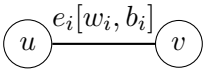
Also it satisfies

$$f_2(y^*) \geq f_2(y^0) = \sum_{j \in N} \sum_{i \in M_j} w_i y^0_{ij} = \sum_{j \in N} (\sum_{i \in M_j} w_i) x^0_j = \sum_{j \in N} c_j x^0_j = f_1(x^0)$$

Thus the result follows. □

The inequality in Theorem 2.1.1 could satisfy as a strict inequality as illustrated in the following example.

Example 2.1.1. *A GPSP instance with* $m = 3$ *and* $n = 3$ *is shown in the graph below:*



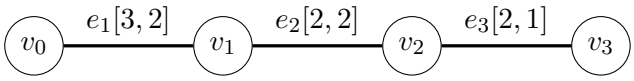


Figure 2.1: Example of strict inequality in Theorem 2.1.1

There are three paths $P_1 = \{e_2, e_3\}$, $P_2 = \{e_1, e_2, e_3\}$, $P_3 = \{e_1, e_2\}$ *with path capacity usage as* $a_1 = 1$, $a_2 = 1$, $a_3 = 2$ *respectively. Then this problem can be formulated as GPSP1:*

$$\begin{aligned} &\textit{Maximize} && 4x_1 + 7x_2 + 5x_3 \\ &\textit{Subject to:} && \\ & && x_2 + 2x_3 \leq 2 \quad (e_1) \\ & && x_1 + x_2 + 2x_3 \leq 2 \quad (e_2) \\ & && x_1 + x_2 \leq 1 \quad (e_3) \\ & && x_1, x_2, x_3 \in \{0, 1\} \end{aligned}$$

GPSP2:

$$\begin{aligned}
&\textit{Maximize} && 2y_{12} + 2y_{13} + 2y_{21} + 2y_{22} + 2y_{23} + y_{31} + y_{32} \\
&\textit{Subject to:} \\
& && y_{21} + y_{31} = 2x_1 \\
& && y_{12} + y_{22} + y_{32} = 3x_2 \\
& && y_{13} + y_{23} = 2x_3 \\
& && y_{12} + 2y_{13} \leq 2 \\
& && y_{21} + y_{22} + 2y_{13} \leq 2 \\
& && y_{31} + y_{32} \leq 1 \\
& && x_1, x_2, x_3 \in \{0, 1\} \\
& && 0 \leq y_{ij} \leq 1 \quad \forall j \in N = \{1, 2, 3\}, i \in M_j
\end{aligned}$$

and GPSP3:

$$\begin{aligned}
&\textit{Maximize} && 2y_{12} + 2y_{13} + 2y_{21} + 2y_{22} + 2y_{23} + y_{31} + y_{32} \\
&\textit{Subject to:} \\
& && y_{12} + 2y_{13} \leq 2 \\
& && y_{21} + y_{22} + 2y_{23} \leq 2 \\
& && y_{31} + y_{32} \leq 1 \\
& && y_{21} = x_1 \quad y_{31} = x_1 \\
& && y_{12} = x_2 \quad y_{22} = x_2 \quad y_{32} = x_2 \\
& && y_{13} = x_3 \quad y_{23} = x_3 \\
& && x_1, x_2, x_3 \in \{0, 1\} \\
& && 0 \leq y_{ij} \leq 1 \quad \forall j \in N, i \in M_j
\end{aligned}$$

The optimal objective function value is 7 for all of these formulations. However, the optimal objective function values of the LP relaxations of GPSP1, GPSP2 and GPSP3 are 9.5, 10.5 and 9.5 respectively. The optimal solutions for each of these LP relaxations are given below:

Solution for LP relaxation of GPSP1

$$x = (0, 1, 0.5)$$

Solution for LP relaxation of GPSP2

$$x = (1, 0.67, 0.5)$$
$$y = \begin{pmatrix} 0 & 1 & 0.5 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

Solution for LP relaxation of GPSP3

$$x = (0, 1, 0.5)$$
$$y = \begin{pmatrix} 0 & 1 & 0.5 \\ 0 & 1 & 0.5 \\ 0 & 1 & 0 \end{pmatrix}$$

In summary, GPSP1 is the most compact formulation and gives equally good upper bound as that of the LP relaxation of GPSP3. Both upper bounds given by these LP relations are tighter than that of GPSP2. The models GPSP2 and GPSP3 include variables corresponding to edges, which may give us flexibility to develop variants of GPSP with additional restrictions on edges.

2.2 Complexity analysis

In the last section, we have shown that GPSP can be formulated as CMDKP along with several other formulations. Interestingly, any CMDKP can be formulated as a GPSP as well.

Theorem 2.2.1. *GPSP and CMDKP are equivalent in the sense that they can be reduced to each other in polynomial time.*

Proof. In the previous section, we have formulated GPSP into GPSP1, which is a CMDKP. Now let us show that any CMDKP can be reduced to a GPSP.

A CMDKP with n variables and m constraints can be defined as (2.23),

$$\begin{aligned} &\text{Maximize} && \sum_{j \in N} c_j x_j \\ &\text{Subject to:} \\ & && \sum_{j \in N} a_{ij} x_j \leq b_i \quad i \in M \\ & && x_j \in \{0,1\} \quad \forall j \in N \end{aligned} \tag{2.23}$$

where

$$a_{ij} = \begin{cases} a_j & i \in M_j \subseteq M \\ 0 & \text{otherwise.} \end{cases}$$

We construct a graph G' for GPSP as follows: Let $V^1 = \{v_i : i = 0, 1, \ldots, m\}$, and $E^1 = \{e_i^1 = (v_{i-1}, v_i) : i = 1, 2, \ldots, m\}$. Assign the capacity and cost of e_i^1 as b_i and 0 respectively. Corresponding to each variable x_j, consider the ordered set $S_j = \{e_i^1 : a_{ij} \neq 0\}$ where the ordering of the elements in S_j are such that e_i^1 appears before e_k^1 if and only if $i < k$. Note that S_j contains at least one edge (assume w.l.o.g. the coefficient matrix of CMDKP has no columns of zeros), and is a collection of paths in G'. Two paths in S_j can be joined to form a single path by introducing a new node and connecting one of the end points of each path to this node by an edge with capacity ∞ and cost 0. Finally, Repeat this process while we get a single path containing all edges of S_j. At last, for each set S_j we append an edge with capacity ∞ and cost c_j at the end to form a path P_j. Let $G = (V, E)$ be the graph obtained by taking the union of all P_j, $j = 1, 2, \ldots, n$. Now we show that GPSP on G is equivalent to the CMDKP under consideration.

We will prove that any feasible solution of one problem can be mapped to that of the other problem, and they share the same objective function value. In the GPSP instance constructed above, selecting path P_j is equivalent to setting $x_j = 1$ in the CMDKP. A one-to-one map can be defined between their solutions $x \in \{0,1\}^n$ and $S \subseteq N$ as

$$x_j = 1 \text{ if and only if } j \in S, \forall j \in N$$

If x is feasible for the CMDKP, constraint $\sum_{j \in N} a_{ij} x_j \leq b_i$ is satisfied for each $i \in M$. Then the capacity restriction for each edge $e_i^1 \in E^1$ is satisfied for GPSP. Note that there are no capacity limit on the other edges, hence the corresponding S is feasible for the GPSP. If S is feasible for the GPSP, due to the capacity restrictions on edge $e_i^1 \in E^1$, x is also feasible

for the CMDKP. Moreover, since the value of path P_j equals to c_j, the objective function value of x for GPSP equals to that of S for CMDKP. The result follows.

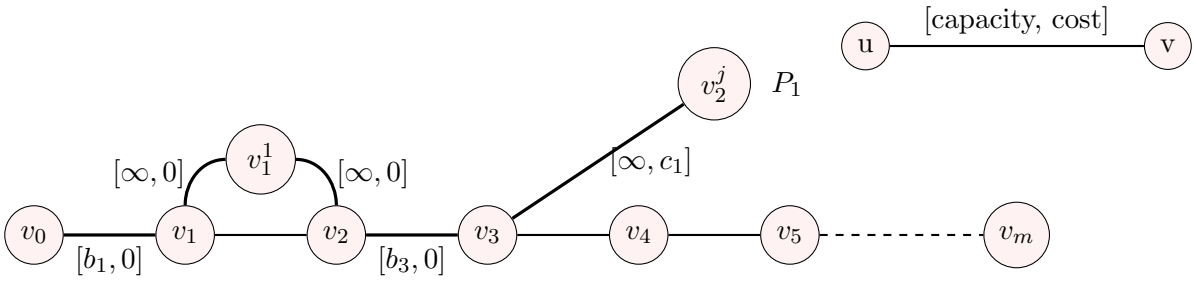


Figure 2.2: Construction GPSP from CMDKP

□

The reduction discussed in the proof Theorem 2.2.1 preserves objective function value and hence it is approximation preserving. Thus we have

Corollary 2.2.1. *CMDKP can be solved by a polynomial time ϵ-approximation algorithm if and only if GPSP can be solved by a polynomial time ϵ-approximation algorithm.*

As a consequence of Corollary 2.2.1, non-approximability results for CMDKP yields non-approximability results for GPSP. For example, consider the *Maximum Clique Problem* (MCP) on a graph $G = (V, E)$. This can be formulated as the 0-1 integer programming problem:

$$\begin{aligned} &\text{Maximize} && \sum_{j=1}^{|V|} x_j \\ &\text{Subject to:} && \\ & && x_i + x_j \leq 1 \quad (i,j) \notin E \\ & && x_j \in \{0,1\} \quad \forall j \in N \end{aligned} \tag{2.24}$$

This is a CMDKP where $b_i = 1$, $a_j = 1$ and $c_j = 1$. Since MCP cannot be approximated within a factor of $|V|^{1-\epsilon}$ for any $\epsilon > 0$ in polynomial time [38], from Corollary 2.2.1, we have

Corollary 2.2.2. *GPSP is NP-hard and cannot be approximated within a factor of $n^{1-\epsilon}$ for any $\epsilon > 0$ in polynomial time even if $a_j \in \{0,1\}$, $c_j \in \{0,1\}$, $b_i = 1$, and each edge is contained in at most two paths.*

We now observe that GPSP remains NP-hard even if the underlying graph has a simple structure.

Theorem 2.2.2. *[55]GPSP on a star graph (GPSP-S) is NP-hard.*

Proof. We reduce the partition problem to GPSP-S. Given n numbers $a_1, a_2, \ldots, a_n$, the partition problem is to determine if there exists a partition S_1, S_2 of $\{1, 2, \ldots, n\}$ such that $\sum_{k \in S_1} a_k = \sum_{k \in S_2} a_k$. Using this instance of the partition problem we construct an instance of GPSP-S as follows. Construct a star with center node 0 and leaf nodes $1, 2, \ldots, n+1$. Let P_i be the path $< n+1, 0, i >$ for $i = 1, 2, \ldots, n$ with capacity usage a_i. Let edge cost $w_{n+1,0} = 0$ and $w_{0j} = a_j$ for $j = 1, 2, \ldots, n$ and set edge capacity $b_{n+1,0} = \frac{1}{2}\sum_{k=1}^{n} a_k$ and $b_{0j} = a_j$ for $j = 1, 2, \ldots, n$. It is easy to verify that the required partition of $\{1, 2, \ldots, n\}$ exists precisely when the optimal objective function value of the constructed instance of GPSP is $\frac{1}{2}\sum_{k=1}^{n} a_k$. The proof now follows from the NP-completeness of the partition problem. □

Theorem 2.2.3. *GPSP on a path graph (GPSP-P) is NP-hard.*

Proof. The NP-hardness of GPSP-P is proved by reduction from knapsack problem. Given item set $N = \{1, 2, \ldots, n\}$ and maximum weight W. For each item $j \in N$ a profit p_j and weight a_j are specified. The knapsack problem is to determine a subset of items S so that the total profit is maximized while the total weight is within weight limit. We reorder the items so that

$$p_1 \leq p_2 \leq \ldots \leq p_n$$

We construct a path graph G for GPSP as follows: Let $V = \{v_i : i = 0, 1, \ldots, n\}$, and $E = \{e_i = (v_{i-1}, v_i) : i = 1, 2, \ldots, n\}$. We assign W and p_1 as the capacity and cost of e_1 respectively, and assign the capacity and cost to be ∞ and $(p_i - p_{i-1})$ respectively for e_i, $i > 1$. The path P_j is defined as $(v_0, v_1, \ldots, v_j)$ with path capacity usage a_j for $j \in N$.

From the construction, the path P_j has cost p_j and path capacity w_j, which corresponds to the item j in knapsack problem. A path selection S is feasible if and only if the capacity restriction on edge e_1 is satisfied, which implies $\sum_{j \in S} w_j \leq W$. Hence a feasible path

selection in above constructed GPSP corresponds to a feasible item selection in underlying knapsack problem. Moreover, they share the same objective function value. The result follows form the NP-hardness of the knapsack problem. □

2.3 Polynomially and pseudo polynomially solvable cases

Let us now identify some polynomially solvable cases. We say GPSP is *separable* if N can be divided into several disjoint subsets $\{N^t, t \in I\}$ such that for any $t \neq s \in I$, $P_i \cap P_j = \emptyset$ where $i \in N^t, j \in N^s$.

Theorem 2.3.1. *If GPSP is separable, also* $\max_{t\in I} |N^t| = O(\log m)$*, then it can be solved in polynomial time.*

Proof. First, we point out a fact that GPSP can be solved in $O(2^n mn)$ time by enumeration. Since GPSP is separable, the problem can be divided into $|I|$ subproblems. Then any subproblem with paths set $N^t, t \in I$ can be solved within $O(2^{|N_t|}m|N_t|) = O(2^{\log m} m \log m) = O(m^2 \log m)$ time, the whole problem will be solved in $O(m^2 n \log m)$ in the worst case. □

Another simple case is when GPSP is restricted on a path graph and all path capacities are identical.

Theorem 2.3.2. *GPSP-P with* $a_j = a_0 > 0$ *for* $j \in N$ *is solvable in polynomial time.*

Proof. Consider the integer programming formulation GPSP1. First, when $a_j = a_0$ for all $j \in N$, we can always transform the coefficient matrix A into a 0-1 matrix by dividing both sides of the constraints $Ax \leq b$ by a_0. We obtain the constraints $A'x \leq \frac{1}{a_0}b$ where $A' = \frac{1}{a_0}A$, $x \in \{0,1\}^n$. Since the left hand side of A' is integer and $x \in \{0,1\}^n$, $A'x \leq \frac{1}{a_0}b$, $x \in \{0,1\}^n$ are equivalent to $A'x \leq \lfloor \frac{b}{a_0} \rfloor$, $x \in \{0,1\}^n$.

When G is a path, all the edges in a path P_j are consecutive, which together with the fact that A' is a 0-1 matrix, the 1's in each column of A' are consecutive. Hence A' is totally unimodular and hence

$$\begin{aligned} &\text{Maximize} && cx \\ &\text{Subject to:} && \\ & && A'x \leq \left\lfloor \frac{b}{a_0} \right\rfloor \\ & && x \in \{0,1\}^n \end{aligned}$$

is polynomially solvable. □

From Theorem 2.3.2 the following corollary can be derived.

Corollary 2.3.1. *If G is a collection of node disjoint paths with $a_j = a_0 > 0$ for $j \in N$, then GPSP is polynomially resolvable.*

Theorem 2.3.3. *GPSP-P can be formulated as a Generalized Network Flow Problem(GNFP).*

Proof. Consider the formulation GPSP1 of GPSP-P. Let A be the coefficient matrix of the resulting formulation. Then as we pointed in the proof of Theorem 2.3.2, all the nonzero entries in the column of A are consecutive such as

$$A = \begin{pmatrix} a_1 & 0 & 0 & \dots \\ a_1 & a_2 & 0 & \dots \\ a_1 & a_2 & a_3 & \dots \\ 0 & 0 & a_3 & \dots \\ & \dots & & \dots \end{pmatrix} \tag{2.25}$$

Insert a n dimension of zero row vector in A right after the last row and set the corresponding right hand side b_{m+1} to be 0 as well, the problem is not affected. Then subtract $(i-1)$th row from ith row for $i = n$ to 2 iteratively. Due to the consecutiveness of nonzero entries, we obtain a matrix with each column has exactly two nonzero entries of the same absolute value but with opposite sign. Again, row operation does not change original problem. An integer programming with such a coefficient matrix can always be seen as a GNFP (maybe with multiple edges). □

The proof given above is a modification of the well known idea of transforming matrices with consecutive ones property into node-arc incidence matrix of a graph [1]. We give below an example to illustrate the transformed GNFP discussed above.

Example 2.3.1. *Consider GPSP*

$$\textit{Maximize} \quad 2x_1 + 3x_2 + x_3 + 2x_4 + 5x_5 + x_6 + 3x_7 + 3x_8 + 5x_9 + x_{10}$$

Subject to:

$$\begin{pmatrix} 1 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 2 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 1 & 7 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 4 & 3 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{10} \end{pmatrix} = \begin{pmatrix} 3 \\ 5 \\ 7 \\ 8 \\ 2 \end{pmatrix}$$

$$x \in \{0,1\}^{10}$$

After row operations described in the proof of Theorem 2.3.3, we obtain the matrix

$$A' = \begin{pmatrix} 1 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 2 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -3 & -2 & 0 & 0 & 7 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2 & 0 & -7 & 0 & 3 & 2 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -4 & -3 & 0 & 5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & -5 \end{pmatrix}$$

Now A' is the coefficient matrix of the generalized network flow problem with inequality constraints on the graph 2.3. Thus we get the following GNFP formulation of GPSP

$$\textit{Maximize} \quad 2x_1 + 3x_2 + x_3 + 2x_4 + 5x_5 + x_6 + 3x_7 + 3x_8 + 5x_9 + x_{10}$$

Subject to:

$$x_1 + x_3 \leq 3$$

$$-x_1 + 2x_3 + 2x_4 + x_5 \leq 2$$

$$-3x_2 - 2x_3 + 7x_6 + 4x_7 \leq 2$$

$$-2x_4 - 7x_6 + 3x_8 + 2x_9 \leq 1$$

$$x_5 + 4x_7 + 3x_8 - 5x + 10 \geq 6$$

$$2x_9 + 5x_{10} \geq 2$$

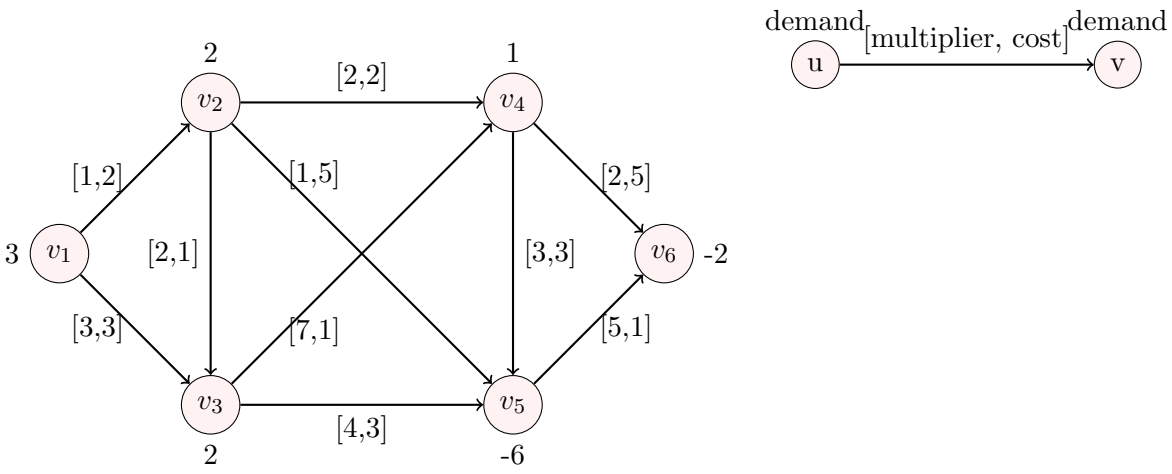


Figure 2.3: Example of GNFP transformed from GPSP

One knot GPSP

A matrix A is said to be *1-knot* if

1. Two consecutive rows have exactly one non-zero element in common column
2. All non-zero elements in a column are the same
3. Two non-consecutive rows have no column with two non-zero entries

Such a matrix can be represented as A below (if necessary by renaming columns)

$$A = \begin{pmatrix} a_1 & \cdots & a_{l_1} & & & & & & \\ & & a_{l_1} & \cdots & a_{l_2} & & & & \\ & & & & \ddots & & & & \\ & & & & & a_{l_{m-2}} & \cdots & a_{l_{m-1}} & \\ & & & & & & & a_{l_{m-1}} & \cdots & a_{l_m} \end{pmatrix} \tag{2.26}$$

where $l_m = n$.

A GPSP with the coefficient matrix in its GPSP1 formulation that is 1-knot is called a 1-knot GPSP (1-KPSP). Such a problem can be formulated as

$$\begin{aligned} \text{Maximize:} \quad & \sum_{j \in N} c_j x_j \\ \text{Subject to:} \quad & \sum_{j=l_{i-1}}^{l_i} a_j x_j \leq b_i \quad i \in M \\ & x_j \in \{0,1\} \quad j \in N. \end{aligned}$$

where $l_0 = 1$.

The variables x_{l_1}, x_{l_2},..., x_{l_m} are called knot variables. When $m = 1$, 1-KPSP is the knapsack problem and hence 1-KPSP is NP-hard. However, knapsack problem can be solved in pseudopolynomial time by dynamic programming [8]. We now show that 1-KPSP can also be solved by a pseudopolynomial algorithm.

Consider the subproblem $P(k)$ constitutes the first k constraints and l_k variables

$$\begin{aligned} P(k): \quad \text{Maximize:} \quad & \sum_{j=1}^{l_k} c_j x_j \\ \text{Subject to:} \quad & \sum_{j=l_{i-1}}^{l_i} a_j x_j \leq b_i \quad i = 1, 2, \ldots, k \\ & x_j \in \{0,1\} \quad j = 1, 2, \ldots, l_k. \end{aligned}$$

Let $P(k|0)$ be the restriction of $P(k)$ with the additional constraint that $x_{l_k} = 0$. Similarly, let $P(k|1)$ be the restriction of $P(k)$ with the additional constraint that $x_{l_k} = 1$. Let $V(k|0)$ and $V(k|1)$ be the optimal objective function values of $P(k|0)$ and $P(k|1)$ respectively.

Consider the knapsack problems:

$$\begin{aligned} KP(k|0|0) \quad \text{Maximize:} \quad & \sum_{j=1+l_k}^{l_{k+1}-1} c_j x_j \\ \text{Subject to:} \quad & \sum_{j=1+l_k}^{l_{k+1}-1} a_j x_j \leq b_{k+1} \\ & x_j \in \{0,1\} \quad j = l_k + 1, l_k + 2, \ldots, l_{k+1} - 1. \end{aligned}$$

$$KP(k|0|1) \quad \text{Maximize:} \quad \sum_{j=1+l_k}^{l_{k+1}-1} c_j x_j$$

$$\text{Subject to:} \quad \sum_{j=1+l_k}^{l_{k+1}-1} a_j x_j \leq b_{k+1} - a_{l_{k+1}}$$

$$x_j \in \{0,1\} \quad j = l_k+1, l_k+2, \ldots, l_{k+1}-1.$$

$$KP(k|1|0) \quad \text{Maximize:} \quad \sum_{j=1+l_k}^{l_{k+1}-1} c_j x_j$$

$$\text{Subject to:} \quad \sum_{j=1+l_k}^{l_{k+1}-1} a_j x_j \leq b_{k+1} - a_{l_k}$$

$$x_j \in \{0,1\} \quad j = l_k+1, l_k+2, \ldots, l_{k+1}-1.$$

$$KP(k|1|1) \quad \text{Maximize:} \quad \sum_{j=1+l_k}^{l_{k+1}-1} c_j x_j$$

$$\text{Subject to:} \quad \sum_{j=1+l_k}^{l_{k+1}-1} a_j x_j \leq b_{k+1} - a_{l_k} - a_{l_{k+1}}$$

$$x_j \in \{0,1\} \quad j = l_k+1, l_k+2, \ldots, l_{k+1}-1.$$

Let $W(k|0|0)$, $W(k|0|1)$, $W(k|1|0)$, $W(k|1|1)$ be the optimal objective function values of $KP(k|0|0)$, $KP(k|0|1)$, $KP(k|1|0)$, $KP(k|1|1)$ respectively. Then

$$V(k+1|0) = \max\{V(k|0) + W(k|0|0), V(k|1) + W(k|1|0) + c_{l_k}\} \tag{2.27}$$

$$V(k+1|1) = \max\left\{V(k|0) + W(k|0|1) + c_{l_{k+1}}, V(k|1) + W(k|1|1) + c_{l_k} + c_{l_{k+1}}\right\} \tag{2.28}$$

The optimal objective function value is given by

$$\max\{V(m|0), V(m|1)\}. \tag{2.29}$$

Using the recurrence relations (2.27) and (2.28) and equation (2.29), by backtracking using the corresponding solutions of the knapsack problems, an optimal solution to 1-KPSP can be constructed.

Theorem 2.3.4. *1-KPSP can be solved in $O(m + nb_{max})$ time, where $b_{max} = max_{i \in M}\{b_i\}$.*

Proof. Knapsack problem $KP(k|\cdot|\cdot)$ can be solved in $O((l_k - l_{k-1})b_k)$ time by dynamic programming, then the complexity of solving all knapsack problems is $O\left(\sum_{k\in M}(l_k - l_{k-1})b_k\right) = O(nb_{max})$. Given values $W(k|\cdot|\cdot)$, update of $V(k|\cdot)$ takes $O(m)$ time due to constant time for each $k \in M$. The total complexity of solving 1-KPSP is $O(m + nb_{max})$. □

In the above algorithm, the knapsack problems $KP(k|\cdot|\cdot)$ are solved using an exact algorithm. If we use an ϵ-approximation algorithm to solve $KP(k|\cdot|\cdot)$, an ϵ-approximate solution to 1-KPSP can be constructed. We summarize this observation in following theorem:

Theorem 2.3.5. *1-KPSP can be solved by a fully polynomial approximation scheme.*

Proof. First we show that given an ϵ-approximation algorithm $\mathcal{A}$ for knapsack problem, let $W^{\mathcal{A}}(k|0|0)$, $W^{\mathcal{A}}(k|0|1)$, $W^{\mathcal{A}}(k|1|0)$, $W^{\mathcal{A}}(k|1|1)$ be the objective function values of $KP(k|0|0)$, $KP(k|0|1)$, $KP(k|1|0)$, $KP(k|1|1)$ obtained from algorithm $\mathcal{A}$ respectively. Then we have

$$W(k|\cdot|\cdot) \leq \epsilon W^{\mathcal{A}}(k|\cdot|\cdot)$$

Let $V^{\mathcal{A}}(k|\cdot)$ be the resulting value obtained from recurrence relation (2.27) and (2.28) by substituting $W(k|\cdot|\cdot)$ with $W^{\mathcal{A}}(k|\cdot|\cdot)$. Then due to the fact that $\epsilon \geq 1$, we have

$$V(k|\cdot) \leq \epsilon V^{\mathcal{A}}(k|\cdot)$$

The results follows from equation (2.29).

Since knapsack problem can be solved by a fully polynomial approximation scheme [60], the conclusion follows. □

Extending our definition of 1-knot matrix, we define a matrix A as *r-knot* if

1. Two consecutive rows have exactly r non-zero elements in common column
2. All non-zero elements in a column are the same
3. Two non-consecutive rows have no column with two non-zero entries

Then GPSP with r-knot matrix as constraint coefficient matrix is called r-knot GPSP (r-KPSP). It can be formulated as

$$\begin{aligned} \text{Maximize:} \quad & \sum_{j\in N} c_j x_j \\ \text{Subject to:} \quad & \sum_{j=l_{i-1}-r+1}^{l_i} a_j x_j \le b_i \quad i \in M \\ & x_j \in \{0,1\} \quad j \in N. \end{aligned}$$

where $l_0 = r$, $l_m = n$.

The algorithm discussed for 1-KPSP can be extended in a natural way to obtain an algorithm for solving r-KPSP. Thus we have

Theorem 2.3.6. *r-KPSP can be solved in pseudopolynomial time for any fixed r or $r = O(\log n)$.*

Proof. For r-KPSP, let $P(k)$ be its subproblem constituting the first k constraints and l_k variables. Applying the similar idea of 1-KPSP, for any $z \in \{0,1\}^r$ we construct $P(k|z)$ as the restriction of $P(k)$ with additional constraint that knot variables $\{x_j\}_{l_k-r}^{l_k}$ are fixed as z. Let $V(k|z)$ be the optimal objective function value of $P(k|z)$. For any $y, z \in \{0,1\}^r$, consider knapsack problem $KP(k|y|z)$ defined as

$$KP(k|y|z) \quad \text{Maximize:} \quad \sum_{l_k+1}^{l_{k+1}-r} c_j x_j \tag{2.30}$$

$$\text{Subject to:} \quad \sum_{l_k}^{l_{k+1}-r} a_j x_j \le b_{k+1} - \sum_{j=l_k-r+1}^{l_k} a_j y_j - \sum_{j=1-r+l_{k+1}}^{l_{k+1}} a_j z_j \tag{2.31}$$

$$x_j \in \{0,1\} \quad j = l_k+1, l_k+2, \ldots, l_{k+1}-r. \tag{2.32}$$

Let $W(k|y|z)$ be the optimal objective function values of $KP(k|y|z)$. Then the recurrence relation becomes

$$V(k+1|z) = \max_{y\in\{0,1\}^r} \{V(k|y) + W(k|y|z)\} + \sum_{j=1-r+l_{k+1}}^{l_{k+1}} c_j z_j \tag{2.33}$$

The optimal objective function value is given by

$$\max_{z\in\{0,1\}^r} V(m|z) \tag{2.34}$$

Then r-KPSP can be solved by using dynamic programming with recurrence relation defined by (2.33) and equation (2.34). The optimal solution can be constructed by backtracking the corresponding solutions of subproblems.

For each k, there are 2^{2r} knapsack problems generated during the process, and $KP(k|\cdot|\cdot)$ can be solved in $O((l_k - l_{k-1})b_k)$ time. Given values $W(k|\cdot|\cdot)$, the update of $V(k|\cdot)$ take $O(m2^{2r})$ time in total. Hence r-KPSP can be solved in $O(m2^{2r} + n2^{2r}b_{max})$ time, which is pseudopolynomial for $r = O(\log n)$. □

Theorem 2.3.7. *r-KPSP can be solved by a fully polynomial approximation scheme when $r = O(\log n)$.*

Proof. The conclusion follows from techniques in Theorem 2.3.6 with similar deduction in Theorem 2.3.5. □

Nested GPSP

In this section we will discuss another special structure of GPSP: *nested* GPSP (NGPSP). Here "nest" refers to the structure of constraint coefficient matrix A. We say a GPSP is nested if in its GPSP1 formulation the nonzero entries in a row of A are all contained in its successor, i.e., $M_i \subseteq M_j$ when $i < j$. After some adjustment of index, the coefficient matrix A of NGPSP can be formulated as

$$A = \begin{pmatrix} a_1 & a_2 & \dots & a_{l_1} & & & \\ a_1 & a_2 & \dots & a_{l_1} & \dots & a_{l_2} & \\ & & & \dots & & & \\ a_1 & a_2 & \dots & a_{l_1} & & \dots & a_{l_m} \end{pmatrix} \tag{2.35}$$

where $a_{l_m} = n$. NGPSP is a special case of GPSP-P.

The nested structure leads to simplicity for solving. For example, CPLEX is able to solve NGPSP within one minute when n is up to 5000 and $m = 0.5n$. This result is impressive if we consider that the general GPSP can not be solved optimally even when $n = 500$.

Theorem 2.3.8. *For NGPSP, if it satisfies a_j is nondecreasing and c_j is nonincreasing*

$$a_1 \geq a_2 \geq \ldots \geq a_n,$$
$$c_1 \leq c_2 \leq \ldots \leq c_n,$$

then the problem is polynomially solvable.

Proof. Any problem of the type NGPSP with n variables, we claim that there exist an optimal solution x^0 with $x_n^0 = 1$. Suppose this is not true. Let x^* be the optimal solution and k is the first index such that $x_k^* = 1$, $k < n$. Then consider $\bar{x}$ given by

$$\bar{x}_j = \begin{cases} x_j^* & \text{if } j \neq k, n \\ 0 & \text{if } j = k \\ 1 & \text{if } j = n \end{cases}$$

$\bar{x}$ is clearly feasible and $\sum_{j \in N} c_j x_j^* \leq \sum_{j \in N} c_j \bar{x}_j$. Thus $\bar{x}$ is also optimal contradicting our assumption. Eliminate the variable x_n from NGPSP by setting $x_n = 1$, and adjust the right hand side b_n as $b_n - a_n$. Let NGPSP$(n-1)$ be the resulting problem. If $x^0 = (x_1^0, x_2^0, \ldots, x_{n-1}^0)$ is an optimal solution to NGPSP$(n-1)$ then $(x_1^0, x_2^0, \ldots, x_{n-1}^0, 1)$ is an optimal solution to NGPSP(n). If the last two rows of NGPSP$(n-1)$ are identical, one of them is redundant and can be discarded. The resulting NGPSP$(n-1)$ has the same structure as NGPSP(n) and hence $x_{n-1}^0 = 1$ in an optimal solution $(x_1^0, x_2^0, \ldots, x_{n-1}^0)$. The proof now follows from recursion.

□

Note that this result is not true for the general case as the following example shows.

Example 2.3.2.

$$\textit{Maximize} \sum_{j=1}^{n-1} x_j + 2x_n$$

Subject to:

$$x_j + x_n \leq 1 \quad j \in N \backslash \{n\}$$
$$x_j \in \{0, 1\} \quad j \in N$$

Although nth item has higher profit and the same capacity consumption, its selection prevents all the other items' from being chosen. The greedy algorithm described in above proof fails to give an optimal solution.

2.4 Heuristic algorithms description

In this chapter we discuss various heuristics for GPSP and study the quality of the solution produced by systematic experimental analysis. We present several heuristics based on standard algorithmic ideas studied extensively for various combinatorial optimization problems. In particular, we develop the algorithms from the following classes:

1. greedy algorithm,
2. semi-greedy algorithm, and
3. multi-start local search algorithm with 2-swap neighborhood.

2.4.1 Greedy algorithm

The greedy algorithm is an adaptation of the corresponding greedy algorithms studied in the context of MDKP [58, 49]. The "greediness" is controlled by a number r, which is called the *utility ratio*. We consider eight different types of utility ratios which are summarized in Table 2.1. The column "Complexity" gives the complexity of both calculating and ordering the variables. Ratio 1 and 2 are new and others are adopted (derived) from the study of MDKP [58, 50, 34].

Ratio	Definition	Remark	Complexity
1	$\frac{c_j}{a_j} - \beta \sum_{i \in M_j} \frac{a_j}{b_i}$	$\beta = 0.25$	$O(L + n \log n)$
2	$\frac{c_j}{a_j} - \beta q_j, q_j = \max_{i \in M_j} \frac{a_j}{\sum_{k \neq j, k \in N_i} a_k}$	$\beta = 0.25$	$O(L \max_i\{\lvert N_i \rvert\} + n \log n)$
3	$c_j / (\sum_{i \in M_j} \mu_i a_j)$	μ	-
4	$c_j / (\lvert M_j \rvert a_j)$		$O(n \log n)$
5	c_j / a_j		$O(n \log n)$
6	c_j		$O(n \log n)$
7	j		$O(1)$
8	random		-

Table 2.1: GPSP greedy algorithm utility ratios

The static greedy algorithm chooses a utility ratio r and identifies a permutation π of

$N = \{1, 2, \ldots, n\}$ such that

$$r_{\pi(1)} \geq r_{\pi(2)} \geq \ldots \geq r_{\pi(n)} \tag{2.36}$$

First of all, the instance-independent Ratio 7 and 8 are presented for the comparison reason. The complexity of Ratio 8 depends on the randomness system. Ratio 7 and its corresponding permutation are obtained without calculation, hence its complexity is $O(1)$. Ratios 5 and 6 are a group of intuitive ratios. Ratio 6 considers the variable contribution to the objective function value which is c_j. Its complexity lies in obtaining the corresponding permutation which takes $O(n \log n)$ time. Ratio 5 considers the price-performance ratio $\frac{c_j}{a_j}$ for $j \in N$. The calculation includes $\frac{c_j}{a_j}$ and corresponding permutation, which is $O(n) + O(n \log n) = O(n \log n)$. Ratio 3 and 4 are variations of ratio $c_j/(\sum_{i \in M} u_i a_{ij})$ where u is called surrogate multiplier. In our case it becomes $c_j/(\sum_{i \in M_j} u_i a_j)$ due to the problem structure. In Ratio 4 $u_i = 1$ and its complexity is $O(n \log n)$. In Ratio 3 u is set to be the optimal solution of the dual problem of the GPSP1 LP relaxation problem. We develop Ratio 1 and 2 based on the problem nature. The term $\frac{c_j}{a_j}$ are included in both ratios for the same reason of Ratio 5. The value $\frac{a_j}{b_i}$ measures the capacity usage of variable x_j respect to constraint i. The larger it is, the more capacity consuming it is to select path P_j. Hence, a weighted summation of these ratios is subtracted from $\frac{c_j}{a_j}$ in Ratio 1. It takes $O(M_j)$ time to calculate r_j, hence the complexity of Ratio 1 is $O(L) + O(n \log n) = O(L + n \log n)$. In Ratio 2, we measure the relative capacity consumption defined as $\frac{a_j}{\sum_{k \neq j, k \in N_i} a_k}$ of x_j for constraint i instead. The weighted maximum of this value among all $i \in M_j$ is subtracted from $\frac{c_j}{a_j}$ to get Ratio 2.

We scan variables x_j in the order given by π and set its value to 1 if feasibility is not violated. Let $(x|x_k = 1)$ represent the solution x' such that $x'_j = x_j$ for $j \neq k$ and $x'_k = 1$. A formal description of the static greedy algorithm is given below.

Algorithm 1: Greedy Algorithm for GPSP

Input: permutation π based on the utility ratio r satisfying (2.36)

Initialize $x \leftarrow \mathbf{0}$;

for $j \leftarrow$ *1 to* n **do**

  if $(x|x_{\pi(j)} = 1)$ *is feasible* **then**

    $x_{\pi(j)} = 1$;

return x

Feasibility checking of setting $x_j = 1$ costs $O(|M_j|)$ time. Thus the complexity of feasibility checking is $O(L)$ in total (recall that $L = \sum_{j \in N} |M_j|$). Let ψ_r be the computational

complexity to calculate the pseudo utility ratio r and to obtain corresponding permutation as given in Table 2.1. Then the complexity of the greedy algorithm is $O(L + \psi_r)$.

2.4.2 Semi-greedy algorithm

In the greedy algorithm we always consider the first available variable with highest possible utility ratio. In the semi-greedy algorithm we pick an element randomly from a candidate list which constitutes of variables with "good enough" utility ratio values, and scan all the variables until the candidate list is empty. The greedy selection is repeated several iterations and the best found solution over all iterations is returned. The parameters length of candidate list $lCand$ and iteration limit $iLim$ together determine the semi-greedy algorithm. When $lCand = 1$ and $iLim = 1$, the algorithm reduces to the greedy algorithm. Semi-greedy type algorithms are successfully studied in the context of vehicle routing problem [23]. Let $f(\cdot)$ denote the objective function. A formal description of the semi-greedy algorithm is given below:

Algorithm 2: Semi-greedy Algorithm for GPSP

Input: $lCand$, $iLim$, permutation π determined by utility ratio r

Initialize: $x^* \leftarrow \mathbf{0}$, $iter \leftarrow 1$;

repeat

  $next \leftarrow lCand + 1$, $x \leftarrow \mathbf{0}$, candidate list $C \leftarrow \{\pi(1), \pi(2), \ldots, \pi(lCand)\}$;

  while $C \neq \emptyset$ **do**

    $j \leftarrow$ select a random element in C ; `/* random selection */`

    if $(x|x_j = 1)$ *is feasible* **then**

      $x_j = 1$;

    if $next \leq n$ **then** `/* update` C `*/`

      $C \leftarrow C \cup \{\pi(next)\} \setminus \{j\}$;

      $next \leftarrow next + 1$; `/*` $next \leftarrow$ `next index to add in` C `*/`

    else

      $C \leftarrow C \setminus \{j\}$;

  if $f(x) > f(x^*)$ **then** `/* update best known solution */`

    $x^* \leftarrow x$;

  $iter \leftarrow iter + 1$;

until $iter = iLim$;

return x^*

We choose $lCand$ as an instance independent parameter, hence the complexity of semi-greedy algorithm is that of the corresponding greedy algorithm times the iteration limit $iLim$.

2.4.3 Multi-start local search algorithm

The multi-start is a strategy combined with neighborhood searching to overcome getting trapped at a local optimum. We develop a multi-start local search algorithm for GPSP based on 1-flip and 2-swap neighborhood. The algorithm starts from an initial solution generated by semi-greedy algorithm, and applies first-improving local search (the search moves to the first improved solution) based on 1-flip and 2-swap neighborhood until a local optimum is reached. These two steps are seen as one iteration. We perform $iLim$ iterations where $iLim$ is a prescribed parameter and return the best known solution found in the whole

procedure. Due to the diversity of initial solution obtained from semi-greedy algorithm, different solution spaces are expected to be explored. A formal description of the algorithm is given below:

Algorithm 3: Multi-start local search algorithm for GPSP

Input: iteration limit $iLim$
Initialize: $x^* \leftarrow \mathbf{0}$;
for $iter \leftarrow 1$ *to* $iLim$ **do**
  $x \leftarrow$ obtained from semi-greedy algorithm;
  $improve \leftarrow true$;
  while $improve$ **do**
    $improve = false$;
    for $add \in J0(x)$ *and* $!improve$ **do**
      if $(x|x_{add} = 1)$ *is feasible* **then**
        $x_{add} = 1$;
        $improve \leftarrow true$;
      else
        for $drop \in J1(x)$ *and* $!improve$ **do**
          if $c_{add} > c_{drop}$ *and* $(x|x_{add} = 1, x_{drop} = 0)$ *is feasible* **then**
            $x_{add} = 1$, $x_{drop} = 0$;
            $improve \leftarrow true$;
  if $f(x) > f(x^*)$ **then**
    $x^* \leftarrow x$;
return x^*.

where $J0(x) = \{j \in N : x_j = 0\}$, $J1(x) = \{j \in N : x_j = 1\}$.

2.5 Computational results and analysis

In this section we report results of extensive computational experiments to analyze the efficiency of the developed heuristic algorithms. In order to conduct effective computational experiments and draw reasonable conclusions, it is important to use general enough test bed. No standard test sets are available for GPSP. Thus we first develop a problem instance

generator. The algorithm performance results follows.

2.5.1 Instance generator and test bed description

Because of the equivalence between GPSP and CMDKP established in Theorem 2.2.1, instead of generating GPSP instances, we generate instance of CMDKP. Recall that a CMDKP instance is specified by constraint coefficient matrix A, constraint right hand side vector b and objective function coefficient vector c. Moreover, due to the special structure of A, a vector a and a family of subsets $\{N_i \in N : i \in M\}$ together determine A .

This instance generator is inspired by the instance generator of MDKP [20, 28] and follows similar procedures as in [28].

We use the following functions within our generator:

- $urand(l, u)$: generates uniformly distributed random integer x such that $l \leq x \leq u$.
- $nrand(l, u)$: generates normally distributed random integer x with mean $\frac{l+u}{2}$ such that $l \leq x \leq u$ is satisfied with high probability (99%). If $x < l$, it is reset to l and if $x > u$, it is reset to u in our application.

To generate a_j values, we simply use $urand(l, u)$ or $nrand(l, u)$ depending on the desired data type with l and u as input parameters which may depend on the instance size.

To generate the set N_i, we first generate $|N_i|$ for all $i \in M$ as a random integer between two input values, say l and u with $l \leq u$. Once these values are generated, N_i is filled with random integers from $\{1, 2, \ldots, n\}$ for all $i \in M$. The a_j values together with the sets N_i, $i \in M$ defines our tentative coefficient matrix A'. We then examine the columns of A'. If there are any columns with all zero entries, for each such column j we generate two row numbers, say r_1, r_2, from $\{1, 2, \ldots, m\}$, then extend N_{r_1} to $N_{r_1} \cup \{j\}$ and N_{r_2} to $N_{r_2} \cup \{j\}$.

The right hand side value b_i is generated as a function of a_j and N_i using the formula

$$b_i = \max \left\{ \alpha \sum_{j \in N_i} a_j, \max_{j \in N_i} a_j \right\} \quad i \in M, \tag{2.37}$$

where $\alpha \in (0, 1)$ is an input parameter. The smaller the value of α, the tighter the corresponding constraint is.

We then generate the c_j values. Depending on the nature of the c_j values, the instances generated are classified into

1. random instances (denoted by R)
2. positively correlated instances (denoted by P)
3. negatively correlated instances (denoted by N)
4. uniform instances (denoted by U)
5. semi uniform instances (denoted by S)
6. PSP inspired instances (denoted by W)

For random instances, we simply choose c_j as a uniformly distributed random integer. Positively correlated instances are generated by first generating a random c vector and rearrange its elements so that c_j's and a_j's have the same ordering. Negatively correlated instances are generated in a similar way except that c_j's and a_j's now have opposite orientation. For uniform instances, we set $a_j = c_j$, $\forall j \in N$ and for semi uniform instances c_j is set to $a_j + urand(l, u)$, where l and u are input parameters. In PSP inspired instances c_j is generated as $\sum_{i \in M_j} w_i$, where $w_i, i \in M$ is generated as $urand(1, 20)$.

In order to assess the nature of the different classes of problem instances generated, we conduct some preliminary experiments by setting $n = 200$ and generate large number of problems with different characteristics and solve the resulting GPSP problems using CPLEX applied on the GPSP1 formulation. The following set of parameter values are used:

- m/n: the ratio of number of constraints to number of variables;
- α: constant used in formula (2.37).
  When $\alpha = 0$, for each constraint we randomly set it as 0.25, 0.50 or 0.75;
- rL: the lower bound of cardinality of set N_i, set as 2 by default;
- rU: the upper bound of cardinality of set N_i;
- U^{au}: the upper bound parameter of a_j with uniform distribution. Then the upper bound of u of a_j is defined as $u = U^{au} * n + 20$. Note that adding 20 is to ensure that when n is small, there is still a room for diverse a_j value;
- N^{au}: the upper bound parameter of a_j with normal distribution. Then the upper bound u of a_j is defined as $u = N^{au} * n + 20$;

- aL: the lower bound of a_j, set as 1 by default;
- cL: the lower bound of c_j, set as 1 by default;
- cU: the upper bound of c_j;
- $acRel$: This parameter determines the problem class.

We analyze the above parameters' affect to the instance solvability from the solving results of CPLEX which is reported in Table 2.2. The parameters and their tested values are in column Parameter and Value respectively. For each parameter value (corresponds to each row in table), take $m/n = 0.30$ for example, we generate instances by varying all other parameter values while fixing $m/n = 0.30$. The number of these instances is shown in column Total. The average solving time of all the optimally solved instances is shown in column AveOptTime, and the percentage of instances that are solved optimally out of its total instances is presented in column AveOpt. We assume that the less AveOptNum is, the more likely that the corresponding parameter value produces difficult (to solve) GPSP instances. Also, given the same number of optimally solved instances, the long AveOptTime is, the more difficult the resulting instances are. The solving time limit of CPLEX is set to be one minute. We give thorough analysis based on the results of Table 2.2 below.

Regarding instance size, we fix variable size n at 200 and ratio $\kappa = \frac{m}{n}$ ranges from 0.3 to 5, which determines the number of the constraints m. As κ increases, the instance size increments. As expected, the optimally solved instances number drops as κ increases. There is an exception when κ increases from 3 to 5, it is probably due to the dramatic shrink of feasible solution set resulting from larger constraints number.

Regarding α, which determines the tightness of constraints according to (2.37), we test the value as 0.25, 0.50 and 0.75. When $\alpha = 0$, we randomly set α as 0.25, 0.50 or 0.75 for each constraints. Hence it is a mixture of the three. As we can see from the second block of Table 2.2, medium value $\alpha = 0.5$ has 34% instanced solved optimally which is the lowest in comparison with 51% for $\alpha = 0.25$ and 43% for $\alpha = 0.75$. This is in accordance with our expectation. When α is small, the set of feasible solutions is smaller. Whereas when α is large, most of the variables may be fixed at 1. The reduced feasible solution set may also be the reason why $\alpha = 0.25$ produces more difficult instances than $\alpha = 0.75$. We observe that mixed type $\alpha = 0$ yields 67% instances to be solved optimally which is the largest among all.

Regarding rL, rU, they together determine the cardinality of sets $N_i, i \in M$. We fix $rL = 2$ so that there is no redundant constraints. rU is set to be $0.2n$ and $0.5n$. Since the larger rU is, the denser matrix A is, $rU = 0.5n$ produces instances that are harder to solve .

Regarding parameters of setting a_j, uniform distribution and normal distribution can be compared from rows where $U^{au} = 0.02$ and $N^{au} = 0.02$. According to the percentage of optimally solved instances number 53% vs. 41%, normal distribution leads to more difficult instances. Further, we compare $U^{au} = 0.01$ and $U^{au} = 0.02$. When $n = 200$, the difference between them is not much. However, from the AveOptTime and AveOptNum, it still shows that $0.01n$ produces slightly harder to solve instances.

Regarding *acRel*, it seems that this parameter heavily affects the solvability of resulting instances. Negatively correlated instances (AveOptNum=100%) and randomly generated instances (AveOptNum=93%) are easiest to solve. Semi uniform instances (AveOptNum=30%) and PSP inspired instances (AveOptNum=34%) are on the medium level while positively correlated instance (AveOptNum=19%) and uniform instance (AveOptNum=18%) are hardest to solve.

Parameter	Value	AveOptTime	AveOptNum(%)	Total
n	200	18.88	49	1440
m/n	0.30	20.86	74	288
	0.50	18.55	56	288
	1.00	23.86	45	288
	3.00	13.10	33	288
	5.00	18.03	35	288
α	0.25	25.43	51	360
	0.50	14.86	34	360
	0.75	10.31	43	360
	0.00	24.92	67	360
rU	0.20	21.21	55	720
	0.50	16.55	43	720
U^{au}	0.01	19.16	53	480
	0.02	22.98	52	480
N^{au}	0.02	14.50	41	480
cU	0.01	19.19	48	720
	0.02	18.57	49	720
acRel	rand(R)	29.84	93	240
	pos(P)	11.65	19	240
	neg(N)	5.80	100	240
	semi(S)	25.82	30	240
	uni(U)	11.84	18	240
	psp(W)	28.34	34	240

Table 2.2: GPSP instance generator parameters comparison

We further analyze the solving process of CPLEX on different types of instances to understand the problem nature. According to the above analysis, we only consider positively correlation instance, PSP inspired instance and random instance that represent different level of hardness. The other instance generator parameters are fixed as $n(1000)$, $m/n(0.5)$, $\alpha(0.50)$, $rL(2)$, $rU(0.5n)$, $U^{au}(0.1)$, $aL(1)$, $cL(1)$, $cU(0.1m)$ and 30 instance are generated for each type of instance. We record the average objective function values of the 30 instances over the time period 60s, 90s, 120s, 150s, 180s, 210s during CPLEX solving process. The results are are presented in Figure 2.4 to 2.6.

We can see that for positively correlated instances and PSP inspired instances, CPLEX stops improving after a certain amount of time. This implies that it either finds an optimum but fails to prove optimality in the given time, or quickly gets stuck at a deep local maximum. However, for random instance CPLEX keeps improving trend in our observation period.

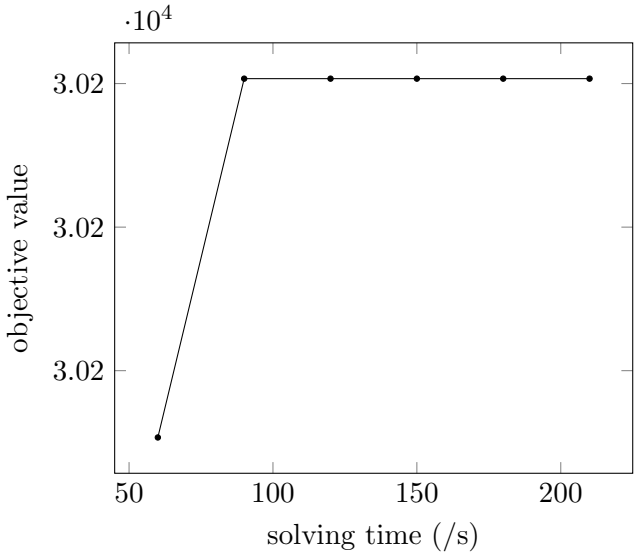


Figure 2.4: GPSP CPLEX-200s solving process observation on P instance

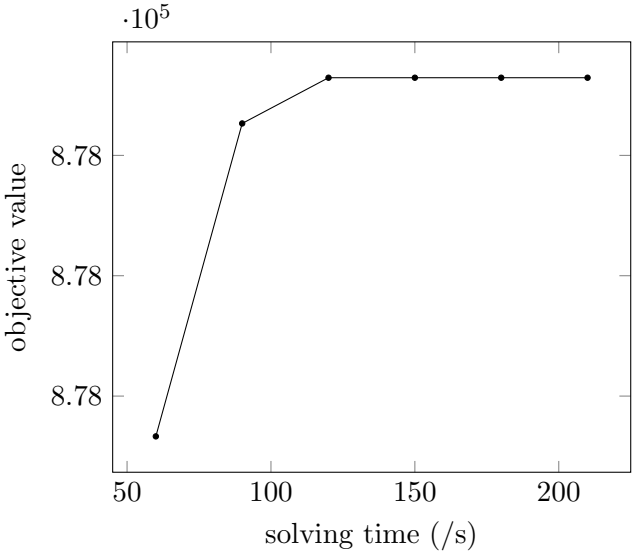


Figure 2.5: GPSP CPLEX-200s solving process observation on W instance

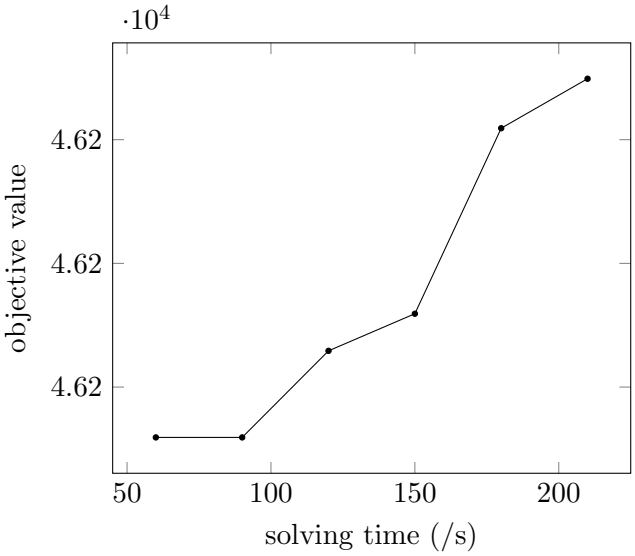


Figure 2.6: GPSP CPLEX-200s solving process observation on R instance

Now we build our test bed by using above instance generator to test the developed heuristic algorithm. We consider different instance classes with various instances size which is defined by (n, m): n(1000, 2000, 5000), m/n(0.1, 0.15, 0.2, 0.25, 0.3). Given instance type and size, the other instance generator parameters are fixed as follows to produce

relatively hard to solve instances: $\alpha(0.50), rL(2), rU(0.5n), U^{au}(0.05), aL(1), cL(1), cU(20)$. To eliminate the bias, 5 instances are generated for each set of fixed instance generator parameters.

The data structure for each instance includes path cost vector c and path capacity usage vector a, edge capacity vector b and edge cost vector w (only in the case of PSP inspired instances). Sets $N_i, i \in M$ and $M_j, j \in N$ are also stored. We maintain a non-zero entry matrix $A' \in Z^{m \times n}$, of which the (i, j)th element is defined as follows:

$$a'_{ij} = \begin{cases} 1 & \text{if } j \in N_i \\ 0 & \text{otherwise} \end{cases}$$

Given A' and a,, the coefficient matrix A is well defined.

All the instances in the test bed are solved by CPLEX given 1 minute, 10 minutes and 1 hour as time limit. The results are presented in the Table 2.3 for positively correlated instances. Different instance sizes are given in the first two columns. The result is reported from the aspects of objective function value, gap percentage between best known upper bound and lower bound returned by CPLEX and the computational time, which are presented in column Obj, Gap and Time respectively. For each instance size (n, m), we calculate the average, maximum and minimum of above values over the 5 randomly generated instances and summarize them in Table 2.3.

We observe that CPLEX reached a good quality solution quickly within one minute and keep improving very slowly in the remaining time, especially for the smaller instances where $n = 1000$. Take the first instance size $(1000, 100)$ for example, CPLEX obtained solution with objective function value 19429, 19455 and 19458 within the three time periods. From 1 minute to 10 minute, the increase in objective function value is 26 and from 10 minutes to 1 hour the improvement is 3.

		AVE			MAX			MIN		
n	m	Obj	Gap	Time	Obj	Gap	Time	Obj	Gap	Time
time = 60s										
1000	100	19429	0.63	60.11	19887	0.67	60.12	19032	0.55	60.10
1000	150	19221	0.97	60.09	19567	1.03	60.10	18682	0.90	60.09
1000	200	18993	1.41	60.07	19317	1.65	60.10	18765	1.17	60.06
1000	250	18801	1.70	60.06	19202	1.98	60.08	18463	1.46	60.05
1000	300	18464	2.01	60.08	18878	2.34	60.12	17888	1.73	60.03
2000	200	65397	0.71	60.08	65953	0.85	60.14	64600	0.60	60.06
2000	300	64083	1.11	60.09	65387	1.44	60.15	63444	0.93	60.04
2000	400	63138	1.39	60.05	64643	1.52	60.06	61908	1.30	60.04
2000	500	62180	1.75	60.07	63886	2.01	60.08	60432	1.60	60.06
2000	600	62292	2.05	60.07	63355	2.34	60.09	60644	1.79	60.04
5000	500	360668	0.69	60.14	362306	0.89	60.22	358234	0.57	60.11
5000	750	356587	1.11	60.20	359674	1.14	60.28	350058	1.04	60.15
5000	1000	350757	1.28	60.31	352323	1.47	60.33	347805	1.17	60.29
5000	1250	346362	3.29	60.42	355756	10.37	60.51	324287	1.38	60.34
5000	1500	319565	11.03	60.41	324119	13.82	60.64	311387	10.16	60.25
time = 600s										
1000	100	19455	0.47	601.97	19921	0.51	602.09	19058	0.43	601.86
1000	150	19239	0.85	602.03	19593	0.94	602.45	18703	0.78	601.68
1000	200	19032	1.17	602.09	19366	1.24	602.45	18798	1.06	601.69
1000	250	18856	1.38	601.96	19292	1.48	602.12	18487	1.30	601.76
1000	300	18532	1.61	601.46	18964	1.78	601.89	17972	1.46	601.09
2000	200	65496	0.55	602.38	65998	0.60	602.92	64658	0.52	601.92
2000	300	64311	0.75	601.98	65556	0.83	602.21	63641	0.67	601.75
2000	400	63199	1.29	601.69	64746	1.60	602.17	62128	1.16	600.89
2000	500	62216	1.70	600.57	63886	1.78	600.80	60432	1.60	600.34
2000	600	62417	1.84	601.23	63355	1.90	603.04	60923	1.77	600.29
5000	500	361102	0.57	600.81	362364	0.59	601.19	359117	0.55	600.64
5000	750	357833	0.75	601.15	361107	0.77	602.41	351216	0.73	600.66
5000	1000	351890	0.96	600.90	353409	1.26	601.09	349152	0.83	600.82
5000	1250	353451	1.11	602.57	357255	1.42	607.09	350406	0.96	600.86
5000	1500	350454	1.23	601.30	353900	1.40	601.62	345675	1.07	601.08
time = 3600s										
1000	100	19458	0.44	3602.98	19921	0.49	3603.39	19059	0.42	3602.65
1000	150	19253	0.77	3602.67	19601	0.80	3603.01	18715	0.73	3602.42
1000	200	19045	1.09	3602.62	19370	1.20	3603.26	18802	0.99	3602.21
1000	250	18870	1.28	3602.65	19304	1.39	3603.57	18495	1.22	3601.97
1000	300	18541	1.53	3602.77	18986	1.61	3603.16	17972	1.35	3602.23
2000	200	65503	0.54	3602.70	65999	0.57	3602.97	64665	0.51	3602.24
2000	300	64333	0.71	3602.76	65565	0.80	3603.99	63672	0.63	3602.05
2000	400	63399	0.96	3602.23	64893	1.00	3604.56	62291	0.89	3601.31
2000	500	62526	1.19	3602.51	64093	1.33	3604.44	60764	1.09	3601.28
2000	600	62630	1.49	3605.09	63604	1.54	3609.34	61193	1.42	3601.69
5000	500	361102	0.57	3606.96	362364	0.59	3613.09	359117	0.55	3601.27
5000	750	357833	0.75	3606.47	361107	0.77	3612.02	351216	0.73	3600.91
5000	1000	352197	0.87	3602.07	353409	0.93	3602.66	349152	0.82	3601.61
5000	1250	353705	1.04	3604.57	357255	1.12	3606.48	351677	0.96	3601.89
5000	1500	350863	1.11	3604.96	353900	1.25	3608.03	345675	1.00	3601.88

Table 2.3: GPSP - the results of CPLEX on Test bed P: time limit

To get a better view, we plot the average objective function values obtained from CPLEX during different time periods in the following graphs for $n = 1000, 2000, 5000$ respectively. Except for instance size $(5000, 1250)$, i.e., $m/n = 0.25$ and $(5000, 1500)$, i.e., $m/n = 0.3$, there is a uniform trend that only a slight improvement happens during the different times periods. However, for the other two instances, we still observe very close objective function values between time limit 10 minutes and 1 hour.

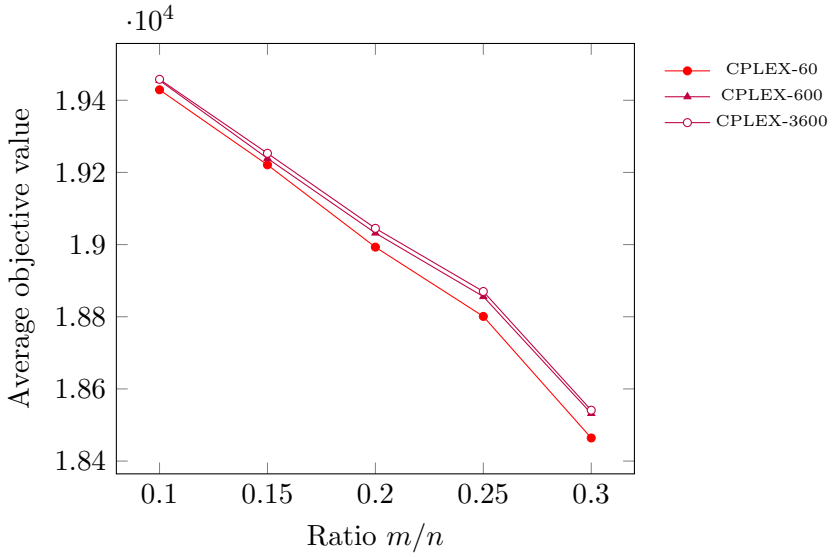


Figure 2.7: GPSP CPLEX time limit comparison on Test bed P $n = 1000$

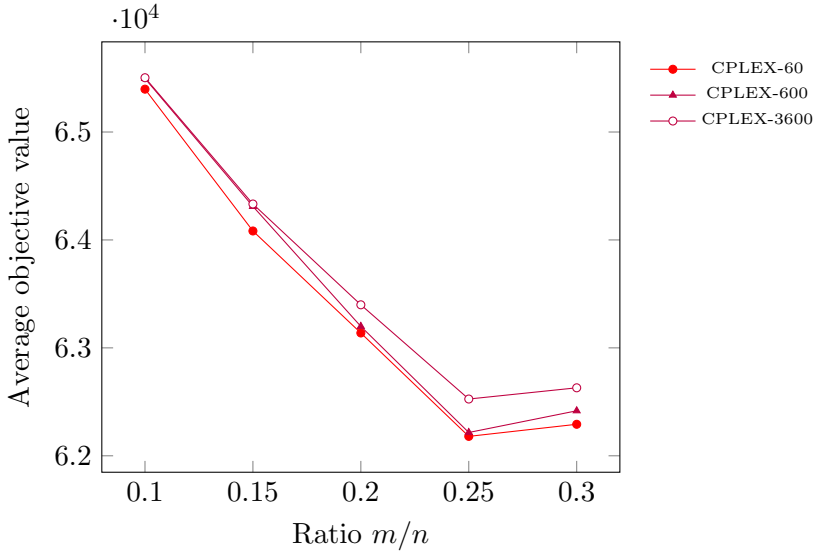


Figure 2.8: GPSP CPLEX time limit comparison on Test bed P $n = 2000$

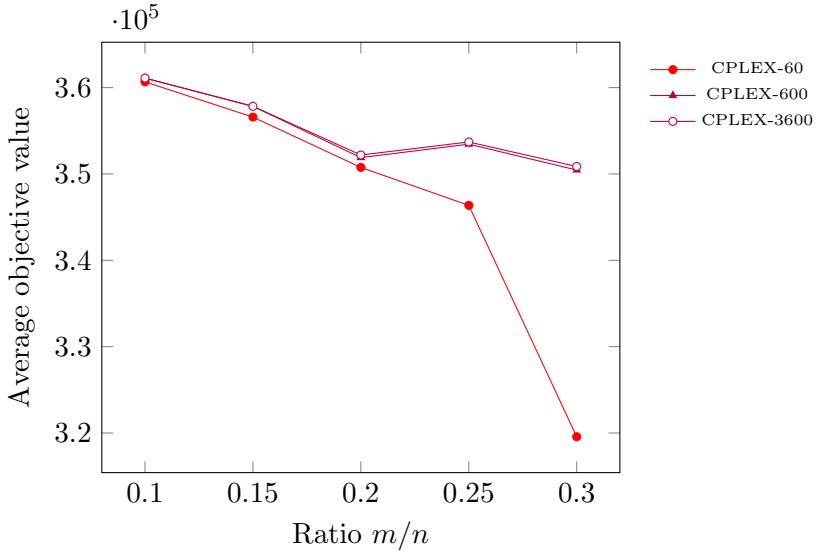


Figure 2.9: GPSP CPLEX time limit comparison on Test bed P $n = 5000$

2.5.2 Experimental analysis

All algorithms are coded in `C++` and complied in Visual Studio 2010. We use CPLEX1.25 of ILog 64-bit version with all default settings as the integer programming solver. All

experiments are conducted on a PC with Intel Core i703770 CPU @ 3.40GHz, RAM 16 GB.

We set as criteria to measure algorithm performance: computational time and solution quality. We use optimally solved instance number and relative percentage deviation from the optimal objective function value if available or from that of the LP relaxation problem otherwise to indicate solution quality. More precisely, let q^*, q^0 and q^A be the objective function value of GPSP1, its LP relaxation problem and that of a solution obtained from algorithm A. Then if q^* is available, the deviation percentage is defined as

$$\delta = \frac{q^* - q^A}{q^*} \times 100$$

Otherwise, the deviation percentage is defined as

$$\delta = \frac{q^0 - q^A}{q^0} \times 100$$

Our experiments are designed as follows:

1. Algorithm parameter tuning: for each algorithm, different algorithm parameters comparison experiments are performed on a subset of test bed.

2. Algorithm performance evaluating: for each algorithm with fixed algorithm parameters, a performance experiments are performed on the entire test bed.

Based on our instance generator parameter analysis in the previous subsection, the positively correlated instances are chosen as the test bed of our algorithm parameter tuning experiments since they represent the hardest to solve instances. The instance size is fixed at $(n, m) = (1000, 200)$ while other parameters are fixed as specified above.

Greedy algorithm with different utility ratios is tested and the results are presented in Table 2.4. We can see that Ratio 4 gives the overall best performance respect to solution quality while the computational time is on the same order as all its peers. Note that some computational time is shown as 0 in the table which implies that it is less than 10 milliseconds.

	Average		Max		Min		
Alg	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time	#Opt Inst
CPLEX	1.44	60.07	1.67	60.10	1.21	60.06	0
GREEDY-1	12.12	**0.00**	13.51	**0.00**	11.22	**0.00**	**0**
GREEDY-2	13.35	0.10	14.88	0.11	12.56	0.09	**0**
GREEDY-3	14.64	**0.00**	15.52	**0.00**	13.39	**0.00**	**0**
GREEDY-4	**6.44**	**0.00**	**7.33**	**0.00**	**5.64**	**0.00**	**0**
GREEDY-5	13.25	**0.00**	15.09	0.01	11.98	**0.00**	**0**
GREEDY-6	14.30	**0.00**	15.08	0.02	12.99	**0.00**	**0**
GREEDY-7	14.07	**0.00**	15.25	**0.00**	13.23	**0.00**	**0**
GREEDY-8	14.44	**0.00**	15.18	**0.00**	13.52	**0.00**	**0**

Table 2.4: GPSP greedy algorithm parameter comparison: utility ratio

Semi-greedy algorithm with different candidate list length $lCand$ is tested and the results are presented in Table 2.5. The more iteration we allow, the more likely we get an improved performance. However, limited to the algorithm itself, there is a limited space for this improvement, so we set $iLim = 50$. The underlying utility ratio is chosen as Ratio 4 based on the performance analysis regarding greedy algorithm above. The candidate list length $lCand$ varies from 5 to 25. As we see from the table, although the computational time is not as negligible as greedy algorithm but still very small. The $lCand = 20$ and $lCand = 25$ give roughly the same solution quality while $lCand = 20$ costs less time.

	Average		Max		Min		
Alg	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time	#Opt Inst
SEMIGREEDY-4-5-50	5.85	0.013	6.76	**0.02**	5.38	**0.000**	**0**
SEMIGREEDY-4-10-50	5.87	**0.009**	6.66	**0.02**	5.34	**0.000**	**0**
SEMIGREEDY-4-15-50	5.80	0.012	**6.42**	**0.02**	5.53	**0.000**	**0**
SEMIGREEDY-4-20-50	**5.67**	**0.009**	6.48	**0.02**	5.40	**0.000**	**0**
SEMIGREEDY-4-25-50	**5.67**	0.010	6.53	**0.02**	**5.32**	**0.000**	**0**

Table 2.5: GPSP semi-greedy algorithm parameter comparison: $lCand$

Multi-start local search algorithm with different iteration limit $iLim$ is tested and the results are reported in Table 2.6. The semi-greedy algorithm embedded in it employs greedy algorithm with Ratio 1,2, 4 and 5 randomly with iteration limit as 50 and candidate list

length 20. The iteration limit for multi-start local search algorithm varies from 10 to 100. Normally the more iterations we run, the more likely we get better quality solution with the price of longer computational time. This is also shown by our experimental results that $iLim = 100$ provides lowest average deviation 4.11% with longest computational time.

	Average		Max		Min		
Alg	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time	#Opt Inst
MULLS-10	4.15	0.696	4.51	0.76	3.82	0.652	**0**
MULLS-20	4.15	1.378	4.73	1.58	3.76	1.189	**0**
MULLS-30	4.17	2.094	4.50	2.29	3.84	1.772	**0**
MULLS-40	4.03	2.731	4.23	3.06	3.75	2.554	**0**
MULLS-50	4.07	3.446	4.36	4.02	3.90	2.959	**0**
MULLS-60	4.04	4.018	4.48	4.50	**3.67**	3.551	**0**
MULLS-70	4.13	4.647	4.36	5.08	3.88	4.133	**0**
MULLS-80	3.95	5.284	4.20	5.71	**3.67**	4.956	**0**
MULLS-90	4.02	6.043	4.29	6.75	3.78	5.380	**0**
MULLS-100	**3.89**	6.565	**4.01**	7.14	3.72	5.785	**0**

Table 2.6: GPSP multi-start local search algorithm parameter comparison: $iLim$

Finally, based on the analysis from algorithm parameter tuning, greedy algorithm with utility Ratio 4, semi-greedy algorithm with Ratio 4, $lCand = 20$, $iLim = 50$ and multi-start local search algorithm with $iLim = 100$ are tested on the whole test bed. The solving results on positively correlated instances (P) are presented in Table 2.8 to 2.10. In each table, instance size n, m are given in the first two columns. For each instance size (n, m), we calculate the average, maximum and minimum of deviations and computational time over the 5 randomly generated instances. We also present the same results for CPLEX with one hour time limit for comparison reason in Table 2.7. Note that here instead of "gap", the deviation is from the optimal objective function value of LP relaxation problem, which is as the same as for the other algorithms. In this way we can have a more fair comparison of algorithm performance.

		AVE		MAX		MIN	
n	m	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time
1000	100	0.53	3602.98	0.57	3603.39	0.49	3602.65
1000	150	0.85	3602.67	0.90	3603.01	0.79	3602.42
1000	200	1.17	3602.62	1.27	3603.26	1.10	3602.21
1000	250	1.37	3602.65	1.48	3603.57	1.32	3601.97
1000	300	1.61	3602.77	1.70	3603.16	1.43	3602.23
2000	200	0.56	3602.70	0.59	3602.97	0.53	3602.24
2000	300	0.73	3602.76	0.82	3603.99	0.65	3602.05
2000	400	0.97	3602.23	1.01	3604.56	0.90	3601.31
2000	500	1.19	3602.51	1.34	3604.44	1.10	3601.28
2000	600	1.49	3605.09	1.53	3609.34	1.41	3601.69
5000	500	0.57	3606.96	0.59	3613.09	0.56	3601.27
5000	750	0.75	3606.47	0.77	3612.02	0.73	3600.91
5000	1000	0.87	3602.07	0.92	3602.66	0.82	3601.61
5000	1250	1.03	3604.57	1.11	3606.48	0.96	3601.89
5000	1500	1.11	3604.96	1.24	3608.03	0.99	3601.88

Table 2.7: GPSP - the results of CPLEX on Test bed P

In Table 2.8, we present the solutions achieved by greedy algorithm on positively correlated testing instances (P). Regarding computational time, greedy algorithm is very fast. For smaller instance ($n = 1000$) the computational time is less than 0.01 seconds, and for largest testing instances ($n = 5000$) it is less than 0.01 seconds. Regarding the solution quality, the average deviation is between 3.44% and 7.37%, which is reasonable. Also, a clear pattern can be observed that given n, as m increases, the average deviation increases as well. This is true for all average, maximal and minimal deviation.

		AVE		MAX		MIN	
n	m	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time
1000	100	4.92	0.00	5.29	0.00	4.47	0.00
1000	150	5.32	0.00	5.68	0.00	5.10	0.00
1000	200	6.44	0.00	7.33	0.02	5.64	0.00
1000	250	6.80	0.00	7.29	0.00	6.40	0.00
1000	300	7.37	0.00	8.70	0.00	6.37	0.00
2000	200	4.70	0.00	5.00	0.00	4.25	0.00
2000	300	4.83	0.00	5.05	0.02	4.58	0.00
2000	400	5.43	0.00	6.01	0.00	5.04	0.00
2000	500	5.60	0.01	6.08	0.02	5.10	0.00
2000	600	5.84	0.00	6.10	0.01	5.32	0.00
5000	500	3.64	0.01	4.01	0.02	3.44	0.00
5000	750	3.71	0.01	3.91	0.02	3.51	0.00
5000	1000	4.00	0.00	4.31	0.02	3.82	0.00
5000	1250	3.97	0.01	4.37	0.02	3.70	0.00
5000	1500	4.41	0.01	4.58	0.02	4.03	0.00

Table 2.8: GPSP - the results of greedy algorithm on Test bed P

In Table 2.9, the solutions achieved by semi-greedy algorithm on positively correlated testing instances (P) are presented. Regarding computational time, it is not as negligible as greedy algorithm. For smaller instances, the computational time is 0.01 seconds and for largest testing instances it increases to as much as 0.25 seconds. However, within less than 1 second, semi-greedy algorithm improves the solution quality for all the testing instances. The average deviation is between 3.17% and 6.96%, which is at least 0.30% less than that of greedy algorithm on average. We also observe a pattern that for fixed n, as m increases the average deviation increases.

		AVE		MAX		MIN	
n	m	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time
1000	100	4.36	0.01	4.66	0.01	3.98	0.00
1000	150	4.84	0.01	5.18	0.02	4.64	0.00
1000	200	5.67	0.01	6.35	0.02	5.30	0.00
1000	250	6.09	0.01	6.47	0.02	5.67	0.00
1000	300	6.52	0.01	6.96	0.01	6.04	0.00
2000	200	4.29	0.03	4.52	0.03	3.78	0.02
2000	300	4.40	0.03	4.67	0.03	4.24	0.03
2000	400	4.77	0.04	4.99	0.05	4.43	0.03
2000	500	5.19	0.04	5.46	0.05	4.88	0.03
2000	600	5.42	0.05	5.92	0.05	4.80	0.05
5000	500	3.42	0.16	3.71	0.17	3.17	0.16
5000	750	3.53	0.18	3.57	0.19	3.50	0.17
5000	1000	3.76	0.20	3.95	0.22	3.43	0.19
5000	1250	3.70	0.22	3.90	0.23	3.52	0.20
5000	1500	4.13	0.25	4.29	0.27	3.87	0.23

Table 2.9: GPSP - the results of semi-greedy algorithm on Test bed P

In Table 2.10 multi-start local search algorithm solving results on positively correlated testing instances are presented. Regarding computational time, we see a dramatic increase comparing to greedy type algorithms. For smaller instances, it is less than 10 seconds while for largest testing instances it goes up to 13 minutes. In return, a further improved solutions are reported. The average deviation is between 2.37% to 4.99%, which is around 1% less than that of semi-greedy algorithm on average. However, we point out the fact that CPLEX outperforms this multi-start local search algorithm respect to both solution quality and computational time. This is because as we analyzed before, the solutions from CPLEX does not improve too much from the first 10 minute to one hour, hence the results in Table 2.7 can roughly represent the results of CPLEX in 10 minutes. In this way, on the same order of time 10 minutes, CPLEX provides better solution with deviation between 0.53% to 1.61%.

		AVE		MAX		MIN	
n	m	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time
1000	100	2.98	5.34	3.09	6.42	2.78	4.25
1000	150	3.30	6.53	3.58	7.20	2.90	6.29
1000	200	3.95	7.42	4.00	8.13	3.86	6.85
1000	250	4.54	8.06	4.87	8.58	4.11	6.64
1000	300	4.71	9.65	4.99	10.73	4.35	9.08
2000	200	2.84	28.57	3.05	31.68	2.61	23.54
2000	300	3.26	33.37	3.48	35.85	3.11	30.55
2000	400	3.55	42.48	3.74	46.00	3.25	39.61
2000	500	3.94	49.70	4.25	51.84	3.67	47.21
2000	600	4.18	55.00	4.60	61.65	3.93	49.72
5000	500	2.45	323.93	2.57	356.83	2.37	285.09
5000	750	2.62	428.80	2.81	504.18	2.51	353.23
5000	1000	2.74	533.56	2.88	607.14	2.62	449.86
5000	1250	2.90	656.40	3.09	768.94	2.74	602.44
5000	1500	3.02	778.91	3.17	876.10	2.89	698.55

Table 2.10: GPSP - the results of multi-start local search algorithm on Test bed P

A summary graph of average deviation of above three heuristic algorithms and CPLEX is given in Figure 2.10. Note that the x coordinates corresponds to the instance size in the same order as Table 2.10. The first group of segments where x from 1 to 5 represents $n = 1000$ with m/n varies from 0.1 to 0.3. The second and the third group of segments represent $n = 2000$ and $n = 5000$ respectively with various m/n from 0.1 to 0.3.

First we observe that fixing n the deviation increase as m increase for all algorithms including CPLEX. Second, the supreme relation respect to solution quality is maintained for all instance size that CPLEX is better than multi-start local search, which is better than semi-greedy algorithm and greedy algorithm. Third, for our developed heuristic algorithms, as n increases, the deviation of heuristic algorithm decreases. We believe that it is mostly due to the larger order of objective function value instead of algorithm performance itself. However, CPLEX manages to maintain the same level deviation for all instances. Finally the conclusion can be drawn that CPLEX is very efficient to solve GPSP. If a fast and reasonable solution is required, semi-greedy algorithm is recommended.

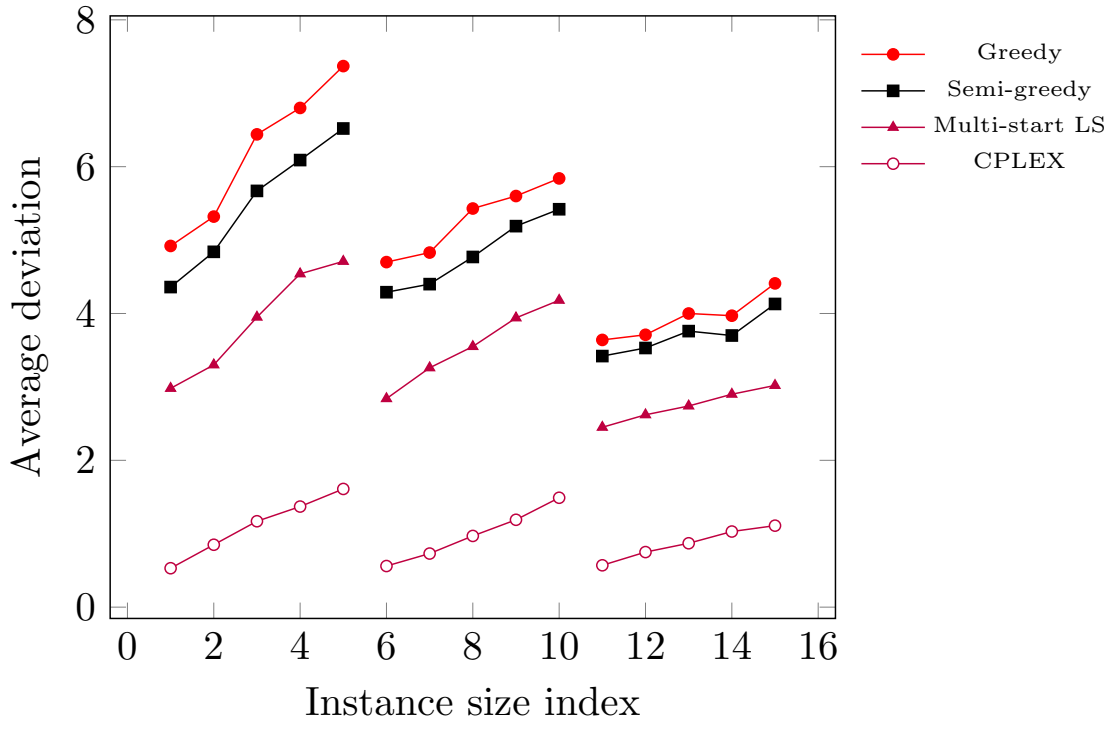


Figure 2.10: GPSP heuristic algorithms comparison on Test bed P

A summary results of the deviation and computational time for all the algorithms on Test bed P is given in Table 2.11. We can draw the same conclusion obtained from above figure.

	Average		Max		Min		
Alg	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time	#Opt Inst
CPLEX	0.99	3603.600	1.70	3613.09	0.49	3600.910	0
GREEDY-4	5.13	**0.003**	8.70	**0.02**	3.44	**0.000**	**0**
SEMIGREEDY-4-20-50	4.67	0.093	6.96	0.31	3.17	**0.000**	**0**
MULLS-100	**3.40**	197.848	**4.99**	876.10	**2.37**	4.245	**0**

Table 2.11: GPSP - the summary results of algorithms on Test bed P

Due to the limitation of space, the results on other types of instance (R, N, U, S, W) are shown in Appendix A. This is because we observe a very similar results to positively correlated instance (P) on other types of testing instances. All the algorithm performance are in accordance with that of CPLEX. In other words, the hard to solve instance for CPLEX

remains hard for our heuristic algorithms, which is indicated by the deviation from optimum or LP relaxation objective function value.

Chapter 3

Benevolent PSP

3.1 BPSP formulation

In the previous chapters, we consider PSP with objective function defined as

$$f(x) = \sum_{j \in N} c_j x_j = \sum_{j \in N} (\sum_{i \in M_j} w_i) x_j = \sum_{i \in M} (\sum_{j \in N_i} x_j) w_i$$

where w_i is the cost of edge e_i. Hence the edge cost w_i contributes to the objective value as many times as the number of selected paths containing it which is $\sum_{j \in N_i} x_j$. Corresponding to the fibre optic network design problem, the nominal cost of a path is charged to each client even though some of its segments (edges) may be shared by more than one client. It seems beneficial for the construction company. However, when the quote is too high, potential clients may cancel their projects or turn to competitor companies. Given this concern, we propose another model that the cost of each edge is equally shared by the clients (paths) that use it.

Let $G = (V, E)$ be a given graph, where the edge set $E = \{e_1, e_2, \ldots, e_m\}$ is the union of n given paths $P_1, P_2, \ldots, P_n$ in G. For each edge $e_i \in E$, a cost w_i and a capacity b_i are prescribed. Also let F^i denote the set of paths that contain edge e_i. Each path P_j has capacity usage a_j. Given a selected set of paths S, we say an edge e_i is *selected* if there is a selected path that contains it and denote this as $e_i \in S$. The *Benevolent Path Selection Problem* (BPSP) is to choose a subset of paths S such that the total cost of the selected edges $\sum_{e_i \in S} w_i$ is maximized while the paths in S obey appropriate capacity restrictions, i.e., $\sum_{P_j \in S \cap F^i} a_j \leq b_i$ for all $e_i \in E$.

We now formulate BPSP as an integer programming problem. For each $j \in N$, define x_j as (2.1). Then an edge e_i is selected if and only if there exists $j \in N_i$ such that $x_j = 1$. Hence the objective function can be expressed as

$$\sum_{i \in M} w_i \max_{j \in N_i} \{x_j\}$$

With capacity constraints

$$\sum_{j \in N_i} a_j x_j \leq b_i \quad i \in M, \tag{3.1}$$

BPSP can be formulated as the following integer programming problem:

$$\begin{aligned} \text{CCP:} \quad \text{Maximize} \quad & \sum_{i \in M} w_i \max_{j \in N_i} \{x_j\} \\ \text{Subject to:} \quad & \\ & \sum_{j \in N_i} a_j x_j \leq b_i \quad i \in M \\ & x_j \in \{0, 1\} \quad \forall j \in N \end{aligned}$$

We call the integer programming problem with above form as *Capacity Cover problem* (CCP) since the the problem is to select/cover as many rows as possible while maintaining capacity constraints.

To linearize this formulation, introduce variable

$$z_i = \begin{cases} 1 & \text{if } e_i \text{ is selected} \\ 0 & \text{otherwise} \end{cases} \qquad i \in M \tag{3.2}$$

Then the objective function can be expressed as $\sum_{i \in M} w_i z_i$. Since an edge e_i is selected if and only if there is a selected path containing it, it implies $z_i = 1$ if and only if $\sum_{j \in M_i} x_j \geq 1$. This relation can be ensured by constraints as follows:

$$z_i \leq \sum_{j \in N_i} x_j \leq |N_i| z_i \tag{3.3}$$

We refer constraints (3.3) as the *edge selection constraints*. When $\sum_{j \in M_i} x_j = 0$, $z_i = 0$ due to the first inequality. When $\sum_{j \in M_i} x_j \geq 1$, we have $z_i = 1$ resulting from the second inequality.

Combining these constraints with path capacity constraints, the BPSP can also be formulated as following integer linear programming problem:

$$\begin{aligned}
\text{BPSP1:}\quad \text{Maximize} \quad & g_1(z) = \sum_{i \in M} w_i z_i \\
\text{Subject to:} \quad & \\
& \sum_{j \in N_i} a_j x_j \leq b_i \quad i \in M \\
& z_i \leq \sum_{j \in N_i} x_j \leq |N_i| z_i \quad i \in M \\
& x_j \in \{0,1\} \quad \forall j \in N \\
& z_i \in \{0,1\} \quad \forall i \in M
\end{aligned}$$

A matrix version is also given for later convenience. Recall matrix $A = (a_{ij})_{m \times n}$ and $A' = (a'_{ij})_{m \times n}$ defined for GPSP as

$$a_{ij} = \begin{cases} a_j & \text{if } i \in M_j \\ 0 & \text{otherwise,} \end{cases} \qquad a'_{ij} = \begin{cases} 1 & \text{if } a_{ij} \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

Let diagonal matrix $D = (d_{ij})_{m \times m}$ be given as

$$d_{ij} = \begin{cases} |N_i| & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

With edge capacity vector $b = (b_1, b_2, \ldots, b_m)^T$, edge cost vector $w = (w_1, w_2, \ldots, w_m)$, $z^T = (z_1, z_2, \ldots, z_m)$ and $x^T = (x_1, x_2, \ldots, x_n)$, the matrix form of the above formulation BPSP1 can be written as

$$\begin{aligned}
\text{Maximize} \quad & g_1(z) = wz \\
\text{Subject to:} \quad & \\
& Ax \leq b \\
& z - A'x \leq 0 \\
& A'x - Dz \leq 0 \\
& x \in \mathbb{B}^n, z \in \mathbb{B}^m
\end{aligned}$$

In BPSP1, there are $m+n$ variables and $3m$ constraints. Note that when w_i is positive, the objective function drives z_i to be 1. Then the corresponding constraints $\sum_{j\in N_i} x_j \leq |N_i|z_i$ are always satisfied. In this case, the constraints number can be reduced to $2m$.

In the previous model, the decision variables represent paths and edges separately. We modify the model using additional variables so that it can be used in applications where additional constraints are added in relation to edges. For $i \in M, j \in N_i$, define y_{ij} as (2.6)

$$y_{ij} = \begin{cases} 1 & \text{if edge } e_i \text{ is selected by path } P_j \\ 0 & \text{otherwise} \end{cases}$$

Then edge capacity constraints can be formulated as

$$\sum_{j\in N_i} a_j y_{ij} \leq b_i \quad \forall i \in M$$

The path selection constraints

$$\sum_{i\in M_j} y_{ij} = |M_j| x_j \quad \forall j \in N$$

are required to ensure that the whole path is selected. Then an alternative formulation of BPSP is given below.

BPSP2: Maximize $g_1(z) = \sum_{i\in M} w_i z_i$

Subject to:

$$\sum_{j\in N_i} a_j y_{ij} \leq b_i \quad i \in M$$

$$\sum_{i\in M_j} y_{ij} = |M_j| x_j \quad j \in N$$

$$z_i \leq \sum_{j\in N_i} x_j \leq |N_i| z_i \quad i \in M$$

$$x_j \in \{0,1\} \quad \forall j \in N$$

$$y_{ij} \in \{0,1\} \quad \forall i \in M, j \in N_i$$

$$z_i \in \{0,1\} \quad \forall i \in M$$

This is a variation of the formulation given in [55]. BPSP2 has $m + n + L$ variables and $3m + n$ constraints.

Note that with variable y_{ij}, the edge selections constraints (3.3) can be replaced by

$$z_i \leq \sum_{j \in N_i} y_{ij} \leq |N_i| z_i \quad i \in M \tag{3.5}$$

The path selection constraints can also be enforced by $y_{ij} = x_j$. Then the third formulation can be formulated as follows:

$$\begin{aligned}
\text{BPSP3:} \quad \text{Maximize} \qquad & g_1(z) = \sum_{i \in M} w_i z_i \\
\text{Subject to:} \qquad & \\
& \sum_{j \in N_i} a_j y_{ij} \leq b_i \quad i \in M \\
& y_{ij} = x_j \quad i \in M, j \in N_i \\
& z_i \leq \sum_{j \in N_i} y_{ij} \leq |N_i| z_i \quad i \in M \\
& x_j \in \{0,1\} \quad \forall j \in N \\
& y_{ij} \in \{0,1\} \quad \forall i \in M, j \in N_i \\
& z_i \in \{0,1\} \quad \forall i \in M
\end{aligned}$$

There are $m + n + L$ variables and $2m + L$ constraints.

These three formulations are equivalent for BPSP. However, the following theorem shows their difference when we relax the integer restrictions. The resulting upper bounds are tighter for some than others. Let $\langle x^0, z^0 \rangle$, $\langle x^*, y^*, z^* \rangle$ and $\langle \bar{x}, \bar{y}, \bar{z} \rangle$ be optimal solutions for the LP relaxation of BPSP1, BPSP2 and BPSP3 respectively, where $x^0, x^*, \bar{x} \in \mathbb{B}^n$, $y^*, \bar{y} \in \mathbb{E}$ and $z^0, z^*, \bar{z} \in \mathbb{B}^m$.

Theorem 3.1.1. $g_1(z^0) = g_1(\bar{z}) \leq g_1(z^*)$.

Proof. Given $\langle x^0, z^0 \rangle$, define $y^0 = (y^0_{ij})$ for $i \in M, j \in N_i$ as

$$y^0_{ij} = \begin{cases} x^0_j & \text{if } i \in M_j \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

Then $\langle x^0, y^0, z^0 \rangle$ is a feasible solution to the LP relaxation of BPSP3. Due to equivalence of y_{ij} and x_j, they also share the same objective function value $g_1(z^0)$. By optimality of $\langle \bar{x}, \bar{y}, \bar{z} \rangle$, we have $g_1(z^0) \leq g_1(\bar{z})$. Given $\langle \bar{x}, \bar{y}, \bar{z} \rangle$, $\langle \bar{x}, \bar{z} \rangle$ also forms a feasible solution for

BPSP1 with the same objective function value $g_1(\bar{z})$. This implies $g_1(z^0) \geq g_1(\bar{z})$. Hence $g_1(z^0) = g_1(\bar{z})$.

In fact, $\langle x^0, y^0, z^0 \rangle$ also maintains the feasibility for BPSP2. Therefore, we have the inequality holds. □

3.2 BPSP complexity and special case study

In the previous section, we have shown that BPSP can be formulated as a CCP. In fact, any CCP can be reduced to BPSP as well.

Theorem 3.2.1. *BPSP and CCP are equivalent in the sense that they can be reduced to each other in polynomial time.*

Proof. We have formulated BPSP into the CPP before. Now let us show that any CCP can be reduced to a BPSP. Given a CCP problem, we construct a graph G of BPSP: Let $V^1 = \{v_i : i = 0, 1, \ldots, m\}$, and $E^1 = \{e_i^1 = (v_{i-1}, v_i) : i = 1, 2, \ldots, m\}$. Let the capacity and cost of e_i^1 be b_i and w_i respectively. Corresponding to each variable x_j, consider the ordered set $S_j = \{e_i^1 : a_{ij} \neq 0\}$ where the ordering of the elements in S_j are such that e_i^1 appears before e_k^1 if and only if $i < k$. Note that S_j contains at least one edge (assume w.l.o.g. the coefficient matrix of CCP has no columns of zeros), and is a collection of paths in G'. Two paths in S_j can be joined to form a single path by introducing a new node and connecting one of the end points of each path to this node by an edge with capacity ∞ and cost 0. Finally, Repeat this process while we get a single path P_j containing all edges of S_j. Let $G = (V, E)$ be the graph obtained by taking the union of all P_j, $j = 1, 2, \ldots, n$.

Note that there is a one-to-one map between the solution $x \in \{0, 1\}^n$ to CCP and the solution $S \subseteq N$ to BPSP given by

$$x_j = 1 \text{ if and only if } j \in S, \forall j \in N \tag{3.7}$$

We will show that BPSP on above constructed graph G is equivalent to the CCP under consideration, i.e., given a feasible solution to one problem, the corresponding solution defined by (3.7) to the other problem is feasible as well and they share the same objective function value.

If given $x \in \{0, 1\}^n$ is feasible to CCP, by edge capacity constraints (3.1), S satisfies capacity constraints therefore is a feasible solution, and vice versa. Also, e_i is selected if and

only if there exists $j \in N_i$ such that $j \in S$ which implies $\max_{j \in N_i}\{x_j\} = 1$ by the mapping relation, hence x and S share the same objective function value. The results follows. □

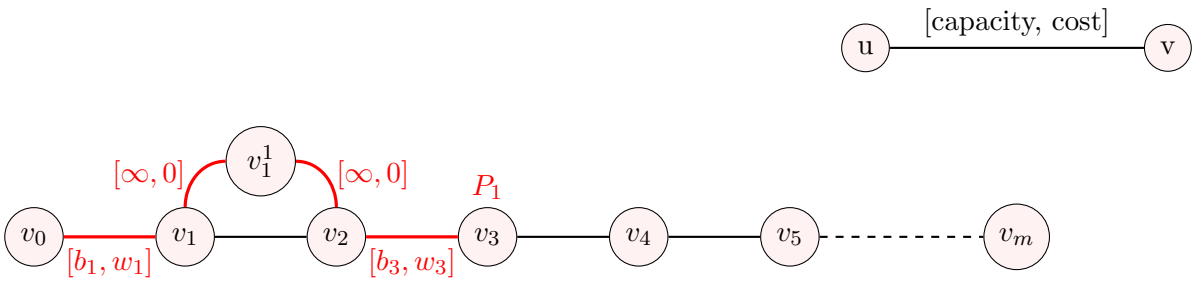


Figure 3.1: Construction BPSP from CCP

With different objectives, BPSP remains NP-hard as GPSP. However, on some special cases BPSP is shown to be solvable in polynomial time. We now present the NP-hardness of BPSP and identify several polynomial time solvable cases for it.

Theorem 3.2.2. *BPSP is NP-hard.*

Proof. We reduce the partition problem to BPSP. Given a set of number $s_1, s_2, \ldots, s_n$, the partition problem is to find a subset S of $N = \{1, 2, \ldots, n\}$ such that $\sum_{i \in S} s_i = \sum_{i \notin S} s_i$. Using this instance, we construct a GPSP instance on graph G as follows: Let vertex set $V = \{v_k\colon\ k = 0, 1, \ldots, n, n+1\}$, define edge set $E = \{(v_0, v_k) : k \in N \cup \{n+1\}\}$. For $j \in N$, path P_j consists of edge (v_{n+1}, v_0) and (v_0, v_j) with capacity usage s_j. Assign edge cost $w_{(0,n+1)} = 0$ and $w_{(0,i)} = s_i$ for $i \in N$ and edge capacity $b_{(0,n+1)} = \frac{1}{2}\sum_{j \in N} s_j$ and $b_{(0,i)} = \infty$ for $i \in N$. Then it is easy to prove that the partition problem has feasible solution if and only if BPSP based on above graph G has optimal objective value as $\frac{1}{2}\sum_{j \in N} s_j$. Due to NP-hardness of the partition problem, BPSP is NP-hard. □

Because of the proof for Theorem 3.2.2, we have

Corollary 3.2.1. *BPSP on a star graph (BPSP-S) is NP-hard.*

According to the equivalent relation between CCP and BPSP shown in Theorem 3.2.1, an immediate result follows that CCP is NP-hard as well. Here we also we point out some facts regarding the complexity of CCP:

Lemma 3.2.1. *Given a feasible solution x to CCP, a corresponding optimal solution z can be found in polynomial time. They are selected as follows:*

$$z_i = \begin{cases} 1 & \text{if } \sum_{j \in N_i} x_j > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.8}$$

However, given solution z, to find a corresponding feasible solution x is NP-hard. We prove this by reduction from the *minimum set covering problem* (MSCP). Let $\mathcal{U}$ be a finite set and F be a family of subsets of $\mathcal{U}$. The MSCP is to find a subset $C \subseteq F$ with minimal cardinality $|C|$ such that the union of subsets in C is U. The MSCP can be formulated as an integer programming problem:

$$\begin{aligned} &\text{Minimize} && \sum_{j \in N} x_j \\ &\text{Subject to:} \\ & && \sum_{j \in N_i} x_j \geq 1 \quad \forall i \in M \\ & && x_j \in \{0,1\} \quad \forall j \in N \end{aligned} \tag{3.9}$$

Given an integer K, the MSCP decision problem can be formulated as

$$\begin{aligned} &\text{Maximize} && 0 \\ &\text{Subject to:} \\ & && \sum_{j \in N} x_j \leq K \\ & && \sum_{j \in N_i} x_j \geq 1 \quad \forall i \in M \\ & && x_j \in \{0,1\} \quad \forall j \in N \end{aligned} \tag{3.10}$$

Theorem 3.2.3. *Given a solution $z \in \{0,1\}^n$ to CCP, obtaining the corresponding feasible solution x satisfying (3.8) is NP-hard.*

Proof. Given z, let $M' = \{i : z_i = 1\}$. Then finding a corresponding x of CCP becomes a

feasibility problem which can be formulated as follows:

$$\begin{aligned} &\text{Maximize} && 0 \\ &\text{Subject to:} && \\ & && \sum_{j \in N_i} a_j x_j \leq b_i \quad i \in M \\ & && \sum_{j \in N_i} x_j \geq 1 \quad i \in M' \\ & && x_j \in \{0,1\} \quad \forall j \in N \end{aligned} \tag{3.11}$$

We can see that the formulation of MSCP decision problem (3.10) is a special case of (3.11). The result follows from the NP-completeness of MSCP decision problem. □

Theorem 3.2.4. *BPSP on a path graph (BPSP-P) with $a_j = a_0$ for all $j \in N$ and $w_i > 0$ for $i \in M$ is solvable in polynomial time.*

Proof. Consider the integer programming formulation BPSP1. As we mentioned earlier, when $w > 0$, constraints $\sum_{j \in N_i} x_j \leq |N_i| z_i$, for $i \in M$ are redundant due to the fact that to maximize wz, z_i is driven to be 1 for $\sum_{j \in N_i} x_j \geq 1$. Because of the uniform path capacity $a_j = a_0$ for all $j \in N$, we can transform the constraint $Ax \leq b$ into $A'x \leq \frac{1}{a_0} b$ by dividing both sides of the constraints by a_0. Since the left hand side of A' is integer and $x \in \{0,1\}^n$, $A'x \leq \frac{1}{a_0} b$ is equivalent to $A'x \leq \left\lfloor \frac{b_i}{a_0} \right\rfloor$. Therefore the formulation of BPSP-P simplifies to

$$\text{Maximize} \qquad wz \tag{3.12}$$

$$\text{Subject to:}$$

$$\begin{pmatrix} A' & 0 \\ -A' & I \end{pmatrix} \begin{pmatrix} x \\ z \end{pmatrix} \leq \begin{pmatrix} \mathbf{0} \\ b' \end{pmatrix} \tag{3.13}$$

$$x \in \{0,1\}^n, z \in \{0,1\}^m \tag{3.14}$$

When the underlying graph is a path, all the edges in a path are consecutive, which together with the fact that A' is a 0-1 matrix, we have 1's in each column of A' are consecutive. Therefore, A' is totally unimodular and the coefficient matrix of constraints (3.13)

$$\begin{pmatrix} A' & 0 \\ -A' & I \end{pmatrix}$$

is totally unimodular as well. By solving this integer programming, BPSP-P satisfying given conditions can be solved in polynomial time. □

A BPSP where the coefficient matrix A of its BPSP1 formulation is 1-knot is called 1-knot BPSP (1-KBPSP). Such a problem can be formulated as

$$\begin{aligned}
\text{Maximize:} \quad & \sum_{i \in M} w_i z_i \\
\text{Subject to:} \quad & \sum_{j=l_{i-1}}^{l_i} a_j x_j \leq b_i \quad i \in M \\
& z_i \leq \sum_{j=l_{i-1}}^{l_i} x_j \leq |N_i| z_i \quad i \in M \\
& x_j \in \{0,1\} \quad j \in N \\
& z_i \in \{0,1\} \quad i \in M.
\end{aligned}$$

where $l_0 = 1$.

Consider the problem $P(k)$ constitutes the first k constraints and l_k variables

$$\begin{aligned}
P(k): \quad \text{Maximize:} \quad & \sum_{i=1}^{k} w_i z_i \\
\text{Subject to:} \quad & \sum_{j=l_{i-1}}^{l_i} a_j x_j \leq b_i \quad i = 1, 2, \ldots, k \\
& z_i \leq \sum_{j=l_{i-1}}^{l_i} x_j \leq |N_i| z_i \quad i = 1, 2, \ldots, k \\
& x_j \in \{0,1\} \quad j = 1, 2, \ldots, l_k \\
& z_i \in \{0,1\} \qquad i = 1, 2, \ldots, k.
\end{aligned}$$

Let $P(k|0)$ be the restriction of $P(k)$ with the additional constraint that $x_{l_k} = 0$. Similarly, let $P(k|1)$ be the restriction of $P(k)$ with the additional constraint that $x_{l_k} = 1$. Let $V(k|0)$ and $V(k|1)$ be the optimal objective function value of $P(k|0)$ and $P(k|1)$ respectively.

Consider the subproblems for $p, q \in \{0, 1\}$:

$$
\begin{aligned}
SP(k|p|q) \quad \text{Maximize:} \quad & w_{k+1} z_{k+1} \\
\text{Subject to:} \quad & \sum_{j=l_k}^{l_{k+1}} a_j x_j \leq b_{k+1} \\
& z_{k+1} \leq \sum_{j=l_k}^{l_{k+1}} x_j \leq |N_k| z_{k+1} \\
& x_{l_k} = p, x_{l_{k+1}} = q \\
& x_j \in \{0, 1\} \quad j = l_k, l_k + 2, \ldots, l_{k+1}.
\end{aligned}
$$

Let $W(k|p|q)$ be the optimal objective function values of $SP(k|p|q)$ for $p, q \in \{0, 1\}$ respectively. In fact, we have

$$
W(k|p|q) = \begin{cases}
-\infty & \text{if } SP(k|p|q) \text{ is infeasible} \\
w_{k+1} & \text{if } p = 1, \text{ or } q = 1 \text{ and } SP(k|p|q) \text{ is feasible} \\
0 & \text{if } p + q = 0 \text{ and } SP(k|p|q) \text{ is feasible, } a_j > b_{k+1}, \forall j \in N_j^{SP} \\
w_{k+1} & \text{if } p + q = 0 \text{ and } SP(k|p|q) \text{ is feasible, } \exists a_j \leq b_{k+1}, j \in N_k^{SP}
\end{cases}
$$

where $N_k^{SP} = \{j = 1 + l_k, \ldots, l_{k+1} - 1\}$. Note that by checking whether inequality $pa_{l_k} + qa_{a_{l_{k+1}}} \leq b_{k+1}$ holds or not, we obtain the feasibility of $SP(k|p|q)$. Hence $SP(k|p|q)$ can be solved in polynomial time. Then

$$V(k+1|0) = \max\{V(k|0) + W(k|0|0), V(k|1) + W(k|1|0)\} \tag{3.15}$$

$$V(k+1|1) = \max\{V(k|0) + W(k|0|1), V(k,1) + W(k|1|1)\} \tag{3.16}$$

The optimal objective function value is given by

$$\max\{V(m|0), V(m|1)\}. \tag{3.17}$$

Using the recurrence relations (3.15) and (3.16) and equation (3.17), by backtracking using the corresponding solutions of the knapsack problems, an optimal solution to 1-KBPSP can be constructed.

Theorem 3.2.5. *1-KBPSP can be solved in $O(L + m)$ time.*

Proof. There are at most m subproblems $SP(k|\cdot|\cdot)$ are constructed and they can be solved in polynomial time $O(|N_k|)$. Hence the complexity of solving subproblems $SP(k|\cdot|\cdot)$ is $O(L)$. Given values $W(k|\cdot|\cdot)$, updating $V(k|\cdot)$ costs $O(m)$ time in total. Hence the complexity of solving 1-KBPSP is $O(L+m)$. □

Extending the 1-KBPSP, a BPSP is called r-knot BPSP (r-KBPSP) if the constraint coefficient matrix A of its BPSP1 formulation is r-knot. It can be formulated as follows:

$$\begin{aligned}
\text{Maximize:} \quad & \sum_{i\in M} w_i z_i \\
\text{Subject to:} \quad & \sum_{j=l_{i-1}-r+1}^{l_i} a_j x_j \le b_i \quad i \in M \\
& z_i \le \sum_{j=l_{i-1}-r+1}^{l_i} x_j \le |N_i| z_i \quad i \in M \\
& x_j \in \{0,1\} \quad j \in N.
\end{aligned}$$

where $l_0 = r$, $l_m = n$.

Theorem 3.2.6. *r-KBPSP can be solved in polynomial time for $r = O(\log n)$.*

Proof. For r-KBPSP, let $P(k)$ be its subproblem constituting the first l_k variables:

$$\begin{aligned}
\text{Maximize:} \quad & \sum_{i=l_1}^{l_k} w_i z_i \\
\text{Subject to:} \quad & \sum_{j=l_{i-1}-r+1}^{l_i} a_j x_j \le b_i \quad i = 1,2,\ldots,l_k \\
& z_i \le \sum_{j=l_{i-1}-r+1}^{l_i} x_j \le |N_i| z_i \quad i = 1,2,\ldots,l_k \\
& x_j \in \{0,1\} \quad i = 1,2,\ldots,l_k.
\end{aligned}$$

Applying the similar idea of 1-KBPSP, for any $\eta \in \{0,1\}^r$ we construct $P(k|\eta)$ as the restriction of $P(k)$ with additional constraint that knot variables $\{x_j\}_{l_k-r+1}^{l_k}$ are fixed as η, i.e., $x_{l_k-i} = \eta_i$. Let $V(k|\eta)$ be the optimal objective function value of $P(k|\eta)$. For any

$y, \eta \in \{0,1\}^r$, consider subproblems $SP(k|y|\eta)$ defined as

$$
\begin{aligned}
SP(k|y|\eta) \quad \text{Maximize:} \quad & w_{k+1} z_{k+1} \\
\text{Subject to:} \quad & \sum_{l_k}^{l_{k+1}} a_j x_j \leq b_{k+1} \\
& z_{k+1} \leq \sum_{l_k}^{l_{k+1}} x_j \leq |N_{k+1}| z_{k+1} \\
& x_{l_{k+1}-j+1} = \eta_j \quad \forall j = 1, \ldots r \\
& x_{l_k-j+1} = y_j \quad \forall j = 1, \ldots r \\
& x_j \in \{0,1\} \quad j = l_k - r + 1, l_k - r + 2, \ldots, l_{k+1}.
\end{aligned}
$$

Let $W(k|y|\eta)$ be the optimal objective function values of $SP(k|y|\eta)$. Then the recurrence relation becomes

$$V(k+1|\eta) = \max_{y \in \{0,1\}^r} \{V(k|y) + W(k|y|\eta)\} \tag{3.18}$$

The optimal objective function value is given by

$$\max_{\eta \in \{0,1\}^r} V(m|\eta) \tag{3.19}$$

$SP(k|y|\eta)$ can be solved in polynomial time and the optimal value $W(k|y|\eta)$ is calculated as

$$
W(k|y|\eta) = \begin{cases}
-\infty & \text{if } SP(k|y|\eta) \text{ is infeasible} \\
w_{k+1} & \text{if } |\eta| + |y| \geq 1, SP(k|y|\eta) \text{ is feasible} \\
0 & \text{if } |\eta| + |y| = 0, SP(k|y|\eta) \text{ is feasible, } a_j > b_{k+1}, \forall j \in N_k^{SP} \\
w_{k+1} & \text{otherwise}
\end{cases} \tag{3.20}
$$

where $N_j^{SP} = \{1 + l_k, \ldots, l_{k+1}\}$. Hence r-KBPSP can be solved by using (3.20) with recurrence relation defined by (3.18) and equation (3.19). The optimal solution can be constructed by backtracking the corresponding solutions of subproblems.

For each k, there are 2^{2r} subproblems $SP(k|\cdot|\cdot)$ generated during the process, and $SP(k|\cdot|\cdot)$ can be solved in $O(|N_k|)$ time. Given values $W(k|\cdot|\cdot)$, the update of values $V(k|\cdot)$ take $O(m2^{2r})$ time in total. Hence r-KBPSP can be solved in $O(m2^{2r} + 2^{2r}L)$ time, which is polynomial for $r = O(\log n)$. □

3.3 Heuristic algorithms for BPSP

In this section, we develop several heuristic algorithms for BPSP and present the algorithm performance analysis based on experimental results. The heuristic algorithms we studied are greedy type algorithms including greedy algorithm, semi-greedy algorithm and local search type algorithm with multi-start.

3.3.1 Greedy algorithm

The greedy algorithm of the BPSP starts from the trivial solution $x = \mathbf{0}$ and considers to select variables one by one in an order determined by a prescribed utility ratio. Before the algorithm terminates, the continuously updated solution x is called partial solution. Let x be the current partial solution, the utility ratio r_j, $j \in N$ is defined as

$$r_j = \sum_{i \in M_j} (1 - \max_{k \in N_i} \{x_k\}) w_i \quad \forall j \in N \tag{3.21}$$

which is interpreted as the total cost of the unselected edges that the path P_j contains. The utility ratio depends on the partial solution hence is updated during the greedy selection procedure. In each iteration, the variable with the greatest utility ratio is considered and is set to 1 if the feasibility is not violated, which is followed by utility ratio update before the next iteration. A formal description of the greedy algorithm of BPSP is given below:

Algorithm 4: Greedy Algorithm for BPSP

```
Initialize x ← 0;
for j ← 1 to n do                                        /* initialize r */
    r_j = Σ_{i∈M_j} w_i;
π ← the permutation of N with decreasing value r_j;
for j ← 1 to n do
    add ← π(j);
    if r_add = 0 then                  /* return x if no possible improvement */
        return x;
    if (x|x_add = 1) is feasible then
        x_add = 1;
        Update r as (3.21) with updated x;
        Reorder π(j + 1) to π(n) according to r;
return x
```

The initialization of the utility ratio takes $O(L)$ time. For each x_j, the feasibility checking costs $O(M_j)$ and the total feasibility checking complexity is $O(L)$. Since cost w_i can be newly added once and the update of utility ratio takes $O(|N_i|)$ time, hence the total time cost for ratio update is up to $O(L)$. With reordering in each iteration, the complexity of the greedy algorithm for BPSP is $O(3L + \sum_{j=1}^{n} j \log j)$.

3.3.2 Semi-greedy algorithm

In semi-greedy algorithm, we also employ greedy selection procedure. However, in each iteration we sequentially consider a variable randomly chosen from a candidate list which consists of variables with "good enough" utility ratio values. The underlying utility ratio is as the same as the one used in greedy algorithm defined as (3.21). This randomness allow us to repeat the greedy selection procedures and the best found solution is returned. A formal description of the semi-greedy algorithm is given below:

Algorithm 5: Semi-greedy Algorithm for BPSP

Input: $lCand$, $iLim$, permutation π determined by utility ratio r

Initialize: $x^* \leftarrow \mathbf{0}$, $iter \leftarrow 1$;

repeat

  $next \leftarrow lCand + 1$, $x \leftarrow \mathbf{0}$, candidate list $C \leftarrow \{\pi(1), \pi(2), \ldots, \pi(lCand)\}$;

  while $C \neq \emptyset$ **do**

    $j \leftarrow$ select a random element in C ; `/* random selection */`

    if $(x|x_j = 1)$ *is feasible* **then**

      $x_j = 1$;

      Update utility ratio r and reorder $\pi(j+1)$ to $\pi(n)$;

    if $next \leq n$ **then** `/* update` C `*/`

      $C \leftarrow C \cup \{\pi(next)\} \setminus \{j\}$;

      $next \leftarrow next + 1$;

    else

      $C \leftarrow C \setminus \{j\}$;

  if $f(x) > f(x^*)$ **then** `/* update best known solution */`

    $x^* \leftarrow x$;

  $iter \leftarrow iter + 1$;

until $iter = iLim$;

return x^*

The parameter $lCand$ determines the length of candidate list and the iteration limit $iLim$ is the number of iterations that we repeat the greedy selection procedure. We set $lCand$ as a constant which is independent from instance size so thats the complexity of the semi-greedy algorithm is the same as the greedy algorithm times $iLim$: $O(iLim(3L + \sum_{j=1}^{n} j \log j))$.

3.3.3 Multi-start local search algorithm

We develop a multi-start local search algorithm based on 2-swap neighborhood for BPSP. The algorithm starts from an initial solution generated by semi-greedy algorithm, and applies first-improving local search based on 2-swap neighborhood until a local optimum is reached. These two steps form one iteration. We perform $iLim$ iterations where $iLim$ is a prescribed parameter and return the best known solution found in the whole procedure. Due

to the diversity of initial solution obtained from semi-greedy algorithm, different solution spaces are expected to be explored. Let $g(\cdot)$ be the objective function. A formal description of the algorithm is given below:

Algorithm 6: Multi-start local search algorithm for BPSP

```
Input: iteration limit iLim
Initialize: x* ← 0;
for iter ← 1 to iLim do
    x ← obtained from semi-greedy algorithm;
    improve ← true;
    while improve do
        improve = false;
        for add ∈ J0(x) and !improve do
            for drop ∈ J1(x) and !improve do
                if g(x|x_add = 1, x_drop = 0) > g(x) and (x|x_add = 1, x_drop = 0) is feasible
                then
                    x_add = 1, x_drop = 0;
                    improve ← true;
    if g(x) > g(x*) then
        x* ← x;
return x*.
```

We also implemented a 1-flip local search after the 2-swap local search procedure. First we scan all the variables in $\{j \in N : x_j = 1\}$ and set $x_j = 0$ if it does not affect the objective function value. Then we scan all the variable in $\{j \in N : x_j = 0\}$ and perform greedy selection if the feasibility is not violated. However, the experiments show that although there are always redundant variables can be removed, no feasible adding move is available to improve the solution. Hence we do not include that procedure.

3.4 Computational results and analysis

In this section, experimental results are presented to analyze the performance of above developed heuristic algorithms for BPSP.

In order to draw reasonable conclusion, we first develop the instance generator to generate representative test bed. Due to the equivalent relation between BPSP and CCP established in Theorem 3.1.1, we generate CCP instance instead of BPSP instance. The instance generator is based on that of GPSP for PSP inspired instance. In fact, except for objective function definition, they contain exactly the same data including constraint coefficient matrix A, edge capacity vector b, edge cost vector w and path capacity usage vector a. Hence the same instance generator parameters are used. We analyze the above parameters' affect to the instance solvability from the solving results of CPLEX which is reported in Table 3.1. The time limit for CPLEX is set as 1 minute.

Parameter	Value	AveOptTime	AveOptNum(%)	Total
n	1000	8.42	91	108
m/n	0.50	0.64	100	36
	1.00	1.19	100	36
	5.00	23.44	72	36
α	0.25	5.52	75	36
	0.50	9.39	100	36
	0.75	10.36	97	36
rU	0.10	4.53	89	36
	0.20	6.42	89	36
	0.50	14.32	94	36
U^{au}	0.05	9.18	91	54
N^{au}	0.05	7.66	91	54

Table 3.1: BPSP instance generator parameters comparison

We can see that CPLEX is very efficient at solving BPSP. In fact, most of the testing instances can be solved within one minute. Regarding ratio $\kappa = m/n$, we test three values 0.5, 1 and 5. When m/n is as small as 0.5 or 1, all the generated instances are solved optimally within one minute. However, when m/n increases to 5, we have 72% instances optimally solved within given time limit. Hence a larger constraints size produces harder to solve instance. Regarding α, we test three values 0.25, 0.50 and 0.75. The optimally

solved instance percentages are 75%, 100% and 97% respectively. This is completely the opposite for GPSP for which $\alpha = 0.50$ produce the hardest instance. We believe that the larger α is, the easier to find a feasible x such that the objective function values reaches its upper bound $\sum_{i \in M} w_i$. Hence $\alpha = 0.25$ gives hardest instance. Regarding rU, which determinates upper bound of nonzero entries in each constraint, we test three values 0.1, 0.2 and 0.5. The latter two values have the same percentage of optimally solved instance 89%, which is smaller than 94% of $rU = 0.1$. This result is in accordance with our expectation. A larger rU leads to a denser coefficient matrix A, hence a larger chance that an edge is selected.

Based on above analysis on instance generator parameters, we see that BPSP is easy for CPLEX to solve. Hence we choose our parameter as follows to generate relatively hard to solve instance: $n(1000)$, $m(5000)$, $\alpha(0.25)$, $rU(0.2)$, $U^{au}(0.05)$, $cU(0.05)$. All the instances are generated with the same set of instance generator parameters with different seed. The solving results of the chosen test bed instances are summarized in Table 3.2, all the instances are optimally solved by CPLEX within 6 minutes.

ID	m	n	α	rU	U^{au}	OPT	Time(s)	UBD(LP relax)
1	5000	1000	0.25	0.2	0.05	671139	165.830	671145
2	5000	1000	0.25	0.2	0.05	681342	281.465	681364
3	5000	1000	0.25	0.2	0.05	675577	324.710	675588
4	5000	1000	0.25	0.2	0.05	674118	334.192	674158
5	5000	1000	0.25	0.2	0.05	668746	108.204	668746
6	5000	1000	0.25	0.2	0.05	669418	336.606	669463
7	5000	1000	0.25	0.2	0.05	676430	118.983	676430
8	5000	1000	0.25	0.2	0.05	674522	294.641	674522
9	5000	1000	0.25	0.2	0.05	673592	341.660	673592
10	5000	1000	0.25	0.2	0.05	683176	213.536	683176

Table 3.2: BPSP - the results of CPLEX on test bed

Greedy algorithm and semi-greedy algorithm with different values of candidate list length $lCand$ are tested and the results are reported in Table 3.3. For semi-greedy algorithm, the iteration limit is fixed at 50 and $lCand$ varies from 5 to 20. First we observe that C-PLEX solved all the instances optimally with average computational time around 4 minutes.

Greedy algorithm solved all the instances in negligible time and the average deviation from the optimal value is 0.33%. Semi-greedy algorithm solved all the instances with average deviation within 0.19% in average time less than 4 seconds. Comparing to CPLEX, greedy type algorithm costs 80 times less computational time with good quality solutions. Compare semi-greedy algorithm with greedy algorithm, its average deviation is nearly half of the latter. Although semi-greedy algorithm costs more time, it is still very efficient. Comparing different candidate list length $lCand$, it seems that $lCand = 10$ provides the average best performance. However, from Figure 3.2 we can see that a supreme $lCand$ value for all the instances is not possible.

	Average		Max		Min		
Alg	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time	#Opt Inst
CPLEX	0.00	251.983	0.00	341.66	0.00	108.204	10
GREEDY	0.33	0.003	0.40	**0.02**	0.26	**0.000**	**0**
SEMIGREEDY-5-50	0.17	3.381	0.25	3.70	**0.07**	3.292	**0**
SEMIGREEDY-10-50	**0.16**	3.371	**0.24**	3.63	0.12	3.322	**0**
SEMIGREEDY-15-50	0.19	3.371	0.29	3.60	0.09	3.307	**0**
SEMIGREEDY-20-50	0.18	3.367	0.33	3.65	0.10	3.307	**0**

Table 3.3: BPSP semi-greedy algorithm parameter comparison: $lCand$

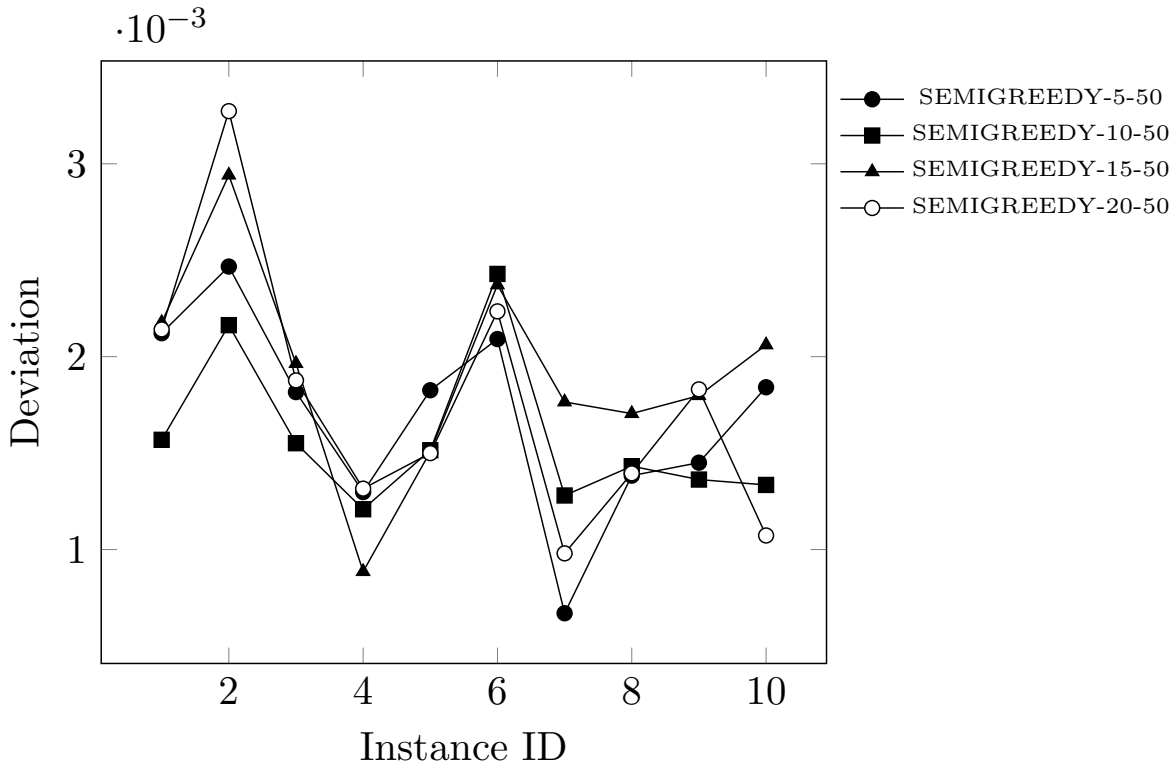


Figure 3.2: BPSP semi-greedy algorithm parameter comparison: $lCand$

Multi-start local search algorithm with different iteration limits $iLim$ are tested on BPSP test bed and the results are reported in Table 3.4. The iteration limit $iLim$ ranging from 10 to 50. The larger $iLim$ is, the more likely the algorithm return higher quality solutions but with a larger time cost. This is also proven by the numerical results that $iLim = 50$ provides lowest average deviation 0.05% and longest computational time. The solution quality gets improved comparing to greedy type algorithms. However, the computational time increases dramatically that it takes longer time than CPLEX without returning any optimal solution. Regarding different values of $iLim$, as it increase, the average deviation decreases and the computational time increases. Overall $iLim = 50$ provides the lowest average deviation, whereas $iLim = 10$ provides same level quality solution with a much less computational time.

	Average		Max		Min		
Alg	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time	#Opt Inst
CPLEX	0.00	251.983	0.00	341.66	0.00	108.204	10
MULLS-10	0.08	**77.626**	0.12	**91.85**	0.05	**62.697**	**0**
MULLS-20	0.07	166.228	0.15	205.55	0.02	139.103	**0**
MULLS-30	0.06	247.658	**0.11**	295.60	0.00	197.537	**0**
MULLS-40	0.06	323.823	**0.11**	391.08	0.02	257.460	**0**
MULLS-50	**0.05**	406.859	**0.11**	484.62	0.01	348.697	**0**

Table 3.4: BPSP multi-start local search algorithm comparison: $iLim$

A summary results of the deviation and computational time for all the algorithms on BPSP Test bed is given in Table 3.5. First, we observe that CPLEX solved all the testing instances optimally with average time 251 seconds, while no heuristic algorithm solved any instance optimally. However, semi-greedy algorithm provided solutions in less than 4 seconds with average deviation 0.16%.

	Average		Max		Min		
Alg	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time	# Opt Inst
CPLEX	0.00	251.983	0.00	341.66	0.00	108.204	10
GREEDY	0.33	0.003	0.40	**0.02**	0.26	**0.000**	**0**
SEMIGREEDY	0.16	3.371	0.24	3.63	0.12	3.322	**0**
MULLS	**0.05**	406.859	**0.11**	484.62	0.01	348.697	**0**

Table 3.5: BPSP - the summary results of algorithms on test bed

3.5 A variation of BPSP

BPSP only considers to select as many edges as possible. However, in practice, enlarging the network, i.e., selecting more paths, is also a goal for the company. Hence, given two sets of paths with the same selected edges, the one with more paths is preferred to the other. Then a variation of BPSP is to find the lexicographic maximum of $(\sum_{i\in M} w_i(\max_{j\in N_i} x_j), \sum_{j\in N} x_j)$.

This problem can be formulated by modifying the objective function of BPSP as

$$\text{Maximize} \quad \Theta \sum_{i \in M} w_i \max_{j \in N_i} \{x_j\} + \sum_{j \in N} x_j$$

Subject to:

$$\sum_{j \in N_i} a_j x_j \leq b_i \quad i \in M$$

$$x_j \in \{0, 1\} \quad \forall j \in N$$

where Θ is a large enough positive number. This variation of BPSP select the path set with maximum cardinality among all the optimal solutions of original BPSP.

Chapter 4

Discounted PSP

In the previous chapters, we studied two models, GPSP and BPSP, for the Path Selection Problem. Each model selects a class of paths that are interesting for the company to make planing decision in addition to the immediate objective of providing quote for one or more projects. The value of the quote need not be the proposed cost identified by the model and it may depend on various "what if" kind of considerations. Offering discount on the nominal cost is one such possibility. To provide a quote for a potential customer, the company have additional information from the set of paths selected under the GPSP or BPSP models. Using this information, the company can determine possible number of the future clients that may use a particular edge. A discount of the construction of such an edge can be calculated based on this information. Such a model for offering discounts is called *discount after selection*. Discount after selection is an available approach. We now discuss another model where path selection consider discounted costs.

4.1 "A priori" discounted PSP model

Recall that the objective function of GPSP

$$f(x) = \sum_{j \in N} c_j x_j = \sum_{j \in N} (\sum_{i \in M_j} w_i) x_j = \sum_{i \in M} w_i (\sum_{j \in N_i} x_j),$$

and the objective function of BPSP

$$g(x) = \sum_{i \in M} w_i (\max_{j \in N_i} \{x_j\}).$$

In GPSP model each client is charged the full cost of an edge, while in BPSP model all the clients using an edge share the cost of such edges. These models are useful when discount after selection is considered as explained earlier. An additional tool to perform appropriate "what if" analysis is to optimize selected paths where discounts are applied "a priori".

We introduce a *discounted edge cost function* as follows: For each $i \in M$, let ϕ_i be a real valued function of $\sum_{j \in N_i} x_j$, i.e., $\phi_i : \{1, 2, \ldots, |N_i|\} \to R^+ \cup \{0\}$. Then the *"a priori" discounted Path Selection Problem* (DPSP) is defined as

$$\begin{aligned} &\text{Maximize} \qquad h(x) = \sum_{i \in M} \phi_i(\alpha_i) \\ &\text{Subject to:} \\ &\qquad \sum_{j \in N_i} a_j x_j \leq b_i \quad i \in M \\ &\qquad x_j \in \{0, 1\} \quad \forall j \in N \end{aligned} \tag{4.1}$$

where $\alpha_i = \sum_{j \in N_i} x_j$. DPSP reduces to GPSP when $\phi_i(\alpha_i) = \phi_i^g(\alpha_i) := w_i \alpha_i$, and it reduces to BPSP when $\phi_i(\alpha_i) = \phi_i^b(\alpha_i) := w_i \min\{\alpha_i, 1\}$. By definition, $\phi_i(\alpha_i)$ represents the total cost of edge e_i while $\frac{1}{\alpha_i}\phi_i(\alpha_i)$ is the partial cost shared by each path that uses this edge. To offer a discount, $\phi_i(\cdot)$ should satisfy following conditions:

$$\phi_i(0) = 0 \tag{4.2}$$

$$\phi_i(\alpha) \leq \phi_i(\beta) \quad \forall\ 0 \leq \alpha \leq \beta \tag{4.3}$$

$$\frac{1}{\alpha}\phi_i(\alpha) \geq \frac{1}{\beta}\phi_i(\beta) \quad \forall\ 0 < \alpha \leq \beta \tag{4.4}$$

$$w_i \leq \phi_i(\alpha) \leq \alpha w_i \quad \alpha > 0 \tag{4.5}$$

By condition (4.2), the incurred edge cost is zero when no selected paths containing it. Condition (4.3) guarantees that the total cost brought by edge e_i is nondecreasing when the selected paths number increases. However, condition (4.4) implies that $\frac{1}{\alpha}\phi_i(\alpha)$ is a nonincreasing function so that the individual share of each edge cost is nonincreasing as the number of selected paths that share this edge increases. Finally, the total cost of each edge is between that of BPSP and GPSP. Hence DPSP provides a trade-off between BPSP and GPSP.

Regarding the choice of $\phi_i(\cdot)$, we first point out following observation:

Lemma 4.1.1. *If $\phi_i(\alpha) = k_i \alpha w_i$ where $k_i \in (0, 1)$ for each $i \in M$, DPSP reduces to GPSP with modified edge costs.*

Proof. If $\phi_i(\alpha_i) = k_i\alpha_i w_i$, then the objective function of DPSP becomes

$$\sum_{i\in M}\phi_i(\alpha_i) = \sum_{i\in M} w_i * k_i(\sum_{j\in N_i} x_j) = \sum_{i\in M}(k_i w_i) * (\sum_{j\in N_i} x_j)$$

which is as the same as the objective function of GPSP with edge cost $w'_i = k_i w_i$. The result follows. □

We consider two types of discounted edge cost function $\phi_i(\alpha)$: concave function and convex function. For each type, three functions are studied:

$$\begin{aligned}
\phi_i^1(\alpha) &= \sqrt{\alpha}\cdot w_i \\
\phi_i^2(\alpha) &= 0.5(\alpha+\sqrt{\alpha})\cdot w_i \\
\phi_i^3(\alpha) &= (2\sqrt{\alpha}-1)\cdot w_i \\
\phi_i^4(\alpha) &= (\alpha-\sqrt{\alpha}+1)\cdot w_i \\
\phi_i^5(\alpha) &= (\alpha-\sqrt{\alpha}+1-1.5\log\alpha)\cdot w_i \\
\phi_i^6(\alpha) &= (\alpha-\alpha^{0.7}+1-\log\alpha)\cdot w_i
\end{aligned}$$

We define $\phi_i^k(0) = 0$ for $k = 1, 2, \ldots, 6$. The following Figure 4.1 and 4.2 illustrates the differences of these discounted edge cost function. All the functions lies in between $\phi_i^b(\cdot)$ and $\phi_i^g(\cdot)$.

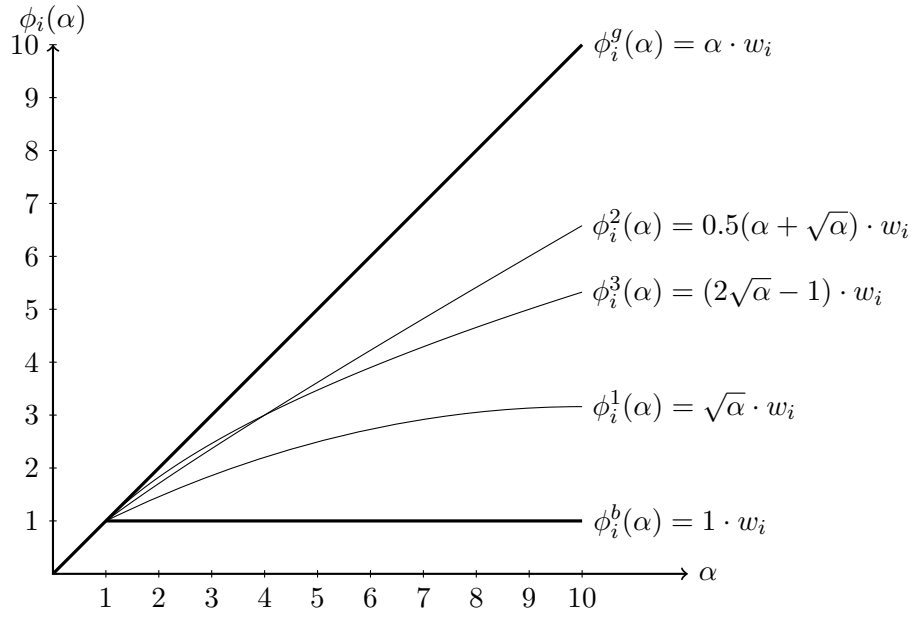


Figure 4.1: Discounted edge cost function: concave

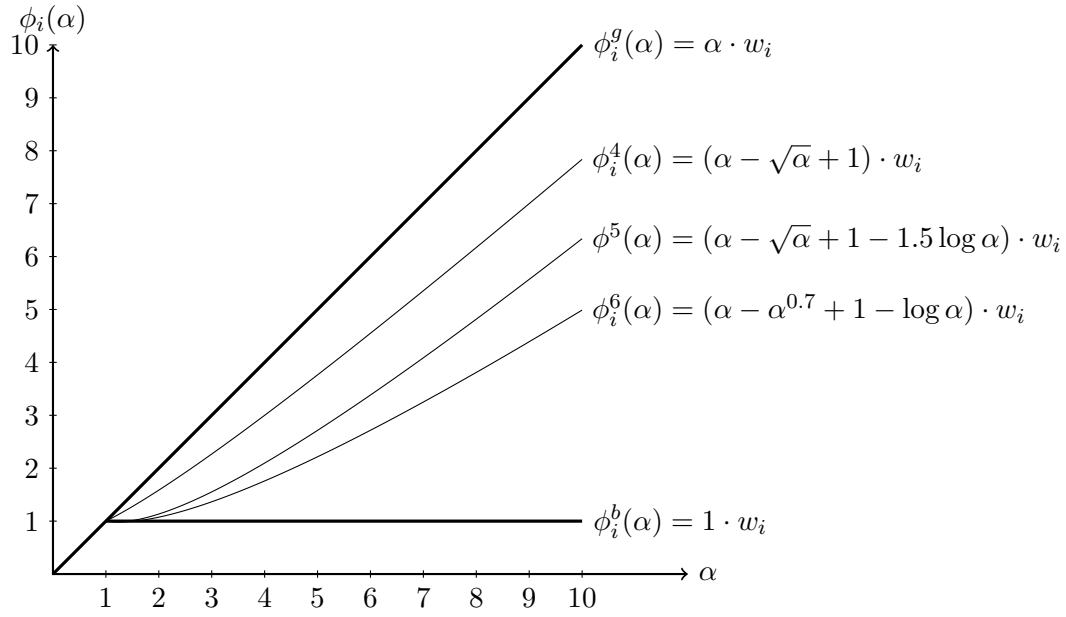


Figure 4.2: Discounted edge cost function: convex

We use CPLEX to solve the DPSP instances with the same data but different choices

of discounted edge cost function $\phi_i(\alpha)$. (The details of DPSP instance generator is given in Section 4.3). Given 0.5 hour as time limit, the solving results of CPLEX are presented in Table 4.1. We choose different instance sizes with $n = 500, 1000$ and various $m/n = 0.5, 1$ accordingly which is given in column (m, n). In column ϕ the corresponding superscript of discounted edge cost functions are given, where k denotes the function $\phi_i^k(\cdot)$. The column $|x|$ shows the number of selected paths in the returned best known solution. The best known objective function value is reported in column OPT if it's optimal and in column LBD otherwise. At last, column Time and LPR give the computational time and the optimal objective function value of its LP relaxation problem respectively.

We first notice that only BPSP instance can be optimally solved by CPLEX in half an hour. Also the objective function values of the best known solutions of DPSP are always between that of BPSP and GPSP. Second, the best known objective function value varies due to the discount and they follow the same order with that of the edge discounted cost function. However, the number of chosen paths are approximately the same or at least on the same level. This implies that different discount function options may mainly affect the objective function value but have little affect to the selected path number. Third, even for instance size $(1000, 200)$ CPLEX fails to return any feasible solution. We also run the same experiments for $n = 2000$, m varies from 300 to 1200 and no feasible solution is returned. This fact demonstrates the necessity to develop heuristic algorithms for DPSP.

Index	(m, n)	ϕ	$\|x\|$	OPT	LBD	Time/s	UBD(LP relax)
1	(250,500)	g	334	-	333442.000	1806.199	339583.231
2	(250,500)	b	106	3865.000	-	163.725	3886.402
3	(250,500)	1	333	-	34176.976	1913.730	34580.258
4	(250,500)	2	331	-	182474.142	1815.594	187077.365
5	(250,500)	3	332	-	64370.924	1814.268	65275.296
6	(250,500)	4	328	-	297370.468	1807.115	313752.296
7	(250,500)	5	322	-	282706.664	1807.086	305840.661
8	(250,500)	6	326	-	239988.933	1812.620	260892.017
9	(500,500)	g	325	-	647076.000	1802.309	667129.708
10	(500,500)	b	101	7594.000	-	1063.644	7618.261
11	(500,500)	1	-	-	0.000	1842.290	67723.367
12	(500,500)	2	-	-	0.000	1900.250	367421.747
13	(500,500)	3	319	-	124421.354	1838.551	127829.449
14	(500,500)	4	316	-	570708.690	1809.469	616898.420
15	(500,500)	5	317	-	551881.813	1822.313	601621.788
16	(500,500)	6	-	-	0.000	1803.959	513601.954
17	(200,1000)	g	684	-	541394.000	1807.115	545859.301
18	(200,1000)	b	568	3004.000	-	19.656	3004.000
19	(200,1000)	1	677	-	38265.715	1820.537	38598.294
20	(200,1000)	2	681	-	288395.535	1923.346	292221.524
21	(200,1000)	3	678	-	73655.991	1808.311	74192.588
22	(200,1000)	4	-	-	0.000	1801.857	515928.951
23	(200,1000)	5	-	-	0.000	1801.173	508757.665
24	(200,1000)	6	-	-	0.000	1801.620	444470.604
25	(500,1000)	g	669	-	1315270.000	1809.315	1334750.199
26	(500,1000)	b	334	7951.000	-	291.112	7968.511
27	(500,1000)	1	-	-	0.000	1807.535	97405.285
28	(500,1000)	2	-	-	0.000	1802.678	716072.506
29	(500,1000)	3	-	-	0.000	1806.514	186842.059
30	(500,1000)	4	-	-	0.000	1801.809	1259993.134
31	(500,1000)	5	-	-	0.000	1814.385	1241632.919
32	(500,1000)	8	-	-	0.000	1822.918	1083515.008

Table 4.1: DPSP discounted edge cost function $\phi_i(\alpha)$ comparison

Note that DPSP is nonlinear due to the discounted edge cost function $\phi_i(\cdot)$. We now introduce an integer linear programming formulation for DPSP. For each $i \in M$, $l = 1, 2, \ldots, |N_i|$, we introduce variable

$$z_i^l = \begin{cases} 1 & \text{if } e_i \text{ is selected by } l \text{ paths,} \\ 0 & \text{otherwise.} \end{cases} \tag{4.6}$$

By definition, $z_i^l = 1$ if and only if $\sum_{j \in N_i} x_j = l$, this relation can be formulated as follows:

$$\sum_{j \in N_i} x_j = \sum_{l=1}^{|N_i|} l z_i^l \quad i \in M \tag{4.7}$$

$$\sum_{l=1}^{|N_i|} z_i^l \leq 1 \quad i \in M \tag{4.8}$$

These two constraints together make sure that there is at most one z_i^l is nonzero, which is the one corresponding to $\sum_{j \in N_i} x_j = l > 0$. We refer this as *edge selection number constraints*.

Since $\phi_i(\cdot)$ is given, the value $\phi_i(l)$ for $l = 1, 2, \ldots, |N_i|$ can be calculated in advance. When $\sum_{j \in N_i} x_j = l > 0$, the cost brought by edge e_i is $\phi_i(l)$. With variable z_i^l, this can be expressed as $\sum_{l=1}^{|N_i|} \phi_i(l) z_i^l$. Hence the objective function is linearized as

$$\sum_{i \in M} \sum_{l=1}^{|N_i|} \phi_i(l) z_i^l \tag{4.9}$$

Then the nonlinear integer programming problem (4.1) is transformed into an integer

linear programming problem:

$$\text{DPSP1:} \quad \text{Maximize} \quad \sum_{i \in M} \sum_{l=1}^{|N_i|} \phi_i(l) z_i^l \tag{4.10}$$

Subject to:

$$\sum_{j \in N_i} a_j x_j \leq b_i \quad i \in M \tag{4.11}$$

$$\sum_{j \in N_i} x_j = \sum_{l=1}^{|N_i|} l z_i^l \quad i \in M \tag{4.12}$$

$$\sum_{l=1}^{|N_i|} z_i^l \leq 1 \quad i \in M \tag{4.13}$$

$$x_j \in \{0,1\} \quad \forall j \in N \tag{4.14}$$

$$z_i^l \in \{0,1\} \quad \forall i \in M, l = 1,2,\ldots,|N_i| \tag{4.15}$$

The NP-hardness of DPSP can be derived from GPSP directly.

Theorem 4.1.1. *DPSP is NP-hard.*

Proof. As we mentioned before, GPSP is a special case of DPSP when the discounted edge cost function is $\phi_i^g(\cdot)$. Due to the NP-hardness of GPSP, the result follows. □

4.2 Heuristic algorithms for DPSP

In this section we develop various heuristic algorithms for DPSP and evaluate their performances based on the computational results. First, widely used heuristic algorithmic ideas are applied to develop algorithms for DPSP. It includes greedy algorithm, semi-greedy algorithm and multi-start local search algorithm. The detailed algorithm description is given in subsections and computational results follow.

Greedy algorithm for DPSP

Greedy algorithm considers the paths one by one in an given order which is determined by an utility ratio. A path under consideration is selected if the feasibility is not violated. A formal description of the greedy algorithm is given below:

Algorithm 7: Greedy Algorithm for DPSP

Input: permutation π based on the decreasing utility ratio value r

Initialize $x \leftarrow \mathbf{0}$;

for $j \leftarrow$ *1 to* n **do**

  if $(x|x_{\pi(j)} = 1)$ *is feasible* **then**

    $x_{\pi(j)} = 1$;

return x

We use the utility ratios summarized in Table 2.1, which are introduced for GPSP. The feasibility checking takes $O(L)$ in total and the calculation of objective function value takes $O(L)$. Together with the variable ordering, the complexity of the greedy algorithm for DPSP is $O(L+\psi_r)$, where ψ_r is the computational complexity to calculate the pseudo utility ratio r and to obtain corresponding permutation.

Semi-greedy algorithm for DPSP

In the semi-greedy algorithm we consider an element randomly picked from a candidate list which constitutes of paths with "good enough" utility ratio values and apply greedy selection procedure. The greedy selection procedure continues until all the paths are scanned. We repeat it several iterations with the best found solution recorded. The parameters candidate list size $lCand$ and iteration limit $iLim$ together determine the semi-greedy algorithm. When $lCand = 1$ and $iLim = 1$, the algorithm reduces to the greedy algorithm. A formal description of the semi-greedy algorithm is given below:

Algorithm 8: Semi-greedy Algorithm for DPSP

Input: $lCand$, $iLim$, permutation π determined by utility ratio r

Initialize: $x^* \leftarrow \mathbf{0}$, $iter \leftarrow 1$;

repeat

  $next \leftarrow lCand + 1$, $x \leftarrow \mathbf{0}$, candidate list $C \leftarrow \{\pi(1), \pi(2), \ldots, \pi(lCand)\}$;

  while $C \neq \emptyset$ **do**

    $j \leftarrow$ select a random element in C ; `/* random selection */`

    if $(x|x_j = 1)$ *is feasible* **then**

      $x_j = 1$;

    if $next \leq n$ **then** `/* update` C `*/`

      $C \leftarrow C \cup \{\pi(next)\} \setminus \{j\}$;

      $next \leftarrow next + 1$;

    else

      $C \leftarrow C \setminus \{j\}$;

  if $h(x) > h(x^*)$ **then** `/* update best known solution */`

    $x^* \leftarrow x$;

  $iter \leftarrow iter + 1$;

until $iter = iLim$;

return x^*

The complexity of semi-greedy algorithm is iteration limit $iLim$ times that of the corresponding greedy algorithm. We choose $lCand$ as an instance independent constant, then the complexity becomes the same.

Multi-start local search algorithm for DPSP

A similar idea of multi-start local search algorithm is also adapted to solve DPSP. The neighborhood of local search is based on 2-swap neighborhood and 1-flip neighborhood. In each iteration, the algorithm starts from a initial solution obtained from semi-greedy algorithm and applies first-improving local search algorithm based on 2-swap neighborhood to explore the neighbors. After reaching a local optimum, a first-improving local search based on 1-flip neighborhood is followed to seek for a saturated solution. We perform $iLim$ iterations where $iLim$ is a prescribed parameter and return the best known solution found

in the whole procedure. To control the running time, we add time limit in this procedure so the computational time is within certain level. A formal description of the algorithm is given below:

Algorithm 9: Multi-start local search algorithm for DPSP

Input: iteration limit $iLim$, time limit $tLim$
Initialize: $x^* \leftarrow \mathbf{0}$, $start \leftarrow$ current time;
for $iter \leftarrow 1$ *to* $iLim$ **do**
  $x \leftarrow$ obtained from semi-greedy algorithm;
  $improve \leftarrow true$; `/* local search 2-swap */`
  while $improve$ **do**
    $improve = false$;
    for $add \in J0(x)$ *and* $!improve$ **do**
      for $drop \in J1(x)$ *and* $!improve$ **do**
        $x' = (x|x_{add} = 1, x_{drop} = 0)$;
        if $h(x') \geq h(x) + 1$ *and* x' *is feasible* **then**
          $x_{add} = 1$, $x_{drop} = 0$;
          $improve \leftarrow true$;
  for $add \in J0(x)$ **do** `/* local search 1-flip */`
    if $(x|x_{add} = 1)$ *is feasible* **then**
      $x_{add} = 1$;
  if $h(x) > h(x^*)$ **then**
    $x^* \leftarrow x$;
  if *current time - start* $\geq tLim$ **then**
    return x^*;
return x^*.

4.3 Computational results and analysis

We present our computational results of developed heuristic algorithms in this section. The instance generator of DPSP is based on that of GPSP for PSP inspired instance. The generator first generate GPSP instances with given parameter, which together with the

discounted edge cost function form DPSP instances.

To form our DPSP test bed, we fix instance generator parameters as follows: $\alpha(0.50)$, $rL(2)$, $rU(0.5n)$, $U^{au}(0.05)$, $aL(1)$, $cL(1)$, $cU(30)$. According to the previous analysis, different functions $\phi_i(\cdot)$ seem to only affect the objective function value. The selected path numbers of the best known solutions are approximately the same or on the same level. Hence we choose $\phi_i^3(\cdot)$ as our discounted edge cost function for all DPSP testing instances. The experiments include two parts: algorithm parameter tuning and algorithm performance evaluating.

Algorithm parameter tuning

The algorithm parameter tuning experiments are conducted on Test bed 1, of which five randomly generated instances with fixed instances size $(n, m) = (1000, 200)$ and fixed other parameters specified as above. For comparison, they are solved by CPLEX given time limit as 10 minutes and the solving results are summarized in Table 4.2. Although no optimal solution is returned within the time limit, the solution quality is high since the objective value is close to that of the LP relaxation.

Index	(m, n)	OPT	LBD	Time/s	UBD(LP relax)
1	(200,1000)	-	73485.848	600.723	74192.588
2	(200,1000)	-	78877.555	600.535	79644.897
3	(200,1000)	-	73153.738	600.530	73992.515
4	(200,1000)	-	76238.198	600.574	76911.462
5	(200,1000)	-	74810.366	600.517	75547.094

Table 4.2: DPSP - the results of CPLEX on Test bed 1

Greedy algorithm with different utility ratios is tested and the results are presented in Table 4.3. The utility Ratio 4 gives the overall best performance regarding to solution quality and computational time. In fact, its average, maximal and minimal deviation are all the lowest among all the greedy algorithm ratios. Also the utility ratios can be divided into two groups by the quality solution. Ratio 1, 2, 4, 5 provide solutions with average deviation between 1.65% and 1.88%, while other ratios provide solution with average deviation between 16.95% and 17.15%.

	Average		Max		Min		
Alg	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time	#Opt Inst
CPLEX	0.98	600.58	1.13	600.72	0.88	600.52	0
GREEDY-1	1.84	0.01	1.99	0.02	1.72	**0.00**	**0**
GREEDY-2	1.88	0.09	1.99	0.09	1.76	0.08	**0**
GREEDY-3	16.95	**0.00**	18.27	0.02	16.14	**0.00**	**0**
GREEDY-4	**1.65**	**0.00**	**1.94**	**0.00**	**1.45**	**0.00**	**0**
GREEDY-5	1.88	**0.00**	1.99	**0.00**	1.76	**0.00**	**0**
GREEDY-6	17.08	**0.00**	17.85	0.01	15.93	**0.00**	**0**
GREEDY-7	16.95	**0.00**	18.27	**0.00**	16.14	**0.00**	**0**
GREEDY-8	17.15	**0.00**	18.04	0.02	16.58	**0.00**	**0**

Table 4.3: DPSP greedy algorithm utility ratio comparison

Semi-greedy algorithm with different candidate list length $lCand$ is tested and the results are presented in Table 4.4. We fix iteration limit $iLim = 50$ and use utility Ratio 4 since it performs better than others. We can see on the same level of computational time, $lCand = 20$ gives overall lowest deviations.

	Average		Max		Min		
Alg	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time	#Opt Inst
CPLEX	0.98	600.58	1.13	600.72	0.88	600.52	0
SEMIGREEDY-5-50	1.49	**0.02**	1.73	**0.03**	1.33	0.02	**0**
SEMIGREEDY-10-50	1.42	**0.02**	1.64	**0.03**	1.32	**0.01**	**0**
SEMIGREEDY-15-50	1.44	**0.02**	1.64	**0.03**	1.33	0.02	**0**
SEMIGREEDY-20-50	**1.39**	**0.02**	**1.56**	**0.03**	**1.22**	**0.01**	**0**
SEMIGREEDY-25-50	**1.39**	**0.02**	**1.56**	**0.03**	1.24	0.02	**0**

Table 4.4: DPSP semi-greedy algorithm parameter comparison: $lCand$

Multi-start local search algorithm with different iteration limit $iLim$ is tested and the results are reported in Table 4.5. The semi-greedy algorithm embedded in it employs utility Ratio 1,2, 4 and 5 randomly with iteration limit as 50 and candidate list length as 20. The iteration limit for multi-start local search algorithm varies from 10 to 100. We see that $iLim = 80$ and 90 give lowest average deviation. However, the computational time is proportional to the iteration limit. Taking this into account, $iLim = 50$ uses shorter time

with an average deviation 1.07 which is close to the best value 1.05 and obtain a lowest minimal deviation among all.

	Average		Max		Min		
Alg	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time	#Opt Inst
CPLEX	0.98	600.58	1.13	600.72	0.88	600.52	0
GREEDY-4	**1.65**	**0.00**	**1.94**	**0.00**	**1.45**	**0.00**	**0**
MULLS-10	1.26	10.83	1.53	12.78	1.06	9.61	**0**
MULLS-20	1.15	21.43	1.27	23.04	1.08	18.65	**0**
MULLS-30	1.11	31.39	1.27	37.94	1.04	28.56	**0**
MULLS-40	1.11	45.05	1.33	51.45	0.99	38.16	**0**
MULLS-50	1.07	54.23	1.19	60.06	**0.93**	49.38	**0**
MULLS-60	1.09	64.62	1.23	72.65	1.01	57.36	**0**
MULLS-70	1.07	73.58	1.18	78.98	0.96	67.74	**0**
MULLS-80	**1.05**	87.07	1.23	95.79	0.95	79.14	**0**
MULLS-90	**1.05**	97.38	**1.12**	104.63	1.01	89.63	**0**
MULLS-100	1.07	108.45	1.22	121.14	0.96	101.35	**0**

Table 4.5: DPSP multi-start local search algorithm parameter comparison: $iLim$

DPSP algorithm performance evaluating

Based on the analysis from algorithm parameter tuning, we test the greedy algorithm with Ratio 4, semi-greedy algorithm with Ratio 4, iteration limit 50 and candidate list length as 20, multi-start local search with iteration limit 50 on the Test bed 2. It consists of DPSP instances with fixed generator parameters specified as above and the instances size n varies from 1000, 2000 and 5000, and m/n ranges from 0.1 to 0.3. In this case, for each instance size 3 instances are generated randomly to eliminate bias. These instances are also solved by CPLEX based heuristic with time limit as 0.5 hour. The algorithms solving results are presented in Table 4.6 and Table 4.7 reporting objective value and deviation respectively.

We first observe that only 9 instances out of 45 have nonzero feasible solution returned by CPLEX in half an hour. For these instances, the deviation of the returned solutions of CPLEX are within 1.09%. However, multi-start local search algorithm outperforms CPLEX on 2 instances out of these 9 instances. Second, comparing greedy algorithm with semi-greedy algorithm, semi-greedy algorithm improves the solution quality with reducing the

deviation around 0.3% at a cost of longer computational time. Comparing semi-greedy algorithm with multi-start local search algorithm, the latter further provides better quality solution with an average 0.3% reduced deviation but with a larger computational time. At last, column BT gives the computational time of obtaining the best returned solution of multi-start local search algorithm. There is no clear pattern that the best returned solution can be found in short time or right before the algorithm terminates.

	CPLEX			GREEDY-4		SEMIGREEDY-20-50-4		MULLS-50		
Inst	Opt	LBD	Time	Val	Time	Val	Time	Val	Time	BT
$n = 1000$										
1	-	41207.44	1815.70	40832.75	**0.00**	40946.36	0.02	**41085.15**	31.70	31.09
2	-	37692.29	1828.16	37661.19	**0.00**	37687.74	0.01	**37772.47**	23.06	14.85
3	-	34940.97	1823.17	34820.02	**0.00**	34961.28	0.01	**35054.31**	26.37	19.53
4	-	56658.11	1818.11	56142.92	**0.00**	56296.10	0.02	**56477.49**	47.39	14.35
5	-	54119.11	1808.08	53566.65	**0.00**	53671.56	0.02	**53920.42**	40.92	16.79
6	-	51700.14	1805.55	51305.19	**0.00**	51379.22	0.02	**51620.35**	52.92	42.83
7	-	73655.99	1853.63	73019.13	**0.01**	73207.05	0.02	**73482.06**	65.30	55.80
8	-	79080.85	1821.97	78492.17	**0.00**	78689.06	0.03	**78832.70**	67.01	34.98
9	-	73194.08	1821.83	72647.34	**0.00**	72842.56	0.02	**73159.69**	60.65	25.30
10	-	0.00	1802.02	96058.96	**0.00**	96124.68	0.03	**96392.03**	61.15	49.35
11	-	0.00	1802.78	88947.42	**0.00**	89296.54	0.02	**89548.96**	87.52	5.85
12	-	0.00	1801.86	80876.25	**0.00**	81102.68	0.03	**81315.38**	93.40	54.60
13	-	0.00	1800.41	113856.14	**0.00**	114077.99	0.03	**114561.07**	74.26	18.63
14	-	0.00	1800.80	109679.40	**0.00**	109827.76	0.03	**110122.93**	95.17	50.19
15	-	0.00	1819.43	106404.52	**0.00**	106852.06	0.03	**107003.86**	94.94	30.04
$n = 2000$										
1	-	0.00	1839.60	105668.40	**0.00**	105748.97	0.06	**106006.12**	303.87	89.54
2	-	0.00	1804.57	98938.77	**0.00**	99005.59	0.06	**99378.59**	416.19	370.52
3	-	0.00	1802.56	103297.83	**0.00**	103411.52	0.06	**103610.35**	298.37	52.86
4	-	0.00	1811.30	158793.82	**0.00**	158835.87	0.08	**159359.59**	544.42	195.77
5	-	0.00	1809.24	160422.53	**0.00**	160725.86	0.08	**161077.84**	520.22	375.17
6	-	0.00	1832.66	155765.71	**0.00**	155980.97	0.08	**156306.62**	534.78	450.55
7	-	0.00	1803.90	205395.22	**0.01**	205639.12	0.09	**206057.80**	767.79	244.66
8	-	0.00	1803.00	219622.95	**0.00**	219793.61	0.08	**220321.24**	707.75	211.66
9	-	0.00	1807.04	209546.51	**0.00**	209901.17	0.09	**210245.57**	547.61	492.17
10	-	0.00	1802.85	273833.21	**0.01**	274142.06	0.15	**274817.32**	961.01	306.91
11	-	0.00	1807.94	245274.62	**0.00**	245658.09	0.11	**246345.59**	1177.91	60.20
12	-	0.00	1806.62	260187.84	**0.00**	260711.60	0.10	**261306.47**	924.39	679.11
13	-	0.00	1804.50	308899.94	**0.00**	308987.74	0.13	**309566.81**	1213.49	267.99
14	-	0.00	1804.26	314471.41	**0.00**	314942.66	0.13	**315925.00**	1233.62	939.67
15	-	0.00	1808.14	311851.38	**0.00**	312479.76	0.13	**313293.69**	1402.93	913.96
$n = 5000$										
1	-	0.00	1805.32	440855.19	**0.01**	441147.81	0.35	**441648.62**	1979.33	95.617
2	-	0.00	1808.09	415341.88	**0.01**	**415569.52**	0.33	415082.83	1918.53	1918.53
3	-	0.00	1803.98	408052.27	**0.01**	408455.27	0.35	**409066.17**	1878.10	937.428
4	-	0.00	1804.52	629984.32	**0.03**	630049.68	0.61	**631128.92**	1980.33	1231.64
5	-	0.00	1803.98	624029.56	**0.02**	624135.73	0.44	**625044.06**	1936.34	1936.33
6	-	0.00	1806.77	632979.02	**0.01**	**633109.88**	0.42	632485.01	2193.47	974.587
7	-	0.00	1810.31	856520.91	**0.02**	857288.91	0.52	**858611.29**	2017.72	790.321
8	-	0.00	1820.24	830287.45	**0.02**	831267.48	0.62	**832425.27**	2257.97	2257.96
9	-	0.00	1841.31	835310.06	**0.02**	835481.86	0.58	**837038.74**	2204.84	558.123
10	-	0.00	1816.12	1085101.00	**0.03**	**1085975.97**	0.63	1084937.19	1850.22	1219.25
11	-	0.00	1810.31	1050882.27	**0.03**	**1051723.59**	0.60	1050344.90	1827.06	604.264
12	-	0.00	1826.84	1026288.69	**0.02**	1027122.88	0.69	**1028201.42**	1837.23	1837.22
13	-	0.00	1823.88	1291241.84	**0.03**	1292144.55	0.72	**1293838.88**	1839.48	630.789
14	-	0.00	1816.50	1248116.12	**0.03**	1249832.22	0.71	**1251754.12**	1825.81	1217.07
15	-	0.00	1810.79	1237909.07	**0.03**	1239678.53	0.80	**1241231.63**	1840.96	1840.94

Table 4.6: DPSP solutions on Test bed 2 – Value

	CPLEX		GREEDY-4		SEMIGREEDY-20-50-4		MULLS-50	
Inst	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time
$n = 1000$								
1	0.50	1815.70	1.41	**0.00**	1.13	0.02	**0.80**	31.70
2	1.08	1828.16	1.16	**0.00**	1.09	0.01	**0.87**	23.06
3	1.19	1823.17	1.53	**0.00**	1.13	0.01	**0.87**	26.37
4	0.64	1818.11	1.55	**0.00**	1.28	0.02	**0.96**	47.39
5	0.59	1808.08	1.60	**0.00**	1.41	0.02	**0.95**	40.92
6	0.87	1805.55	1.63	**0.00**	1.49	0.02	**1.03**	52.92
7	0.72	1853.63	1.58	**0.01**	1.33	0.02	**0.96**	65.30
8	0.71	1821.97	1.45	**0.00**	1.20	0.03	**1.02**	67.01
9	1.08	1821.83	1.82	**0.00**	1.55	0.02	**1.13**	60.65
10	-	1802.02	1.51	**0.00**	1.44	0.03	**1.16**	61.15
11	-	1802.78	1.85	**0.00**	1.46	0.02	**1.18**	87.52
12	-	1801.86	1.82	**0.00**	1.54	0.03	**1.28**	93.40
13	-	1800.41	1.77	**0.00**	1.58	0.03	**1.17**	74.26
14	-	1800.80	1.61	**0.00**	1.48	0.03	**1.21**	95.17
15	-	1819.43	1.85	**0.00**	1.44	0.03	**1.30**	94.94
$n = 2000$								
1	-	1839.60	1.09	**0.00**	1.02	0.06	**0.78**	303.87
2	-	1804.57	1.22	**0.00**	1.15	0.06	**0.78**	416.19
3	-	1802.56	0.95	**0.00**	0.85	0.06	**0.65**	298.37
4	-	1811.30	1.14	**0.00**	1.12	0.08	**0.79**	544.42
5	-	1809.24	1.23	**0.00**	1.04	0.08	**0.83**	520.22
6	-	1832.66	1.08	**0.00**	0.94	0.08	**0.73**	534.78
7	-	1803.90	1.20	**0.01**	1.08	0.09	**0.88**	767.79
8	-	1803.00	1.20	**0.00**	1.13	0.08	**0.89**	707.75
9	-	1807.04	1.09	**0.00**	0.92	0.09	**0.76**	547.61
10	-	1802.85	1.24	**0.01**	1.13	0.15	**0.89**	961.01
11	-	1807.94	1.44	**0.00**	1.28	0.11	**1.01**	1177.91
12	-	1806.62	1.35	**0.00**	1.15	0.10	**0.92**	924.39
13	-	1804.50	1.22	**0.00**	1.19	0.13	**1.00**	1213.49
14	-	1804.26	1.46	**0.00**	1.32	0.13	**1.01**	1233.62
15	-	1808.14	1.52	**0.00**	1.32	0.13	**1.06**	1402.93
$n = 5000$								
1	-	1805.32	0.78	**0.01**	0.71	0.35	**0.60**	1979.33
2	-	1808.09	0.88	**0.01**	**0.83**	0.33	0.94	1918.53
3	-	1803.98	0.85	**0.01**	0.75	0.35	**0.60**	1878.10
4	-	1804.52	0.82	**0.03**	0.81	0.61	**0.64**	1980.33
5	-	1803.98	0.80	**0.02**	0.78	0.44	**0.63**	1936.34
6	-	1806.77	0.85	**0.01**	**0.83**	0.42	0.93	2193.47
7	-	1810.31	0.92	**0.02**	0.83	0.52	**0.68**	2017.72
8	-	1820.24	0.94	**0.02**	0.82	0.62	**0.69**	2257.97
9	-	1841.31	0.89	**0.02**	0.87	0.58	**0.68**	2204.84
10	-	1816.12	0.90	**0.03**	**0.82**	0.63	0.92	1850.22
11	-	1810.31	**0.95**	**0.03**	**0.87**	0.60	1.00	1827.06
12	-	1826.84	1.00	**0.02**	0.92	0.69	**0.81**	1837.23
13	-	1823.88	0.90	**0.03**	0.83	0.72	**0.70**	1839.48
14	-	1816.50	1.18	**0.03**	1.04	0.71	**0.89**	1825.81
15	-	1810.79	1.10	**0.03**	0.96	0.80	**0.84**	1840.96

Table 4.7: DPSP solutions on Test bed 2 – Deviation

To further analyze each algorithm, we also present a summarized results for each algorithm in Table 4.8 to 4.10. Recall that 3 instances are generated for each given instance size, we calculate the average, maximum, minimum deviation and time over the 3 instances with each given size to eliminate bias.

Greedy algorithm summary solving results are given in Table 4.8. We first observe that the computational time is negligible even for largest testing instance and the deviation is between 0.78% and 1.85%. Second we see a trend that the deviation is decreasing as the instance size increasing. However, we believe this results from the enlarged objective function value order instead of algorithm performance. This observation and conclusion also applicable for semi-greedy algorithm and multi-start local search algorithm.

		AVE		MAX		MIN	
n	m	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time
1000	100	1.37	0.00	1.53	0.00	1.16	0.00
1000	150	1.59	0.00	1.63	0.00	1.55	0.00
1000	200	1.62	0.00	1.82	0.01	1.45	0.00
1000	250	1.72	0.00	1.85	0.00	1.51	0.00
1000	300	1.74	0.00	1.85	0.00	1.61	0.00
2000	200	1.09	0.00	1.22	0.00	0.95	0.00
2000	300	1.15	0.00	1.23	0.00	1.08	0.00
2000	400	1.17	0.00	1.20	0.01	1.09	0.00
2000	500	1.34	0.01	1.44	0.01	1.24	0.00
2000	600	1.40	0.00	1.52	0.00	1.22	0.00
5000	500	0.84	0.01	0.88	0.01	0.78	0.01
5000	750	0.82	0.02	0.85	0.03	0.80	0.01
5000	1000	0.92	0.02	0.94	0.02	0.89	0.02
5000	1250	0.95	0.03	1.00	0.03	0.90	0.02
5000	1500	1.06	0.03	1.18	0.03	0.90	0.03

Table 4.8: DPSP - the results of greedy algorithm on Test bed 2

Semi-greedy algorithm summary solving results are given in Table 4.9. Its computational time is still less than one second even for the largest testing instance with $n = 5000, m = 1500$. The deviation is between 0.71% and 1.58%.

		AVE		MAX		MIN	
n	m	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time
1000	100	1.12	0.02	1.13	0.02	1.09	0.01
1000	150	1.39	0.02	1.49	0.02	1.28	0.02
1000	200	1.36	0.02	1.55	0.03	1.20	0.02
1000	250	1.48	0.03	1.54	0.03	1.44	0.02
1000	300	1.50	0.03	1.58	0.03	1.44	0.03
2000	200	1.00	0.06	1.15	0.06	0.85	0.06
2000	300	1.03	0.08	1.12	0.08	0.94	0.08
2000	400	1.05	0.09	1.13	0.09	0.92	0.08
2000	500	1.19	0.12	1.28	0.15	1.13	0.10
2000	600	1.27	0.13	1.32	0.13	1.19	0.13
5000	500	0.76	0.34	0.83	0.35	0.71	0.33
5000	750	0.81	0.49	0.83	0.61	0.78	0.42
5000	1000	0.84	0.57	0.87	0.62	0.82	0.52
5000	1250	0.87	0.64	0.92	0.69	0.82	0.60
5000	1500	0.95	0.74	1.04	0.80	0.83	0.71

Table 4.9: DPSP - the results of semi-greedy algorithm on Test bed 2

Multi-start local search algorithm summary solving results are given in Table 4.10. We see that its computational time increases dramatically as the instance size increase. The deviation is between 0.60% to 1.17%. We also tested multi-start local search algorithm without time limit. The computational time for $n = 5000$ goes to 5 hours. There are 4 cases that this algorithm fails to improve the solution quality comparing to semi-greedy algorithm. It is due to the time limit we set so in half an hour it fails to find an improved solution.

		AVE		MAX		MIN	
n	m	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time
1000	100	0.85	27.05	0.87	31.70	0.80	23.06
1000	150	0.98	47.08	1.03	52.92	0.95	40.92
1000	200	1.03	64.32	1.13	67.01	0.96	60.65
1000	250	1.21	80.69	1.28	93.40	1.16	61.15
1000	300	1.22	88.12	1.30	95.17	1.17	74.26
2000	200	0.74	339.48	0.78	416.19	0.65	298.37
2000	300	0.78	533.14	0.83	544.42	0.73	520.22
2000	400	0.84	674.38	0.89	767.79	0.76	547.61
2000	500	0.94	1021.10	1.01	1177.91	0.89	924.39
2000	600	1.02	1283.35	1.06	1402.93	1.00	1213.49
5000	500	0.72	1925.32	0.94	1979.33	0.60	1878.10
5000	750	0.74	2036.71	0.93	2193.47	0.63	1936.34
5000	1000	0.68	2160.18	0.69	2257.97	0.68	2017.72
5000	1250	0.91	1838.17	1.00	1850.22	0.81	1827.06
5000	1500	0.81	1835.42	0.89	1840.96	0.70	1825.81

Table 4.10: DPSP - the results of multi-start local search algorithm on Test bed 2

A summary result is presented in Table 4.11, 4.12 and 4.13. In summary, greedy algorithm has excellent speed advantage and the solution quality is good with deviation within 1.85%. Semi-greedy algorithm returns better solution with deviation at most 1.58% while maintaining the advantage of short computational time. Multi-start local search algorithm returns the best known solution with deviation within 1.30% in most cases at a compromising speed. All the heuristic algorithms are able to provide feasible solutions, while CPLEX is able to return nonzero feasible solutions for 9 instances out of 45 in half an hour.

We also did experiments to solve a DPSP instance with $n = 2000, m = 600$ with time limit up to 5 hours. CPLEX is able to find feasible solutions within this time and the solution quality is better than our heuristic algorithms. Moreover, taking greedy solution as initial solution, CPLEX manages to get improvement within 15 minutes of which the quality is better than that of our heuristic algorithms.

	Average		Max		Min		
Alg	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time	#Opt Inst
CPLEX	-	1814.90	100.00	1853.63	0.50	1800.41	0
GREEDY-4	1.61	**0.00**	1.85	**0.01**	1.16	**0.00**	**0**
SEMIGREEDY-20-50-4	1.37	0.02	1.58	0.03	1.09	0.01	**0**
MULLS-50	**1.06**	61.45	**1.30**	95.17	**0.80**	23.06	**0**

Table 4.11: DPSP - the summary results of algorithms on Testbed 2: $n = 1000$

	Average		Max		Min		
Alg	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time	#Opt Inst
CPLEX	-	1809.88	100.00	1839.60	100.00	1802.56	0
GREEDY-4	1.23	**0.00**	1.52	**0.01**	0.95	**0.00**	**0**
SEMIGREEDY-20-50-4	1.11	0.10	1.32	0.15	0.85	0.06	**0**
MULLS-50	**0.87**	770.29	**1.06**	1402.93	**0.65**	298.37	**0**

Table 4.12: DPSP - the summary results of algorithms on Testbed 2: $n = 2000$

	Average		Max		Min		
Alg	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time	#Opt Inst
CPLEX	-	1813.93	100.00	1841.31	100.00	1803.98	0
GREEDY-4	0.92	**0.02**	1.18	**0.03**	0.78	**0.01**	**0**
SEMIGREEDY-20-50-4	0.85	0.56	1.04	0.80	0.71	0.33	**0**
MULLS-50	**0.77**	1959.16	**1.00**	2257.97	**0.60**	1825.81	**0**

Table 4.13: DPSP - the summary results of algorithms on Testbed 2: $n = 5000$

4.4 Extension of DPSP

Based on the above DPSP model, more realistic elements could be taken into consideration. In this section, we further introduce our extended DPSP model. Two issues can be added into the model.

First, some edges may be owned or partially owned by existing customers or an interested third party, for whom a future sharing fee is promised. Let $\beta_l \in (0, 1)$ be the payout ratio if such an edge is selected by l paths, then amount $\beta_l \phi_i(l)$ of edge cost goes to the edge owner and $(1 - \beta_l)\phi_i(l)$ is the revenue of the company. However, this situation can be easily

incorporated in our original DPSP model by modifying the corresponding function $\phi_i(\cdot)$ to $\phi'_i(\cdot)$ such that $\phi'_i(l) = (1 - \beta_l)\phi_i(l)$ for $l \in N_i$. This modification is always applicable since the discounted edge cost function need not be continuous. Note that the resulting discounted edge cost function may not obey conditions in (4.2) to (4.5).

Second, the budget information of some clients may be available already. Deciding to construct these paths of the customers requires that the path cost meets the budget. Let $N^b \subseteq N$ be a path index set and for each $j \in N^b$ a *budget* θ_j is prescribed. To formulate this budget restriction, we recall that for each edge e_i, the total discounted cost is expressed as $\sum_{l=1}^{|N_i|} z_i^l \phi_i(l)$. Then for each selected path that shares this edge, the cost it shares is $\sum_{l=1}^{|N_i|} \frac{1}{l} z_i^l \phi_i(l)$. Hence the total cost of the path P_j is given by $c_j := \sum_{i \in M_j} \sum_{l=1}^{|N_i|} \frac{1}{l} z_i^l \phi_i(l)$. The budget restriction comes to effect only when path P_j is selected, the budget constraint can be formulated as

$$\sum_{i \in M_j} \sum_{l=1}^{|N_i|} \frac{1}{l} z_i^l \phi_i(l) \leq \theta_j + \Theta(1 - x_j) \quad j \in N^b \tag{4.16}$$

where Θ is a large enough number. When the path is selected $x_j = 1$, the constraint is effective. Otherwise, Θ ensures that the constraint always holds. We refer it as *path budget constraints*.

Chapter 5

PSP with Capacity Expansion

5.1 EPSP model

In this section, we study another version of PSP to provide lower quotes for clients. Since network maintenance and lease are also sources of revenue for the company, it is willing to take certain risks to invest on building some segments by itself. Moreover, the edge capacities are not strict restrictions. As long as there are enough additional investment, increasing the edge capacity is very possible. Hence we consider another problem that provide the minimum investment the company has to make so that at least a number of paths are selected. In this way, the cost of a path is reduced to the nominal cost subtracted by the cost of selected building edges. A resulting lower quote may attract more customers and an enlarged network will lead to a return in the future.

We consider the following scenario: Let $G = (V, E)$ be a underlying graph of GPSP with paths $F = \{P_j : j \in N\}$ in G. Define $N_i = \{j \in N : e_i \in P_j\}$. For each edge $e_i, i \in M$, a cost w_i and a capacity b_i are prescribed. To increase the capacity of edge e_i, a minimum unit u_i with a cost d_i are given, i.e., the expanded edge capacity is always in the form of $b_i + lu_i$ where l is a nonnegative integer. Let a_j denotes the capacity of a path P_j and for $j \in N^b \subseteq N$ a *budget* θ_j is prescribed. The *Path Selection Problem with edge capacity expansion* (EPSP) is to select a set of edge $M^b \subseteq M$ to build and assign expansion amount t_i for each edge e_i, $i \in M$ such that the total cost on edge building and capacity expansion is minimized while at least γ paths are selected, and these selected paths obey edge capacity restriction and budget constraints.

We now formulate EPSP into integer programming problem. For each $j \in N$, define x_j

as (2.1). For $i \in M$, let

$$s_i = \begin{cases} 1 & \text{if } e_i \text{ is selected to build} \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

Then the edge building expense is $\sum_{i \in M} w_i s_i$. For $j \in M$, define nonnegative integer variable $t_i \in \mathbb{Z}_+$ as the edge capacity expansion amount for e_i, and the resulting capacity of e_i becomes $b_i + t_i u_i$. Then the edge expansion cost is $\sum_{i \in M} d_i t_i$. Therefore, the objective function is formulated as

$$p(s,t) = \sum_{i \in M} w_i s_i + \sum_{i \in M} d_i t_i \tag{5.2}$$

Since the expanded edge capacity for edge e_i becomes $b_i + t_i u_i$, the edge capacity constraints are modified to

$$\sum_{j \in N_i} a_j x_j \leq b_i + t_i u_i \quad \forall i \in M \tag{5.3}$$

The client does not pay for edge e_i if it is selected to build and the path cost is reduced from its nominal cost $\sum_{i \in M_j} w_i$ to $\sum_{i \in M_j} w_i - \sum_{i \in M_j} w_i s_i$. Let $c_j = \sum_{i \in M_j} w_i$, hence the budget constraints are modified as

$$c_i x_j - \sum_{i \in M_j} w_i s_i \leq \theta_j + \Theta(1 - x_j) \quad \forall j \in N^b \tag{5.4}$$

where Θ is a large enough positive number. Together with the selected paths quantity restriction, the integer programming formulation for EPSP is given as follows:

EPSP: Minimize $\quad p(s,t) = \sum_{i \in M} w_i s_i + \sum_{i \in M} d_i t_i$

Subject to:

$$\sum_{j \in N_i} a_j x_j \leq b_i + t_i u_i \quad \forall i \in M$$

$$c_j x_j - \sum_{i \in M_j} w_i s_i \leq \theta_j + \Theta(1 - x_j) \quad \forall j \in N^b$$

$$\sum_{j \in N} x_j \geq \gamma$$

$$x_j \in \{0,1\} \quad \forall j \in N$$

$$s_i \in \{0,1\} \quad \forall i \in M$$

$$t_i \in \mathbb{Z}_+ \quad \forall i \in M$$

We now given our complexity analysis for EPSP.

Theorem 5.1.1. *EPSP is NP-hard.*

Proof. The formulation (3.10) of MSCP decision problem is a special case of EPSP where $w_i = 0$, $d_i = 0$, $u_i = 0$ for $i \in M$ and $a_j = 1$ for $j \in N$. Due to the NP-completeness of MSCP, EPSP is NP-hard. □

Theorem 5.1.2. *Given a feasible solution $x \in \{0,1\}^n$ to EPSP, the corresponding optimal solution $t \in$ can be obtained in polynomial time while solving the corresponding optimal solution s is NP-hard.*

Proof. Given x, let $J = \{j : x_j = 1\}$. To find the corresponding feasible solution s, t, EPSP becomes

$$\begin{aligned}
&\text{Minimize:} \quad \sum_{i \in M} w_i s_i + \sum_{i \in M} d_i t_i \\
&\text{Subject to:} \\
&\qquad \sum_{i \in M_j} t_i u_i \geq b_i - \sum_{j \in N_i \cap J} a_j \quad i \in M \\
&\qquad \sum_{i \in M_j} w_i s_i \geq c_j - \theta_j \quad j \in N^b \cap J \\
&\qquad s_i \in \{0,1\} \quad i \in M \\
&\qquad t_i \in \mathbb{Z}_+ \quad i \in M
\end{aligned} \tag{5.5}$$

It is obvious that problem (5.5) can be divided into two independent subproblems respect to s and t. Moreover, the optimal solution t of (5.5) is given as follows

$$t_i = \max\left\{0, \left\lceil \frac{b_i - \sum_{j \in N_i \cap J} a_j}{u_i} \right\rceil \right\} \quad i \in M \tag{5.6}$$

This implies that t can be obtained within polynomial time.

We prove the second part is true by showing that it is true on a special case. Consider a EPSP instance where $\gamma = n$, $w_i = 1$, $d_i = 0$ for $i \in M$ and $\theta_j = c_j - 1$ for $j \in N$. Then the only feasible solution x is $x_j = 1$ for all $j \in N$. The above feasibility problem (5.5) (which

is also equivalent to the original EPSP1) becomes

$$\begin{aligned} &\text{Minimize:} && \sum_{i \in M} s_i \\ &\text{Subject to:} && \\ & && \sum_{i \in M_j} s_i \geq 1 \quad j \in N \\ & && s_i \in \{0,1\} \quad i \in M \end{aligned}$$

Since there is no restriction for M_j, the above problem is a general MSCP. Due to the NP-completeness of MSCP, the result follows. □

5.2 Heuristic algorithm for EPSP

We develop a greedy algorithm and a semi-greedy algorithm for EPSP in this section.

Greedy algorithm for EPSP

The objective function contains two parts: the cost of building edges and the cost of expanding edge capacities. The greedy algorithm first considers to select path without building any edges or expanding edge capacity. To do this, we apply greedy algorithm on all the paths without budget restrictions with given original edge capacity. Let x be the resulting partial solution and initialize $s = \mathbf{0}, \mathbf{t} = \mathbf{0}$. Then for the remaining paths including those with budget constraints and those are not selected in the first round, we calculate the approximate cost to select path P_j as

$$r_j = \max\{0, c_j - \theta_j - \sum_{i \in M_j} w_i s_i\} + \sum_{i \in M_j} \max\{0, \left\lceil \frac{\sum_{k \in N_i} a_k x_k + a_j - b_i - t_i u_i}{u_i} \right\rceil * d_i\} \quad (5.7)$$

In each iteration, we select the path with lowest value r_j. To do this, we expand edge capacity to meet capacity constraints and select edges to build to meet budget constraints. It is straightforward to expand edge capacity as given in formula (5.6). To select edges to build, we simply check all the edges that are not chosen to build before and add them to be build until the cost of the path under consideration is less than its budget. At last we update r_j for unconsidered paths and the procedure terminates as soon as γ paths are selected. A formal description of the greedy algorithm is given below:

Algorithm 10: Greedy Algorithm for EPSP

Input: permutation π of $N \setminus N^b$

Initialize $x \leftarrow \mathbf{0}$, $s \leftarrow \mathbf{0}$, $t \leftarrow \mathbf{0}$;

for $j \leftarrow 1$ *to* $|N \setminus N^b|$ **do** `/* first round selection */`

  $path \leftarrow \pi(j)$;

  if $(x|x_{path} = 1)$ *is feasible* **then**

    $x_{path} = 1$;

    if $\sum_{j \in N} x_j \geq \gamma$ **then**

      return x;

$S' = \{j \in N : x_j = 0\}$;

for $j \in S'$ **do** `/* initialize` r_j `*/`

  $r_j = \max\{0, c_j - \theta_j\} + \sum_{i \in M_j} \max\{0, \lceil \frac{\sum_{k \in N_i} a_k x_k + a_j - b_i}{u_i} \rceil * d_i\}$;

$\sigma \leftarrow$ permutation of S' according to decreasing value r_j;

for $j \leftarrow 1$ *to* $|S'|$ **do** `/* second round selection */`

  $path \leftarrow \sigma(j)$;

  $x_{path} = 1$;

  for $k \in M_j$ **do** `/* expand capacity */`

    if $a_k x_k + a_j - b_i - t_i u_i > 0$ **then**

      $t_i \leftarrow t_i + \lceil \frac{\sum_{k \in N_i} a_k x_k + a_j - b_i - t_i u_i}{u_i} \rceil$;

  $\delta_j = c_j - \theta_j - \sum_{i \in M_j} w_i s_i$;

  if $\delta_j > 0$ **then** `/* build edge */`

    for $k \in M_j \cap \{i \in M : s_i = 0\}$ *and* $\delta_j > 0$ **do**

      $s_k = 1$;

  Update ratio $r_{\sigma(k)}$ for $k = j + 1$ to $|S'|$ according to (5.7);

  Update permutation from $\sigma(j + 1)$ to $\sigma(|S'|)$;

  if $\sum_{j \in N} x_j \geq \gamma$ **then**

    return x;

Semi-greedy algorithm for EPSP

Based on the greedy algorithm, we develop a semi-greedy algorithm. In each iteration, we keep the first round selection as the same as greedy algorithm and allow some extent of less "greediness" in the second round selection. More precisely, instead of selecting path with lowest value r_j, we pick one randomly from a candidate list with good enough values r_j and set it as one. We repeat the greedy selection procedure $iLim$ iterations and return the best found solution. A formal description of the semi-greedy algorithm is given below:

Algorithm 11: Semi-greedy Algorithm for EPSP

Input: $lCand$, $iLim$, permutation π of N

Initialize: $x^* \leftarrow \mathbf{0}$, $s^* \leftarrow \mathbf{0}$, $t^* \leftarrow \mathbf{0}$, $iter \leftarrow 1$;

repeat

  $x \leftarrow \mathbf{0}$, $s \leftarrow \mathbf{0}$, $t \leftarrow \mathbf{0}$;

  for $j \leftarrow 1$ *to* $|N \setminus N^b|$ **do** `/* first round selection */`

    $path \leftarrow \pi(j)$;

    if $(x|x_{path} = 1)$ *is feasible* **then**

      $x_{path} = 1$;

      if $\sum_{j \in N} x_j \geq \gamma$ **then**

        Go to step 1;

  $S' = \{j \in N : x_j = 0\}$;

  for $j \in S'$ **do** `/* initialize` r_j `*/`

    $r_j = \max\{0, c_j - \theta_j\} + \sum_{i \in M_j} \max\{0, \lceil \frac{\sum_{k \in N_i} a_k x_k + a_j - b_i}{u_i} \rceil * d_i\}$;

  $\sigma \leftarrow$ permutation of S' according to decreasing value r_j;

  $next \leftarrow lCand + 1$, candidate list $C \leftarrow \{\sigma(1), \sigma(2), \ldots, \sigma(lCand)\}$;

  while $C \neq \emptyset$ **do**

    $path \leftarrow$ select a random element in C ; `/* random selection */`

    $x_{path} = 1$;

    for $k \in M_j$ **do** `/* expand capacity */`

      if $a_k x_k + a_j - b_i - t_i u_i > 0$ **then**

        $t_i \leftarrow t_i + \lceil \frac{\sum_{k \in N_i} a_k x_k + a_j - b_i - t_i u_i}{u_i} \rceil$;

    if $c_j - \theta_j - \sum_{i \in M_j} w_i s_i > 0$ **then** `/* build edge */`

      for $k \in M_j \cap \{i \in M : s_i = 0\}$ **do**

        if $c_j - \theta_j - \sum_{i \in M_j} w_i s_i > 0$ **then**

          $s_k = 1$;

    Update permutation from $\sigma(j+1)$ to $\sigma(|S'|)$ according to updated value r_j;

    if $next \leq n$ **then** `/* update` C `*/`

      $C \leftarrow C \cup \{\sigma(next)\} \setminus \{path\}$;

      $next \leftarrow next + 1$;

    else

      $C \leftarrow C \setminus \{path\}$;

    if $\sum_{j \in N} x_j \geq \gamma$ **then**

      Go to step 1;

  Step 1:

  if $p(s,t) > p(s^*, t^*)$ **then** `/* update best known solution */`

    $x^* \leftarrow x$, $s^* \leftarrow s$, $t^* \leftarrow t$;

  $iter \leftarrow iter + 1$;

until $iter = iLim$;

5.3 Computational results and analysis

The instance generator for EPSP is based on that of GPSP. The additional data of EPSP is generated as follows: Given $\beta \in (0,1)$, we set the cardinality of path set with budget N^b as βn and the elements of N^b are filled randomly from N. After setting N^b, the budget θ_j is set as a percentage of the nominal path cost κc_j where $\kappa \in (0,1)$ which is generated between a predefined bounds. The selected path number γ is set as τn where $\tau \in (0,1)$. At last, we set the edge capacity expansion unit $u_i = 1$ with associated cost $d_i = \frac{w_i}{2b_i}$. The involved parameters are summarized as follows:

- $\beta \in (0,1)$: determines cardinality of N^b, $|N^b| = \beta n$;
- dl, du: lower bound and upper bound to generate κ, $\kappa = urand(dl*100, du*100)/100$;
- $\tau \in (0,1)$: determines selected path number γ, $\gamma = \tau n$;

For our test bed instance, we set instance size as $n(2000)$, $m(600, 1000)$, $\alpha(0.50)$, $rl(2)$, $ru(0.2n)$, $U^{au}(0.05)$, $al(1)$, $cl(1)$, $cu(30)$, $\beta(0.5, 0.8)$, $\tau(0.5, 0.6, 0.7)$, $dl(0.50)$, $du(0.80)$. The instances are solved by CPLEX given time limit 0.5 hour, the solving results are reported in Table 5.1. There are no optimal solution returned during the time limit for all instances in test bed.

ID	(m, n)	α	β	$dl - du$	τ	OPT	LBD	Time/s	UBD(LP relax)
1	(600,2000)	0.5	0.5	0.5-0.8	0.5	-	723.75	1800.49	0.00
2	(600,2000)	0.5	0.5	0.5-0.8	0.5	-	531.64	1800.97	0.00
3	(600,2000)	0.5	0.5	0.5-0.8	0.5	-	488.48	1801.67	0.00
4	(600,2000)	0.5	0.5	0.5-0.8	0.6	-	1885.11	1803.48	0.00
5	(600,2000)	0.5	0.5	0.5-0.8	0.6	-	1841.16	1800.70	0.00
6	(600,2000)	0.5	0.5	0.5-0.8	0.6	-	1872.15	1800.72	0.00
7	(600,2000)	0.5	0.5	0.5-0.8	0.7	-	2808.33	1807.00	33.77
8	(600,2000)	0.5	0.5	0.5-0.8	0.7	-	2679.09	1808.17	27.80
9	(600,2000)	0.5	0.5	0.5-0.8	0.7	-	2851.57	1807.44	33.52
10	(600,2000)	0.5	0.8	0.5-0.8	0.5	-	1757.87	1800.94	0.00
11	(600,2000)	0.5	0.8	0.5-0.8	0.5	-	1755.31	1802.93	0.00
12	(600,2000)	0.5	0.8	0.5-0.8	0.5	-	1785.08	1803.73	0.00
13	(600,2000)	0.5	0.8	0.5-0.8	0.6	-	2455.69	1804.00	0.00
14	(600,2000)	0.5	0.8	0.5-0.8	0.6	-	2327.19	1800.84	0.00
15	(600,2000)	0.5	0.8	0.5-0.8	0.6	-	2511.15	1802.13	0.00
16	(600,2000)	0.5	0.8	0.5-0.8	0.7	-	3346.91	1806.00	33.92
17	(600,2000)	0.5	0.8	0.5-0.8	0.7	-	3140.13	1805.47	27.92
18	(600,2000)	0.5	0.8	0.5-0.8	0.7	-	3309.58	1804.77	33.67
19	(1000,2000)	0.5	0.5	0.5-0.8	0.5	-	1076.52	1802.82	0.00
20	(1000,2000)	0.5	0.5	0.5-0.8	0.5	-	1058.27	1800.65	0.00
21	(1000,2000)	0.5	0.5	0.5-0.8	0.5	-	1284.08	1800.59	0.00
22	(1000,2000)	0.5	0.5	0.5-0.8	0.6	-	3294.17	1800.60	0.00
23	(1000,2000)	0.5	0.5	0.5-0.8	0.6	-	2965.40	1801.51	0.00
24	(1000,2000)	0.5	0.5	0.5-0.8	0.6	-	3415.54	1800.73	0.00
25	(1000,2000)	0.5	0.5	0.5-0.8	0.7	-	4844.88	1800.45	72.27
26	(1000,2000)	0.5	0.5	0.5-0.8	0.7	-	4456.49	1800.69	55.98
27	(1000,2000)	0.5	0.5	0.5-0.8	0.7	-	4908.44	1800.88	123.63
28	(1000,2000)	0.5	0.8	0.5-0.8	0.5	-	2828.26	1802.62	0.00
29	(1000,2000)	0.5	0.8	0.5-0.8	0.5	-	2889.61	1800.72	0.00
30	(1000,2000)	0.5	0.8	0.5-0.8	0.5	-	2990.17	1800.72	0.00
31	(1000,2000)	0.5	0.8	0.5-0.8	0.6	-	3836.34	1801.18	0.00
32	(1000,2000)	0.5	0.8	0.5-0.8	0.6	-	3890.76	1801.44	0.00
33	(1000,2000)	0.5	0.8	0.5-0.8	0.6	-	4118.56	1800.55	0.00
34	(1000,2000)	0.5	0.8	0.5-0.8	0.7	-	5148.10	1800.09	72.68
35	(1000,2000)	0.5	0.8	0.5-0.8	0.7	-	5008.74	1804.80	56.34
36	(1000,2000)	0.5	0.8	0.5-0.8	0.7	-	5361.79	1800.16	124.13

Table 5.1: EPSP - the result of CPLEX on test bed

We test greedy algorithm on instance E-1, E-1 and E-3 with different ratios defined in Table 2.1. All the ratios give the same solution, hence we choose Ratio 5 for the greedy algorithm to run on the whole test bed. Semi-greedy algorithm with $iLim = 50$, $lCand = 5$ and Ratio 5 is tested on the test bed. The result is given in Table .

We first observe that CPLEX almost always provides better solution than our developed greedy algorithm and semi-greedy algorithm. However, for instances 1, 2, 3, 19, 20 and 21 it loses this supreme. These instance are generated with $\tau = 0.5$. Second, comparing greedy algorithm with semi-greedy algorithm, the former gives a better solution only for 3 out of 36 instances. Although semi-greedy algorithm does not outperform its peer in all cases, its overall more competitive performance is still proved. Third, we notice that the computational time for both greedy algorithm and semi-greedy algorithm are on a larger order comparing to that for GPSP, BPSP and DPSP. This is due to the ratio update and the fact that even given selected path set we also need to select the edges to build.

	CPLEX			GREEDY-5		SEMIGREEDY-50-5-5	
Inst	Opt	LBD	Time	Val	Time	Val	Time
1	-	723.75	1800.49	**183.19**	0.06	**183.19**	2.92
2	-	531.64	1800.97	**185.03**	0.05	**185.03**	2.59
3	-	488.48	1801.67	**148.93**	0.05	**264.21**	2.50
4	-	1885.11	1803.48	3036.17	0.19	2893.06	8.14
5	-	1841.16	1800.70	2820.31	0.17	2749.49	7.75
6	-	1872.15	1800.72	2903.22	0.19	2763.52	7.80
7	-	2808.33	1807.00	4492.12	0.30	4468.05	11.72
8	-	2679.09	1808.17	4244.17	0.28	4138.37	10.90
9	-	2851.57	1807.44	4420.98	0.30	4346.04	11.12
10	-	1757.87	1800.94	2784.52	0.50	2808.98	19.56
11	-	1755.31	1802.93	2803.57	0.47	2697.53	18.30
12	-	1785.08	1803.73	2766.83	0.48	2711.86	19.05
13	-	2455.69	1804.00	3807.04	0.67	3725.28	24.57
14	-	2327.19	1800.84	3764.00	0.65	3627.46	24.01
15	-	2511.15	1802.13	3794.70	0.67	3713.79	24.23
16	-	3346.91	1806.00	5071.53	0.81	4965.54	27.94
17	-	3140.13	1805.47	4908.17	0.79	4761.66	26.83
18	-	3309.58	1804.77	4803.11	0.81	4764.53	27.61
19	-	1076.52	1802.82	**211.41**	0.09	**211.41**	4.07
20	-	1058.27	1800.65	**322.15**	0.13	**322.15**	5.40
21	-	1284.08	1800.59	**378.38**	0.12	**378.38**	5.71
22	-	3294.17	1800.60	5188.72	0.31	5025.73	12.95
23	-	2965.40	1801.51	4778.46	0.33	4624.53	13.88
24	-	3415.54	1800.73	5113.41	0.31	5027.05	13.74
25	-	4844.88	1800.45	7775.31	0.48	7684.93	18.39
26	-	4456.49	1800.69	7377.70	0.50	7269.29	19.11
27	-	4908.44	1800.88	7632.54	0.50	7627.20	19.10
28	-	2828.26	1802.62	4889.03	0.78	4774.13	30.90
29	-	2889.61	1800.72	4839.75	0.75	4754.50	29.61
30	-	2990.17	1800.72	4693.28	0.78	4612.43	30.87
31	-	3836.34	1801.18	6491.61	1.11	6438.60	39.72
32	-	3890.76	1801.44	6525.06	1.08	6355.12	39.17
33	-	4118.56	1800.55	6326.89	1.09	6328.61	40.00
34	-	5148.10	1800.09	8506.46	1.34	8373.49	45.05
35	-	5008.74	1804.80	8384.49	1.29	8247.37	44.10
36	-	5361.79	1800.16	8498.41	1.34	8471.31	45.72

Table 5.2: EPSP heuristic algorithms comparison

Chapter 6

Conclusion

In our thesis, we consider a network design problem arising in fiber optical network construction. The problem is formulated to an integer program. Four different models are covered: (1) Greedy Path Selection Problem (GPSP) (2) Benevolent Path Selection Problem (BPSP) (3) "a priori" Discounted Path Selection Problem (DPSP) and (4) Path Selection Problem with capacity expansion (EPSP). The optimal solution in each model is a collection of paths. Although the company is not considering the construction of all these paths, the solution of paths obtained can be used to estimate potential revenue structure and allow the company to make appropriate quotes for clients under consideration.

Each of the models GPSP, BPSP, DPSP and EPSP are analyzed in detail. GPSP is shown to be NP-hard and equivalent to Column Restricted Multi-dimensional Knapsack Problem (CMDKP). The transformation given in the proof is approximation preserving and hence all approximation (non-approximation) results for CMDKP can be translated into GPSP. We also identified some polynomially solvable special cases. In addition, different integer programming formulations are analyzed for their strength by means of the corresponding linear programming relaxations.

We developed various types of greedy heuristics for GPSP with different characteristics. Extensions of these heuristics within a semi-greedy framework are also considered. For experimental analysis, we constructed different classes of test problem showing approximate relationship between value of c_j's (cost) and a_j's (capacity usage) used in the model. This resulted in six different classes of problems: (1) random instances (2) positively correlated instances (3) negatively correlated instances (4) uniform instances (5) semi uniform instances and (6) PSP inspired instances. The greedy and semi-greedy algorithms are tested for each

of these classes. These algorithms produced very good solutions (average deviation from LP relaxation upper bound within 7.37%) in almost negligible computational effect.

We also used a general purpose ILP solver (CPLEX) with fixed time limit as a heuristic. This performed better than semi-greedy and greedy algorithm but at the expense of computational time on the average 600 times more than that of semi-greedy. A multi start version of a local search heuristic was also investigated but it was not competitive with CPLEX heuristic.

For BPSP we have obtained complexity results similar to that we discussed in the case of GPSP. In particular, we show that BPSP is equivalent to Capacity Cover Problem. Unlike GPSP, the BPSP on r-knot problem are solvable in polynomial time.

The greedy, semi-greedy, CPLEX and multi-start local search are extended to BPSP as well along with experimental analysis. We observed that BPSP is relatively simpler compared to GPSP.

One drawback of the BPSP model is that if a path selection covers all edges, it is optimal and including more paths will not affect the objective function value. However, for the purpose of our original problem, it is useful to select as many paths as possible. To accommodate this, we introduced a lexicographic version of the problem which can also be formulated as an integer linear programming problem.

Another model considered in this thesis is the discounted path selection model. We briefly discuss "discount after selection" model followed by extensive analysis of "a priori" discounted model (DPSP). We developed greedy, semi-greedy and multi-start versions of heuristics for this problem along with CPLEX based heuristic.

If CPLEX is used without specifying a starting solution, it could not produce any feasible solution (except zero solution) within half an hour. However, when the greedy solution (which uses almost negligible computational time) is provided as a starting solution, it produced solutions better than multi-start version of the local search heuristic.

Finally, we analyzed the capacity expansion problem and showed that it is also NP-hard. Computational results with CPLEX based heuristics are provided.

Our experimental analysis produced some interesting outcome. Note that GPSP is a special case of the multi-dimensional knapsack problem, which is well studied using various heuristics, including variations of flip and 2-exchange based heuristics. However, when restricted to GPSP, such heuristics were not competitive with CPLEX used on a heuristic with time limit. This may be because we used the most recent version of CPLEX while

studies on multi-dimensional knapsack problem are somewhat old. Further, it is possible that the restriction that nonzero elements in each column are the same may have some role to play in such impressive performance of CPLEX.

We did not consider theoretical analysis of heuristics for these problems and it is left as a topic for future investigation. Since the maximum clique problem is a special case of GPSP, interesting results with nice performance ratios may be difficult. However, we believe domination analysis could be used to obtain interesting theoretical results.

Appendix A

Computational Results for GPSP: other types

		AVE			MAX			MIN		
n	m	Obj	Gap	Time	Obj	Gap	Time	Obj	Gap	Time
1000	100	186587	0.43	3605.15	202610	0.49	3606.69	171017	0.35	3604.38
1000	150	274658	0.59	3605.74	288663	0.66	3606.72	255968	0.46	3604.72
1000	200	361009	0.66	3606.03	390925	0.71	3606.88	339244	0.60	3605.38
1000	250	444566	0.72	3605.86	476517	0.77	3607.06	390276	0.66	3605.38
1000	300	542272	0.85	3612.00	586076	0.90	3617.25	507592	0.76	3608.75
2000	200	724985	0.37	3607.35	738663	0.39	3608.41	701297	0.32	3605.16
2000	300	1086669	0.47	3606.30	1125257	0.49	3609.59	1066562	0.46	3604.13
2000	400	1427765	0.52	3615.98	1499139	0.56	3623.59	1312204	0.49	3602.56
2000	500	1757717	0.58	3603.29	1867167	0.62	3605.97	1605451	0.54	3602.22
2000	600	2092842	0.65	3612.59	2235938	0.68	3624.38	2021612	0.59	3602.44
5000	500	4564144	0.27	3603.36	4665680	0.29	3604.72	4442887	0.25	3602.56
5000	750	6829932	0.34	3602.86	7008941	0.37	3603.09	6676311	0.31	3602.41
5000	1000	9087050	0.38	3603.14	9375483	0.40	3603.31	8770072	0.35	3602.97
5000	1250	11247795	0.43	3602.78	11568992	0.45	3603.56	11051914	0.41	3601.94
5000	1500	13553428	0.47	3602.03	13819853	0.49	3602.81	13026055	0.43	3601.38

Table A.1: GPSP - the results of CPLEX on Test bed W

		AVE			MAX			MIN		
n	m	Obj	Gap	Time	Obj	Gap	Time	Obj	Gap	Time
1000	100	28012	0.17	3605.23	28514	0.19	3606.43	27457	0.15	3604.57
1000	150	27853	0.27	3605.87	28581	0.31	3608.62	26889	0.21	3604.06
1000	200	27803	0.33	3607.70	28070	0.39	3609.75	27614	0.28	3604.32
1000	250	27598	0.44	3607.74	28275	0.50	3612.54	26966	0.38	3605.26
1000	300	27405	0.47	3612.58	27854	0.51	3619.38	26783	0.41	3606.09
2000	200	96526	0.19	3607.08	97077	0.22	3614.66	96056	0.14	3604.30
2000	300	95975	0.28	3612.54	98308	0.33	3632.85	94864	0.25	3602.64
2000	400	95787	0.33	3625.87	98375	0.38	3664.95	93063	0.27	3603.94
2000	500	94485	0.36	3619.24	96382	0.40	3635.81	91633	0.35	3609.19
2000	600	95078	0.44	3626.71	96394	0.46	3631.88	92753	0.41	3620.69
5000	500	543957	0.17	3612.16	545886	0.18	3627.09	539698	0.15	3603.19
5000	750	541900	0.22	3614.94	547689	0.23	3629.69	537331	0.21	3602.81
5000	1000	536238	0.27	3603.37	540547	0.31	3606.27	532919	0.22	3602.33
5000	1250	539791	0.31	3603.19	545587	0.33	3604.39	537591	0.31	3602.55
5000	1500	535720	0.35	3603.24	541657	0.37	3604.31	531889	0.32	3602.64

Table A.2: GPSP - the results of CPLEX on Test bed R

		AVE			MAX			MIN		
n	m	Obj	Gap	Time	Obj	Gap	Time	Obj	Gap	Time
1000	100	19336	0.44	3603.85	19576	0.50	3604.06	19132	0.38	3603.50
1000	150	19511	0.74	3602.99	20317	0.78	3603.13	19045	0.72	3602.88
1000	200	18797	0.99	3602.64	19196	1.11	3602.81	18464	0.92	3602.38
1000	250	18828	1.28	3602.61	19096	1.38	3603.31	18486	1.09	3602.44
1000	300	18604	1.51	3602.94	18903	1.61	3604.13	18251	1.43	3602.25
2000	200	65285	0.45	3602.28	66193	0.49	3602.50	64208	0.41	3602.13
2000	300	64199	0.68	3608.34	64889	0.70	3630.81	63317	0.65	3601.94
2000	400	62497	0.89	3602.35	63914	0.94	3603.75	61764	0.84	3601.31
2000	500	62405	1.07	3603.80	64545	1.12	3605.06	61060	1.00	3602.31
2000	600	62557	1.33	3603.29	63998	1.37	3604.69	61480	1.29	3601.75
5000	500	358248	0.50	3608.24	359632	0.52	3613.00	357235	0.48	3601.31
5000	750	353240	0.72	3603.33	354715	0.74	3606.69	351409	0.68	3601.69
5000	1000	350292	0.87	3603.34	354086	0.90	3607.19	347518	0.80	3601.44
5000	1250	350077	1.01	3605.92	351682	1.21	3612.19	346803	0.94	3601.44
5000	1500	346777	1.18	3601.76	349762	1.29	3602.69	341435	1.03	3601.25

Table A.3: GPSP - the results of CPLEX on Test bed U

		AVE			MAX			MIN		
n	m	Obj	Gap	Time	Obj	Gap	Time	Obj	Gap	Time
1000	100	45780	0.42	3604.00	46243	0.49	3604.22	45360	0.36	3603.59
1000	150	45933	0.58	3603.96	46533	0.66	3604.59	45360	0.45	3603.56
1000	200	45292	0.69	3603.80	45720	0.74	3604.16	44853	0.65	3603.50
1000	250	45067	0.88	3603.43	45832	0.97	3604.00	44340	0.73	3603.00
1000	300	44746	0.95	3604.07	45322	1.12	3604.19	43961	0.76	3603.94
2000	200	157383	0.41	3602.95	157959	0.46	3603.03	156850	0.34	3602.88
2000	300	156090	0.53	3609.32	157650	0.60	3611.69	154422	0.48	3607.31
2000	400	154458	0.63	3607.20	157559	0.67	3612.00	153279	0.61	3603.81
2000	500	153008	0.72	3615.34	156984	0.82	3622.09	149366	0.61	3604.38
2000	600	153897	0.84	3613.64	155134	0.87	3618.78	151316	0.82	3604.53
5000	500	880364	0.33	3618.68	884306	0.37	3632.88	874740	0.30	3603.50
5000	750	873993	0.44	3602.54	881000	0.46	3604.00	868033	0.42	3602.09
5000	1000	866034	0.53	3603.20	869161	0.58	3603.81	862593	0.48	3602.13
5000	1250	869826	0.57	3602.63	874956	0.61	3603.25	865386	0.52	3602.09
5000	1500	862836	0.66	3602.39	871594	0.72	3603.44	856780	0.55	3601.31

Table A.4: GPSP - the results of CPLEX on Test bed S

		AVE			MAX			MIN		
n	m	Obj	Gap	Time	Obj	Gap	Time	Obj	Gap	Time
1000	100	31541	0.06	3606.12	32224	0.08	3609.00	30969	0.04	3604.06
1000	150	31666	0.08	3605.68	32193	0.10	3607.56	30670	0.07	3602.19
1000	200	31774	0.11	3606.09	32153	0.14	3606.69	31345	0.09	3605.56
1000	250	31577	0.12	3608.65	32352	0.15	3615.94	30924	0.11	3604.88
1000	300	31378	0.16	3622.56	31985	0.19	3638.25	30685	0.14	3608.50
2000	200	109579	0.07	3607.70	110857	0.08	3616.94	108283	0.06	3605.13
2000	300	108983	0.10	3629.41	110279	0.11	3639.75	107660	0.07	3610.69
2000	400	108669	0.11	3624.84	110789	0.13	3651.38	106374	0.10	3606.06
2000	500	107958	0.12	3628.68	110036	0.14	3662.75	105335	0.11	3606.69
2000	600	108261	0.14	3641.84	109435	0.16	3656.81	106117	0.11	3633.25
5000	500	614575	0.07	3640.20	616005	0.08	3646.81	612261	0.07	3635.19
5000	750	614822	0.09	3639.29	619032	0.09	3654.56	607909	0.08	3615.88
5000	1000	608546	0.10	3636.67	612629	0.10	3645.38	605070	0.09	3630.19
5000	1250	612281	0.12	3651.32	618247	0.13	3686.88	609931	0.10	3610.63
5000	1500	609234	0.12	3644.75	613116	0.13	3675.94	602635	0.10	3605.50

Table A.5: GPSP - the results of CPLEX on Test bed N

		AVE		MAX		MIN	
n	m	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time
1000	100	2.69	0.00	3.01	0.00	2.51	0.00
1000	150	3.05	0.00	3.77	0.00	2.64	0.00
1000	200	2.99	0.00	3.69	0.00	2.72	0.00
1000	250	3.09	0.00	3.34	0.02	2.64	0.00
1000	300	3.39	0.00	3.92	0.00	2.98	0.00
2000	200	2.01	0.00	2.31	0.00	1.75	0.00
2000	300	2.25	0.00	2.44	0.00	1.95	0.00
2000	400	2.36	0.00	2.46	0.00	2.20	0.00
2000	500	2.55	0.00	2.93	0.02	2.29	0.00
2000	600	2.88	0.01	3.17	0.02	2.56	0.00
5000	500	1.71	0.00	1.81	0.02	1.65	0.00
5000	750	1.74	0.01	1.82	0.02	1.65	0.00
5000	1000	1.80	0.01	1.93	0.02	1.63	0.00
5000	1250	1.94	0.01	2.10	0.02	1.73	0.00
5000	1500	2.00	0.01	2.13	0.02	1.81	0.01

Table A.6: GPSP - the results of greedy algorithm on Test bed W

		AVE		MAX		MIN	
n	m	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time
1000	100	1.66	0.00	1.96	0.01	1.35	0.00
1000	150	1.90	0.00	2.18	0.00	1.50	0.00
1000	200	2.53	0.00	2.77	0.00	2.17	0.00
1000	250	2.23	0.00	2.53	0.00	2.03	0.00
1000	300	2.48	0.00	2.78	0.02	2.29	0.00
2000	200	1.41	0.00	1.72	0.00	1.15	0.00
2000	300	1.49	0.00	1.62	0.00	1.32	0.00
2000	400	1.70	0.00	1.90	0.00	1.41	0.00
2000	500	1.94	0.00	2.34	0.00	1.64	0.00
2000	600	2.05	0.00	2.38	0.01	1.81	0.00
5000	500	1.18	0.01	1.33	0.02	1.10	0.00
5000	750	1.30	0.01	1.37	0.02	1.20	0.00
5000	1000	1.33	0.02	1.63	0.02	1.13	0.01
5000	1250	1.40	0.01	1.45	0.02	1.32	0.00
5000	1500	1.59	0.02	1.71	0.02	1.50	0.02

Table A.7: GPSP - the results of greedy algorithm on Test bed R

		AVE		MAX		MIN	
n	m	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time
1000	100	5.17	0.00	6.21	0.01	4.86	0.00
1000	150	5.55	0.00	6.29	0.00	4.95	0.00
1000	200	6.17	0.00	6.88	0.00	4.97	0.00
1000	250	6.73	0.00	7.11	0.00	5.96	0.00
1000	300	7.15	0.00	7.88	0.02	6.60	0.00
2000	200	4.30	0.00	4.62	0.01	3.98	0.00
2000	300	4.65	0.00	4.81	0.00	4.53	0.00
2000	400	5.34	0.00	5.62	0.02	4.87	0.00
2000	500	5.79	0.00	6.22	0.00	5.21	0.00
2000	600	6.33	0.01	6.77	0.02	6.11	0.00
5000	500	3.53	0.00	3.68	0.00	3.36	0.00
5000	750	3.79	0.00	4.00	0.02	3.58	0.00
5000	1000	3.99	0.01	4.25	0.02	3.87	0.00
5000	1250	4.01	0.01	4.12	0.02	3.78	0.00
5000	1500	4.21	0.02	4.46	0.02	3.98	0.02

Table A.8: GPSP - the results of greedy algorithm on Test bed U

		AVE		MAX		MIN	
n	m	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time
1000	100	3.04	0.00	3.46	0.00	2.32	0.00
1000	150	3.33	0.00	3.68	0.00	2.97	0.00
1000	200	4.15	0.00	4.79	0.00	3.74	0.00
1000	250	3.66	0.00	4.18	0.00	3.29	0.00
1000	300	3.72	0.00	4.63	0.00	3.18	0.00
2000	200	2.44	0.00	2.56	0.00	2.24	0.00
2000	300	2.66	0.00	3.57	0.00	2.30	0.00
2000	400	2.93	0.00	3.60	0.00	2.39	0.00
2000	500	3.18	0.00	3.60	0.01	2.79	0.00
2000	600	3.50	0.01	3.64	0.02	3.41	0.00
5000	500	2.04	0.01	2.19	0.01	1.88	0.00
5000	750	2.16	0.01	2.31	0.02	2.00	0.00
5000	1000	2.26	0.02	2.49	0.02	2.13	0.01
5000	1250	2.42	0.02	2.68	0.02	2.25	0.02
5000	1500	2.84	0.02	3.05	0.02	2.61	0.01

Table A.9: GPSP - the results of greedy algorithm on Test bed S

		AVE		MAX		MIN	
n	m	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time
1000	100	0.73	0.00	0.94	0.00	0.62	0.00
1000	150	0.94	0.00	1.21	0.00	0.82	0.00
1000	200	0.96	0.00	1.23	0.01	0.77	0.00
1000	250	1.05	0.00	1.20	0.00	0.81	0.00
1000	300	1.15	0.00	1.36	0.00	1.05	0.00
2000	200	0.62	0.00	0.71	0.00	0.52	0.00
2000	300	0.72	0.00	0.83	0.02	0.59	0.00
2000	400	0.69	0.00	0.76	0.02	0.59	0.00
2000	500	0.81	0.00	0.91	0.02	0.71	0.00
2000	600	1.03	0.00	1.20	0.00	0.92	0.00
5000	500	0.56	0.01	0.61	0.02	0.51	0.00
5000	750	0.53	0.01	0.59	0.02	0.45	0.00
5000	1000	0.59	0.01	0.74	0.02	0.51	0.00
5000	1250	0.63	0.01	0.69	0.02	0.58	0.00
5000	1500	0.66	0.02	0.69	0.02	0.60	0.02

Table A.10: GPSP - the results of greedy algorithm on Test bed N

		AVE		MAX		MIN	
n	m	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time
1000	100	2.27	0.01	2.54	0.02	2.07	0.01
1000	150	2.72	0.02	3.21	0.03	2.19	0.01
1000	200	2.67	0.02	3.32	0.02	2.36	0.01
1000	250	2.56	0.02	2.78	0.02	2.23	0.01
1000	300	2.93	0.02	3.15	0.02	2.77	0.01
2000	200	1.81	0.04	2.18	0.05	1.60	0.03
2000	300	1.91	0.04	2.03	0.06	1.74	0.03
2000	400	2.07	0.05	2.24	0.05	1.79	0.05
2000	500	2.29	0.05	2.56	0.06	2.14	0.05
2000	600	2.60	0.05	2.92	0.06	2.27	0.05
5000	500	1.54	0.19	1.68	0.20	1.46	0.17
5000	750	1.55	0.24	1.60	0.25	1.48	0.22
5000	1000	1.67	0.27	1.76	0.31	1.48	0.25
5000	1250	1.73	0.29	1.82	0.30	1.61	0.28
5000	1500	1.81	0.33	1.93	0.37	1.68	0.30

Table A.11: GPSP - the results of semi-greedy algorithm on Test bed W

		AVE		MAX		MIN	
n	m	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time
1000	100	1.30	0.01	1.43	0.01	1.10	0.01
1000	150	1.57	0.01	1.80	0.01	1.38	0.01
1000	200	1.99	0.01	2.24	0.01	1.67	0.00
1000	250	1.93	0.01	2.33	0.02	1.63	0.00
1000	300	2.06	0.02	2.22	0.03	1.82	0.01
2000	200	1.21	0.03	1.51	0.05	1.02	0.03
2000	300	1.36	0.04	1.54	0.05	1.22	0.03
2000	400	1.56	0.05	1.85	0.05	1.29	0.05
2000	500	1.60	0.05	1.80	0.05	1.39	0.05
2000	600	1.83	0.06	2.14	0.06	1.59	0.05
5000	500	1.06	0.18	1.18	0.19	1.00	0.17
5000	750	1.19	0.21	1.30	0.22	1.06	0.20
5000	1000	1.20	0.24	1.40	0.25	1.04	0.23
5000	1250	1.30	0.28	1.34	0.30	1.22	0.27
5000	1500	1.45	0.31	1.57	0.33	1.38	0.30

Table A.12: GPSP - the results of semi-greedy algorithm on Test bed R

		AVE		MAX		MIN	
n	m	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time
1000	100	4.48	0.01	5.00	0.01	4.18	0.01
1000	150	4.73	0.01	5.17	0.03	4.26	0.00
1000	200	5.58	0.01	6.35	0.02	4.92	0.00
1000	250	5.98	0.02	6.43	0.02	5.49	0.01
1000	300	6.56	0.01	6.93	0.02	6.24	0.00
2000	200	4.04	0.03	4.31	0.03	3.58	0.03
2000	300	4.35	0.04	4.52	0.05	4.16	0.03
2000	400	5.10	0.03	5.49	0.05	4.78	0.03
2000	500	5.29	0.04	5.64	0.05	5.06	0.03
2000	600	5.84	0.05	6.41	0.05	5.40	0.05
5000	500	3.40	0.17	3.70	0.19	3.13	0.16
5000	750	3.54	0.19	3.72	0.20	3.47	0.19
5000	1000	3.82	0.22	4.18	0.23	3.57	0.20
5000	1250	3.83	0.24	3.92	0.25	3.71	0.23
5000	1500	4.00	0.28	4.33	0.30	3.74	0.26

Table A.13: GPSP - the results of semi-greedy algorithm on Test bed U

		AVE		MAX		MIN	
n	m	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time
1000	100	2.56	0.01	3.06	0.01	1.88	0.01
1000	150	2.85	0.01	3.28	0.01	2.35	0.01
1000	200	3.50	0.02	4.07	0.02	3.19	0.01
1000	250	3.32	0.02	3.98	0.03	2.98	0.01
1000	300	3.27	0.02	4.09	0.02	2.63	0.02
2000	200	2.23	0.03	2.27	0.05	2.20	0.03
2000	300	2.41	0.04	3.10	0.05	2.11	0.03
2000	400	2.62	0.04	3.00	0.05	2.29	0.03
2000	500	2.94	0.05	3.26	0.05	2.53	0.05
2000	600	3.17	0.05	3.54	0.06	2.82	0.05
5000	500	1.84	0.18	2.01	0.19	1.65	0.17
5000	750	2.03	0.21	2.18	0.22	1.88	0.20
5000	1000	2.09	0.25	2.46	0.25	1.95	0.25
5000	1250	2.30	0.28	2.48	0.28	2.15	0.26
5000	1500	2.56	0.30	2.68	0.31	2.43	0.28

Table A.14: GPSP - the results of semi-greedy algorithm on Test bed S

		AVE		MAX		MIN	
n	m	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time
1000	100	0.66	0.02	0.85	0.02	0.56	0.01
1000	150	0.75	0.01	0.96	0.01	0.63	0.01
1000	200	0.83	0.01	1.03	0.01	0.67	0.01
1000	250	0.91	0.02	1.04	0.02	0.75	0.01
1000	300	0.96	0.02	1.06	0.02	0.87	0.01
2000	200	0.51	0.03	0.57	0.05	0.43	0.03
2000	300	0.64	0.03	0.74	0.03	0.52	0.03
2000	400	0.60	0.04	0.61	0.05	0.58	0.03
2000	500	0.73	0.05	0.82	0.06	0.65	0.05
2000	600	0.90	0.05	1.01	0.06	0.80	0.05
5000	500	0.50	0.19	0.54	0.20	0.44	0.19
5000	750	0.49	0.22	0.53	0.23	0.42	0.22
5000	1000	0.53	0.25	0.64	0.26	0.44	0.25
5000	1250	0.58	0.29	0.62	0.30	0.52	0.28
5000	1500	0.60	0.33	0.64	0.36	0.58	0.31

Table A.15: GPSP - the results of semi-greedy algorithm on Test bed N

		AVE		MAX		MIN	
n	m	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time
1000	100	1.40	4.12	1.49	5.08	1.27	3.54
1000	150	1.70	5.12	1.89	5.38	1.50	4.90
1000	200	1.79	6.03	2.08	6.35	1.57	5.62
1000	250	1.84	6.78	2.01	7.67	1.63	6.30
1000	300	2.07	7.92	2.17	8.28	1.99	7.16
2000	200	1.14	22.42	1.30	23.63	0.99	20.97
2000	300	1.31	28.64	1.41	30.25	1.24	25.86
2000	400	1.39	35.73	1.42	38.14	1.35	33.76
2000	500	1.52	45.41	1.58	49.69	1.45	41.71
2000	600	1.73	54.03	1.88	60.84	1.64	49.76
5000	500	0.95	364.88	0.97	404.38	0.92	315.70
5000	750	0.99	510.91	1.04	587.50	0.88	415.88
5000	1000	1.09	672.87	1.12	744.45	1.05	620.63
5000	1250	1.16	859.60	1.22	942.26	1.12	751.95
5000	1500	1.21	1016.13	1.26	1072.99	1.16	933.05

Table A.16: GPSP - the results of multi-start local search algorithm on Test bed W

		AVE		MAX		MIN	
n	m	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time
1000	100	1.04	3.34	1.12	3.60	0.98	3.07
1000	150	1.41	4.32	1.56	4.53	1.22	4.14
1000	200	1.64	5.29	1.92	5.68	1.44	5.07
1000	250	1.64	5.95	1.92	6.19	1.48	5.68
1000	300	1.79	7.02	1.88	7.67	1.65	6.05
2000	200	1.01	17.79	1.24	18.75	0.83	17.29
2000	300	1.16	22.94	1.27	24.65	1.00	21.61
2000	400	1.37	28.86	1.51	30.08	1.16	27.78
2000	500	1.35	35.46	1.59	40.39	1.16	31.89
2000	600	1.55	41.66	1.73	43.31	1.40	37.14
5000	500	0.91	226.10	1.01	254.16	0.87	196.59
5000	750	1.06	303.89	1.11	353.78	0.96	267.24
5000	1000	1.09	380.95	1.27	415.66	0.94	354.15
5000	1250	1.15	460.03	1.19	537.83	1.12	404.70
5000	1500	1.31	527.88	1.43	579.29	1.26	465.74

Table A.17: GPSP - the results of multi-start local search algorithm on Test bed R

		AVE		MAX		MIN	
n	m	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time
1000	100	3.07	5.67	3.34	5.93	2.81	5.41
1000	150	3.38	6.67	3.74	7.04	3.21	6.07
1000	200	3.94	7.69	4.37	7.92	3.67	7.49
1000	250	4.32	8.64	4.73	9.28	4.12	7.66
1000	300	4.69	9.54	5.15	9.73	4.33	9.27
2000	200	2.92	29.80	3.15	32.49	2.70	28.36
2000	300	3.19	36.89	3.44	40.62	2.98	33.45
2000	400	3.65	46.07	3.95	49.72	3.28	41.92
2000	500	4.03	54.30	4.11	58.45	3.87	47.45
2000	600	4.34	66.14	4.72	69.79	3.98	60.14
5000	500	2.49	320.89	2.58	374.81	2.43	292.30
5000	750	2.69	465.23	2.79	506.44	2.60	413.26
5000	1000	2.82	560.02	2.94	796.04	2.69	437.27
5000	1250	2.91	704.14	3.06	875.85	2.75	633.64
5000	1500	3.00	895.27	3.06	1055.70	2.95	748.60

Table A.18: GPSP - the results of multi-start local search algorithm on Test bed U

		AVE		MAX		MIN	
n	m	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time
1000	100	1.77	4.50	2.07	5.02	1.37	4.04
1000	150	2.08	5.24	2.35	5.56	1.86	5.08
1000	200	2.51	6.32	2.69	6.82	2.35	6.02
1000	250	2.64	7.14	3.05	7.58	2.37	6.68
1000	300	2.64	8.08	3.07	8.66	2.15	7.44
2000	200	1.68	22.01	1.73	22.45	1.60	21.39
2000	300	1.86	28.25	2.25	28.89	1.67	27.55
2000	400	2.05	34.64	2.32	36.30	1.75	33.27
2000	500	2.33	41.47	2.47	46.13	2.02	38.77
2000	600	2.46	47.40	2.56	51.51	2.37	41.65
5000	500	1.43	268.56	1.53	284.45	1.36	246.51
5000	750	1.61	372.39	1.74	452.63	1.42	317.27
5000	1000	1.69	441.34	1.92	535.63	1.57	406.08
5000	1250	1.87	526.81	2.03	613.39	1.73	469.09
5000	1500	1.99	608.78	2.10	693.20	1.90	555.91

Table A.19: GPSP - the results of multi-start local search algorithm on Test bed S

		AVE		MAX		MIN	
n	m	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time
1000	100	0.48	3.04	0.65	3.25	0.41	2.77
1000	150	0.61	3.90	0.76	4.18	0.55	3.59
1000	200	0.67	4.80	0.88	5.06	0.52	4.59
1000	250	0.72	5.72	0.77	6.22	0.63	5.26
1000	300	0.83	6.60	0.96	7.08	0.76	6.10
2000	200	0.39	15.43	0.44	16.60	0.30	14.34
2000	300	0.53	19.29	0.60	20.28	0.43	18.78
2000	400	0.51	23.98	0.53	24.85	0.48	23.14
2000	500	0.62	29.77	0.70	32.25	0.55	28.19
2000	600	0.77	35.38	0.89	39.33	0.68	33.04
5000	500	0.39	170.65	0.43	188.98	0.34	160.20
5000	750	0.41	213.28	0.46	236.73	0.34	193.89
5000	1000	0.45	280.13	0.55	296.81	0.38	253.16
5000	1250	0.51	334.83	0.54	355.04	0.47	322.64
5000	1500	0.54	391.47	0.58	424.13	0.49	373.36

Table A.20: GPSP - the results of multi-start local search algorithm on Test bed N

	Average		Max		Min		
Alg	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time	#Opt Inst
CPLEX	0.61	3606.298	1.16	3624.38	0.28	3601.380	0
GREEDY-4	2.43	**0.004**	3.92	**0.02**	1.63	**0.000**	**0**
SEMIGREEDY-4-20-50	2.14	0.108	3.32	0.37	1.46	0.010	**0**
MULLS-100	**1.42**	242.705	**2.17**	1072.99	**0.88**	3.539	**0**

Table A.21: GPSP - the summary results of algorithms on Test bed W

	Average		Max		Min		
Alg	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time	#Opt Inst
CPLEX	0.43	3611.165	0.82	3664.95	0.17	3602.330	0
GREEDY-4	1.75	**0.005**	2.78	**0.02**	1.10	**0.000**	**0**
SEMIGREEDY-4-20-50	1.51	0.100	2.33	0.33	1.00	**0.000**	**0**
MULLS-100	**1.30**	138.099	**1.92**	579.29	**0.83**	3.070	**0**

Table A.22: GPSP - the summary results of algorithms on Test bed R

	Average		Max		Min		
Alg	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time	#Opt Inst
CPLEX	0.94	3603.845	1.69	3630.81	0.43	3601.250	0
GREEDY-4	5.11	**0.003**	7.88	**0.02**	3.36	**0.000**	**0**
SEMIGREEDY-4-20-50	4.70	0.090	6.93	0.30	3.13	**0.000**	**0**
MULLS-100	**3.43**	214.465	**5.15**	1055.70	**2.43**	5.406	**0**

Table A.23: GPSP - the summary results of algorithms on Test bed U

	Average		Max		Min		
Alg	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time	#Opt Inst
CPLEX	0.70	3606.477	1.35	3632.88	0.31	3601.310	0
GREEDY-4	2.95	**0.005**	4.79	**0.02**	1.88	**0.000**	**0**
SEMIGREEDY-4-20-50	2.65	0.100	4.09	0.31	1.65	0.010	**0**
MULLS-100	**2.04**	161.527	**3.07**	693.20	**1.36**	4.045	**0**

Table A.24: GPSP - the summary results of algorithms on Test bed S

	Average		Max		Min		
Alg	Dev(%)	Time	Dev(%)	Time	Dev(%)	Time	#Opt Inst
CPLEX	0.24	3626.253	0.50	3686.88	0.10	3602.190	0
GREEDY-4	0.78	**0.005**	1.36	**0.02**	0.45	**0.000**	**0**
SEMIGREEDY-4-20-50	0.68	0.104	1.06	0.36	0.42	0.010	**0**
MULLS-100	**0.56**	102.552	**0.96**	424.13	**0.30**	2.775	**0**

Table A.25: GPSP - the summary results of algorithms on Test bed N

Bibliography

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[2] B. Alidaee, G. Kochenberger, K. Lewis, M. Lewis, and H. Wang. A new approach for modeling and solving set packing problems. *European Journal of Operational Research*, 186(2):504–512, April 2008.

[3] N. Bansal, A. Chakrabarti, A. Epstein, and B. Schieber. A quasi-PTAS for unsplittable flow on line graphs. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing - STOC '06*, page 721, New York, New York, USA, 2006. ACM Press.

[4] N. Bansal and Z. Friggstad. A logarithmic approximation for unsplittable flow on line graphs. *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 702–709, 2009.

[5] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *J. ACM*, 48(5):1069–1090, Sep 2001.

[6] C. Barnhart, C.A. Hane, and P.H. Vance. Integer multicommodity flow problems. In WilliamH. Cunningham, S.Thomas McCormick, and Maurice Queyranne, editors, *Integer Programming and Combinatorial Optimization*, volume 1084 of *Lecture Notes in Computer Science*, pages 58–71. Springer Berlin Heidelberg, 1996.

[7] A. Baveja and A. Srinivasan. Approximating low-congestion routing and column-restricted packing problems. *Information Processing Letters*, 74(1-2):19–25, April 2000.

[8] R. Bellman. Dynamic programming. *Princeton University Press*, 1957.

[9] P. Bonsma, J. Schulz, and A. Wiese. A Constant Factor Approximation Algorithm for Unsplittable Flow on Paths. *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 47–56, October 2011.

[10] S. Boussier, M. Vasquez, Y. Vimont, S. Hanafi, and P. Michelon. A multi-level search strategy for the 01 Multidimensional Knapsack Problem. *Discrete Applied Mathematics*, 158(2):97–109, jan 2010.

[11] A.V. Cabot. An enumeration algorithm for knapsack problems. *Operations Research*, 18:306–311, 1970.

[12] G. Calinescu, A. Chakrabarti, H. Karloff, and Y. Rabani. Improved approximation algorithms for resource allocation, 2002.

[13] A. Chakrabarti, C. Chekuri, A. Gupta, and A. Kumar. Approximation Algorithms for the Unsplittable Flow Problem. *Algorithmica*, 47(1):53–78, August 2006.

[14] D. Chakrabarty, E. Grant, and J. Könemann. On column-restricted and priority covering integer programs. In *Proceedings of the 14th international conference on Integer Programming and Combinatorial Optimization*, IPCO'10, pages 355–368, Berlin, Heidelberg, 2010. Springer-Verlag.

[15] A.K. Chandra, D.S. Hirshberg, and C.K. Wong. Approximate algorithms for some generalized knapsack problems. *Theoretical Computer Science*, 3:293–304, 1976.

[16] C. Chekuri, A. Ene, and N. Korula. Unsplittable Flow in Paths and Trees and Column-Restricted Packing Integer Programs. *in Proceedings of the 12th International Workshop (APPROX 09) and 13th International Workshop (RANDOM 09) on Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, 5687:42–55, 2009.

[17] C. Chekuri, M. Mydlarz, and F.B. Shepherd. Multicommodity demand flow in a tree and packing integer programs. *ACM Trans. Algorithms*, 3(3), aug 2007.

[18] C. Chekuri, M. Mydlarz, F.B. Shepherd, B. Labs, M. Ave, and M. Hill. Multicommodity Demand Flow in a Tree (Extended Abstract). pages 410–425, 2003.

[19] M. Chrobak, G.J. Woeginger, K. Makino, and H. Xu. Caching is hard even in the fault model. 6346:195–206, 2010.

[20] P.C. Chu and J.E. Beasley. A Genetic Algorithm for the Multidimensional Knapsack Problem. 86:63–86, 1998.

[21] G. Cornujols. Combinatorial optimization: Packing and covering, 2000.

[22] F. Dammeyer and S. Voß. Dynamic tabu list management using the reverse elimination method. *Annals of Operations Research*, 41(2):29–46, jun 1993.

[23] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1):pp. 80–91, 1959.

[24] X. Delorme, X. Gandibleux, and J. Rodriguez. GRASP for set packing problems. *European Journal of Operational Research*, 153(3):564–580, March 2004.

[25] A Drexl. A simulated annealing approach to the multiconstraint zero-one knapsack problem. *Computing*, 40(1):1–8, mar 1988.

[26] A. Fréville. The multidimensional 01 knapsack problem: An overview. *European Journal of Operational Research*, 155(1):1–21, may 2004.

[27] A. Fréville and S. Hanafi. The Multidimensional 0-1 Knapsack ProblemBounds and Computational Aspects. *Annals of Operations Research*, 139(1):195–227, oct 2005.

[28] A. Freville and G. Plateau. An efficient preprocessing procedure for the multidimensional 01 knapsack problem. *Discrete Applied Mathematics*, 49(1-3):189–212, mar 1994.

[29] A.M. Frieze and M.R. Clarke. Approximation algorithms for the m-dimensional 0-1 knapsack problem: Worst-case and probablistic analyses. *European Journal of Operational Research*, 15:100–109, 1984.

[30] X. Gandibleux. An ant colony optimization inspired algorithm for the Set Packing Problem with application to railway infrastructure Outline of the ACO heuristic for the SPP. 2005.

[31] M.R. Garey and D.S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[32] G.V. Gens and E.V. Levner. Computational complexity of approximation algorithms for combinatorial problems. In Ji Bev, editor, *Mathematical Foundations of Computer Science 1979*, volume 74 of *Lecture Notes in Computer Science*, pages 292–300. Springer Berlin Heidelberg, 1979.

[33] P.C. Gilmore and R.E. Gomory. The Theory and Computation of Knapsack Functions. *Operations Research*, 14(6):1045–1074, nov 1966.

[34] F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8(1):156–166, 1977.

[35] J. Gottlieb. Permutation-based evolutionary algorithms for multidimensional knapsack problems. In *Proceedings of the 2000 ACM symposium on Applied computing - SAC '00*, pages 408–414, New York, New York, USA, 2000. ACM Press.

[36] Y. Guo, A. Lim, B. Rodrigues, and Y. Zhu. Heuristics for a bidding problem. *Comput. Oper. Res*, 33, 2006.

[37] M.M. Halldórsson. Approximations of weighted independent set and hereditary subset problems. *Journal of Graph A and Applications*, 4(1):1–16, 2000.

[38] J. Hastad. Clique is hard to approximate within n1- epsiv;. In *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*, pages 627–636, 1996.

[39] F.S. Hillier. Efficient Heuristic Procedures for Integer Linear Programming with an Interior. *Operations Research*, 17(4):600–637, jul 1969.

[40] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.

[41] S.G. Kolliopoulos and C. Stein. Approximating Disjoint-Path Problems Using Packing Integer Programs. 2004.

[42] S.G. Kolliopoulos. Approximating covering integer programs with multiplicity constraints. *Discrete Appl. Math*, 129:461–473, 2000.

[43] S.G. Kolliopoulos and C. Stein. Approximating Disjoint-Path Problems Using Greedy Algorithms and Packing Integer Programs. pages 153–168, 1998.

[44] S.G. Kolliopoulos and N.E. Young. Approximation algorithms for covering/packing integer programs, 2005.

[45] B. Korte and R. Schrader. On the existence of fast approximation schemes. *in: O.L. Mangasarian, R.R. Meyer, S.M. Robinson (Eds.), Nonlinear Programming, 4, Academic Press*, pages 415–437, 1980.

[46] M. Landete, J.F. Monge, and A.M. Rodríguez-Chía. Alternative formulations for the Set Packing Problem and their application to the Winner Determination Problem. *Annals of Operations Research*, January 2012.

[47] H.C. Lau and Y.G. Goh. An intelligent brokering system to support multi-agent web-based 4th-party logistics. In *Tools with Artificial Intelligence, 2002. (ICTAI 2002). Proceedings. 14th IEEE International Conference on*, pages 154–161, 2002.

[48] S. Leonardi, A. Marchetti-Spaccamela, and A. Vitaletti. Approximation algorithms for bandwidth and storage allocation problems under real time constraints. In *Proceedings of the 20th Conference on Foundations of Software Technology and Theoretical Computer Science*, FST TCS 2000, pages 409–420, London, UK, UK, 2000. Springer-Verlag.

[49] R. Loulou and E. Michaelides. New Greedy-Like Heuristics for the Multidimensional 0-1 Knapsack Problem. *Operations Research*, 27(6):1101–1114, nov 1979.

[50] M.J. Magazine and O. Oguz. A fully polynomial approximation algorithm for the 0-1 knapsack problem. *European Journal of Operational Research*, 8:270–273, 1981.

[51] G.B. Mathews. On the partition of numbers. *Proceedings of the London Mathematical Society*, 28:486–490, 1897.

[52] A. Merel, X. Gandibleux, and S. Demassey. A collaborative combination between column generation and ant colony optimization for solving set packing problems. In *9th Metaheuristics International Conference (MIC'11)*, Udine, Italy, jul 2011.

[53] M.W. Padberg. On the facial structure of set packing polyhedra. *Mathematical Programming*, 5(1):199–215, December 1973.

[54] C.A. Phillips, R.N. Uma, and J. Wein. Off-line admission control for general scheduling problems. In *Journal of Scheduling*, pages 879–888, 2000.

[55] A.P. Punnen. *The path selection problem and optimal pricing.* 2013.

[56] S. Senju and Y. Toyoda. An Approach to Linear Programming with 0-1 Variables. *Management Science*, 15(4):B–196–B–207, December 1968.

[57] A. Srinivasan. Improved approximation guarantees for packing and covering integer programs. *SIAM J. Comput*, 29:648–670, 1995.

[58] Y. Toyoda. A simplified algorithm for obtaining approximate solutions to zero-one programming problems. *Management Science*, 21(12):pp. 1417–1427, 1975.

[59] L. Trevisan. Non-approximability results for optimization problems on bounded degree instances, 2001.

[60] V.V. Vazirani. *Approximation Algorithms.* Springer-Verlag New York, Inc., New York, NY, USA, 2001.

[61] H.M. Weingartner and D.N. Ness. Methods for the solution of the multidimensional 0/1 knapsack problem. *Operations Research*, 15:83–103, 1967.

[62] P.J. Zwaneveld, L.G. Kroon, E. Romeijn, M. Salomon, and S. P.M. van Hoesel. Routing trains through railway stations: model formulation and algorithms. Open access publications from maastricht university, Maastricht University, 1996.