Clemson University
TigerPrints

All Dissertations | Dissertations

8-2011

METAHEURISTICS FOR HUB LOCATION MODELS

Ornurai Sangsawang
Clemson University, osangsawang@yahoo.com

Follow this and additional works at: https://tigerprints.clemson.edu/all_dissertations

Part of the Industrial Engineering Commons

Recommended Citation
Sangsawang, Ornurai, "METAHEURISTICS FOR HUB LOCATION MODELS" (2011). *All Dissertations*. 804.
https://tigerprints.clemson.edu/all_dissertations/804

This Dissertation is brought to you for free and open access by the Dissertations at TigerPrints. It has been accepted for inclusion in All Dissertations by an authorized administrator of TigerPrints. For more information, please contact kokeefe@clemson.edu.

METAHEURISTICS FOR HUB LOCATION MODELS

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Industrial Engineering

---

by
Ornurai Sangsawang
August 2011

Accepted by:
Dr. Mary E.Kurz, Committee Chair
Dr. Byung Rae Cho
Dr. William G. Ferrell
Dr. Scott J. Mason

ABSTRACT

In this research, we propose metaheuristics for solving two p-hub median problems. The first p-hub median problem, which is NP-hard, is the uncapacitated single p-hub median problem (USApHMP). In this problem, metaheuristics such as genetic algorithms, simulated annealing and tabu search, are applied in different types of representations. Caching is also applied to speed up computational time of the algorithms. The results clearly demonstrate that tabu search with a permutation solution representation, augmented with caching is the highest performing method, both in terms of solution quality and computational time among these algorithms for the USApHMP. We also investigate the performance of hybrid metaheuristics, formed by path-relinking augmentation of the three base algorithms (genetic algorithms, simulated annealing and tabu search). The results indicate that hybridrization with path-relinking improvees the performance of base algorithms except tabu search since a good base metaheuristic does not require path-relinking. For the second p-hub median problem, the NP-hard uncapacitated multiple p-hub median problem (UMApHMP), we proposed Multiple TS. We identify multiple nodes using the convex hull and methods derived from the tabu search for the USApMHP. We find optimal allocations using the Single Reallocation Exchange procedure, developed for the USApHMP. The results show that implementing tabu search with a geometric interpretation allows nearly all optimal solutions to be found.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Mary Beth Kurz for her great insights and invaluable guidance throughout research.

I also would like to thank my committee members Dr. Byung Rae Cho, Dr. William G. Ferrell, and Dr. Scott J. Mason for their suggestions to gain different perspective.

I thank the Royal Thai Government for the financial support. My deepest gratitude goes to my family for their care and love who have always supported and encouraged me. I am sure that the achievement would have not been possible without them.

TABLE OF CONTENTS

Page

TITLE PAGE i

ABSTRACT ii

ACKNOWLEDGEMENTS iii

LIST OF TABLES v

LIST OF FIGURES vi

CHAPTER

I. INTRODUCTION 1

Introduction 1
Literature Review and Background 4
Research Problem 19
Research Overview 19

II. METAHEURISTICS FOR THE USApHMP 20

Introduction 20
Metaheuristics 21
Genetic algorithms 27
Simulated Annealing 32
Tabu Search 35
Caching 40
Computational Experiments 42
Analysis 47
Conclusions 54

III. PATH RELINKING FOR HUB LOCATION 56

Introduction 56
Path Relinking approach 56
Path Relinking approach for the USApHMP 57
Computational experiment 62
Analysis 69

Table of Contents (Continued)

Page

Conclusion 73

IV. METAHEURISTICS FOR THE UMApHMP.. 75

Introduction.. 75
Metaheuristics.. 75
Computational experiment.. 85
Conclusion 88

V. CONCLUSIONS AND FUTURE RESEARCH 89

Conclusions.. 89
Future Research.. 90

APPENDICES 92

A: Red-black tree 92

REFERENCES 95

LIST OF TABLES

Table Page

2.1 Results from Ernst’s formulation for USApHMP 21
2.2 Decoding random keys 24
2.3 Set of neighborhoods generated from TABUHUB 38
2.4 Number of neighborhoods generated from Tabu search algorithm 39
2.5 Updating the cache list 42
2.6 Results of the SA 44
2.7 Results of the GA 45
2.8 Results of the TS 46
2.9 Wilcoxon signed rank test for the time of binary GA with (C) and without caching (nC) 48
2.10 Time comparisons between no caching (nC) and caching (C) algorithm for each algorithm 50
2.11 Gap comparisons 51
2.12 Time comparisons 51
2.13 Winner comparisons 51
3.1 Results of the Binary GAPR 63
3.2 Results of the Permutation GAPR 64
3.3 Results of the Binary SAPR 65
3.4 Results of the Permutation SAPR 66
3.5 Results of the Binary TSPR 67
3.6 Results of the permutation TSPR 68
3.7 Time and performance comparisons 70
3.8 The recommended algorithms comparisons 72
4.1 Result from mathematic programming for UMApHMP 76
4.2 Single Node Exchange procedure to seek the best set of multiple node 82
4.3 Numerical example: distance of selected multiple nodes 84
4.4 Numerical example: updating priorities of candidate multiple nodes 84
4.5 Computational results 87

LIST OF FIGURES

Figure Page

1.1 A Fully Connected Network 4
1.2 Hub and Spoke Network 4
1.3 Typical Hub Location Network 5
1.4 Single Allocation *p*-hub Median Network 8
1.5 Multiple Allocation *p*-Hub Median Network 8
2.1 Solution Represented 22
2.2 Genetic algorithm procedure 29
2.3 Cut and splice crossover for a binary GA 31
2.4 Single point crossover for a permutation GA 32
2.5 Simulated annealing procedure 33
2.6 Tabu search algorithm 36
2.7 Caching procedure 41
2.8 Gap of GA, SA, and TS 47
2.9 Computational time of GA, SA, and TS 48
3.1 Path Relinking 57
3.2 Path Relinking for USApHMP 58
3.3 Numerical example of Path relinking for hub location 59
3.4 Numerical example of Path relinking for hub location 2 60
3.5 PR-local search algorithms for USApHMP procedure 61
3.6 Gap of SAPR 69
3.7 Gap of GAPR 69
4.1 Solution representation for the UMA*p*HMP 78
4.2 Tabu Search for the UMA*p*HMP procedure 79
4.3 Generating initial multiple nodes from convex hull 80
4.4 Convex hull 81
4.5 Gap of MultipleTS 86
4.6 Computational time of MultipleTS 86

CHAPTER ONE

INTRODUCTION

1. Introduction

One of the many versions of network problems involves reducing the number of arcs along which flows can be made, from allowing all pair-wise flows in a fully connected network to allowing flows between only a subset of nodes. Flows from different cities, for example, such as packages, mail, or passengers, are collected at *hubs*, transferred between hubs along *hub links* in order to economically consolidate flows on the same route, and distributed to their destinations. ***Hub location*** is the name of the problem concerned with determining which nodes in a network are designated as hubs (facility location) and which non-hub nodes are connected to which hubs (allocation). In a *p*-hub problem, hubs are fully interconnected and the total number of hubs is fixed at *p*. Flow between an origin and a destination move from the origin to a hub node, possibly to a second hub node along a hub link, and then to the destination node. In a single allocation hub problem, each non-hub node must be allocated to only one hub node. In multiple allocation hub problems, each non-hub node may be allocated to more than one hub, depending on to which node the flows are destined. Hub location has been used in several applications, including for the geography and design of facility locations, transportation, airport, postal delivery, trucking industry, freight, distribution, telecommunication, and digital data transmission networks.

The frequency of generating solutions to these problems depends on the application. In an airline hub location problem, the costs to change the hub airport to be another airport are high. Therefore, the airport network is designed in long term planning due to high installation cost.
In a telecommunication hub and computer network problem, we have the other extreme.

The system may require upgrades in the medium term due to rapid changing of technology and hardware lifecycle. In addition, the network is frequently redesigned because of increasing capacity or number of client computers in the system. In a humanitarian crisis, the most important goal is to provide humanitarian aid to the victims of natural disaster as soon as possible. Therefore, in uncertain situations in natural disasters such as earthquakes, tsunami, and typhoons, it is necessary to update frequently to evacuate victims. Moreover, redesigning hub network is also of concern after a natural disaster to deliver food, medical care, energy and water supplies.

1.1 Real world Data Sets

Real world data sets for hub location consist of different data sets known as the CAB, AP, and Turkish data sets. The Civil Aeronautics Board data set (CAB) is based on airline passenger traffic in the United States in 1970, with n = 10, 20, and 25 cities. The CAB data set was firstly introduced by O'Kelly (1987). The Australian Postal data set (AP) is derived from postal delivery network of Australian Postal. Each node represents a postal district and a hub represents a mail sorting center. Ernst (1996) first introduced the AP data set. The data set contains up to 200 nodes. The Turkish network data set was first introduced by Tan and Kara (2007). The data represents cargo delivery system between 81 cities in Turkey.

In practical applications, a single allocation network is appropriate to simplify customer service when a large amount of flows is involved such as in the postal delivery or trucking industries. Flows are collected to sorting centers and are then conveyed along the same allocation link from demand node to hub. The single allocation model is also suitable for environments with high set-up costs for arcs such as optical fibers in large-scale data communication networks, or for topology construction in peer to peer systems (Steffen, 2007).

Multiple allocation is used to minimize transportation costs in distribution networks when a variety of allocation arcs are worthwhile. In fact, many transportation networks utilize the multiple allocation model. For example, many airways provide airline routes through several airline hubs to allow passengers to choose their flights based on their preferences, price, or time constraints. If a flexible allocation network is desired, the multiple allocation model results in lower total transportation costs than does the single allocation model.

1.2 Capacitated and Uncapacitated Hub Locations

Hub location problems are categorized as either capacitated and uncapacitated. The capacitated hub problem involves with capacity constraint on the amount of flow through hub. On the other hand, there are no capacity restrictions for the uncapacitated hub problem. The capacity constraint is not be considered on the hub. Both capacitated and uncapacitated hub location problems may involve with installation cost.

While the hub location problem that has received the most attention from researchers so far is the single allocation *p*-hub median problem (Campbell & Ernst, 2002), this dissertation will focus on metaheuristics for solving uncapacitated *p*-hub median problems with both single and multiple allocations. The following sections further describe the related work and orient the reader to remaining chapters in the dissertation.

2. Literature Review and Background

2.1 Definitions and Introduction to the Hub Location Model

A network consists of a set of points and a set of lines that connect pairs of points, and *nodes* are the points in the network. *An arc* is a link between a pair of nodes, and *a path* is a sequence of arcs connecting node *i* to node *j* that makes flow from node *i* to node *j* along the path feasible.

A fully connected network is a network in which each of the nodes is connected to each other as shown in Figure 1. The main disadvantage of this type of network from the point of view of transporting flows between origin-destination pairs is that the number of arcs increases quadratically with the number of nodes, as the number of arcs is $N (N\text{-}1)$ for a network with N nodes.

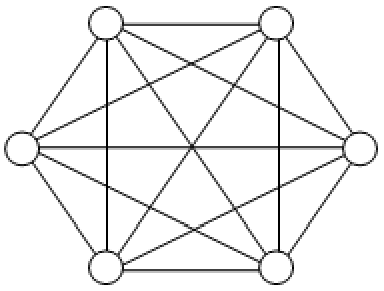


Figure 1.1 A Fully Connected Network

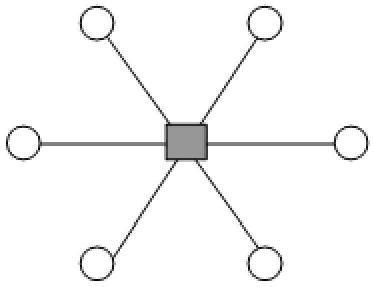
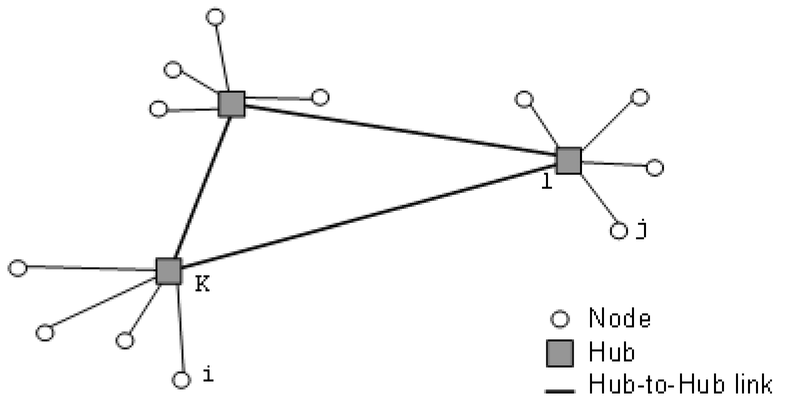


Figure 1.2 Hub and Spoke Network

Hubs are facilities that act as transshipment points in distribution systems to concentrate flows from different origins and then move them to the same destination. *A hub and spoke network* consists of one hub as the transshipment point and a set of spokes connecting to the non-hub nodes (see Figure 1.2). In hub-and-spoke networks, one node is assigned as the hub that connects all of the other nodes, and the number of paths is $2(N\text{-}1)$. The main advantages of hub-

and-spoke networks is that the transportation costs are significantly less when using this type of network than when using a fully connected network because a smaller number of routes is required. (O'Kelly, 1986a).



Source: Bridget (2008)

Figure 1.3 Typical Hub Location Network

A hub-and-spoke network can have multiple hubs. In this case, a *hub arc* or *hub link* is a link for transferring flow between hubs. Flows along hub links have reduced transportation costs because of increased consolidation. The reduced transportation costs are accounted for by a *discount factor* which sets the transportation costs along the hub links to be a percentage of the costs if those arcs were not hub links. In a typical hub location network, each origin-destination path consists of three components: collection from an origin to the first hub, possibly transfer between hubs along the hub links, and distribution from the last hub to the final destination as shown in Figure 1.3. Note that hubs may also serve as origins or destinations.

Research aspects of the hub location problem

The phrase “hub location” is used in many ways throughout the literature. Here, we review the two major types of hub location problems that motivate this dissertation: single and *p*-hub median problems. Researchers have approached hub location problems with two major techniques: exact methods (mostly based in mathematical programming), and approximation methods (generally using heuristic or metaheuristic methods). A heuristic is a set of rules in an algorithm that is used to find a good solution that limits the search for solutions. In more complex problems, metaheuristics are largely focused on solving hard optimization problems. Metaheuristics are approximate algorithms that guide the search process, but are not guaranteed to find an optimal solution in a bounded time. Many hub location problems are complex and consume computational time. Providing increasingly more robust methods, metaheuristics are mathematical optimization methods that have provided powerful solutions to the difficulties associated with real-world problems. Next, we review 4 models and selected heuristics in the literature related to this dissertation.

2.2 The single hub location model

The objective function of the single hub location model is to minimize the total cost to connect all nodes through a single hub while satisfying all the flows required between origin-destination pairs. Total flow costs are minimized when a single hub location model is applied (O’Kelly, 1986a).

Problem 1: Single hub location model

Inputs

C_{ij} is the unit flow cost on arc *(i, j)*

O_i is the total flow out of node *i*

D_i is the total flow into node *i*

N is the number of nodes

$i, j \in \{1, \ldots, N\}$

Decision variables

X_j = 1 if *j* is a hub

= 0 otherwise

Y_{ij} = 1 if node *i* is connected to a hub located at node *j*

= 0 if not

Min $\sum_i \sum_j C_{ij} Y_{ij} (O_i + D_i)$

s.t. $\sum_j X_j = 1$ (1)

$Y_{ij} - X_j \le 0 \qquad \forall i, j$ (2)

$X_j \in \{0,1\} \qquad \forall j$ (3)

$Y_{ij} \in \{0,1\} \qquad \forall i, j$ (4)

Constraint (1) ensures that we locate one hub. Constraint set (2) states that node *i* will be connected to hub *j* only when *j* is assigned to be a hub. Constraint sets (3) and (4) are standard integer restrictions.

2.3 The *p*-hub median model

The hub location problems that have received the most attention from researchers so far are *p*-hub median problems. There are two types of models for hub networks, which are categorized by the way the nodes are allocated to hubs. In a single allocation network model, every node is routed through only one hub as show in Figure 1.4. In a multiple allocation network model, each node can receive and send flows through more than one hub as seen in Figure 1.5.

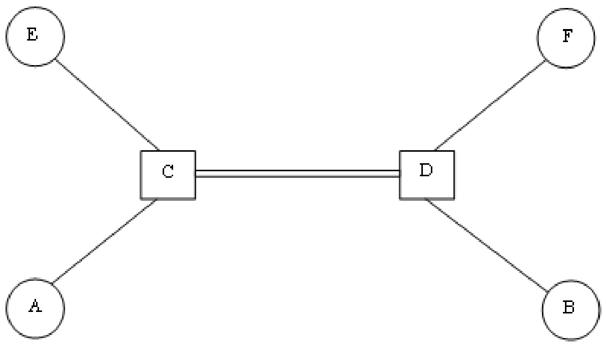


Figure 1.4 Single Allocation *p*-hub Median Network

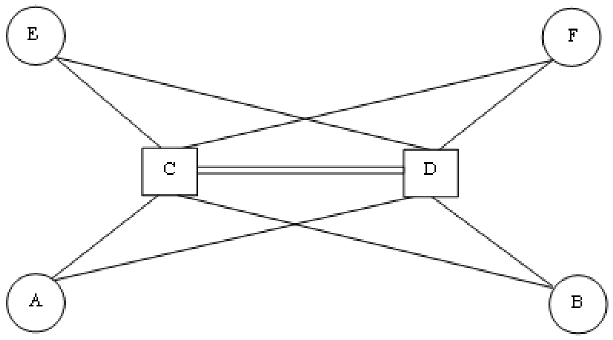


Figure 1.5 Multiple Allocation *p*-Hub Median Network

When the discount factor, which reduces the unit costs on hub arcs, is low, hub-to-hub transportation increases, and hubs are spread out further (O'Kelly, 1986a). The *p*–hub median model locates exactly *p* hubs in a network and allocates non-hub nodes to hubs to minimize the total travel cost. The *p*–hub median model is useful for real world situations such as transportation network, airline network, facility, and distribution center locations.

2.3.1 Uncapacitated single allocation *p*-hub median formulation

Skorin-Kapov (1996) proposed a mixed integer formulation to solve the uncapacitated single allocation *p*-hub median problem (known as USApHMP).

Problem 2: Single allocation p-hub median model by Skorin-Kapov (1996)

Additional notation

W_{ij} is the flow from node *i* to node *j*

α is the discount factor

p is the number of hubs

Decision Variables

X_{ijkm} is the fraction of flow from node *i* to node *j* that is routed via hubs at locations *k* and *m* (see Figure 1.3)

X_{ik} is 1 if origin *i* is allocated to hub *k*

0 otherwise

X_{kk} is 1 if *k* is a hub

0 otherwise

Min $\sum_i \sum_j \sum_k \sum_m W_{ij} X_{ijkm} (C_{ik} + C_{mj} + \alpha C_{km})$

s.t. $\sum_k X_{ik} = 1 \qquad \forall i \qquad (5)$

$\sum_k X_{kk} = p \qquad (6)$

$X_{ik} - X_{kk} \le 0 \qquad \forall i, k \qquad (7)$

$\sum_m X_{ijkm} = X_{ik} \qquad \forall i, j, k \qquad (8)$

$\sum_k X_{ijkm} = X_{jm} \qquad \forall i, j, m \qquad (9)$

$X_{ijkm} \ge 0 \qquad \forall i, j, k, m \qquad (10)$

$X_{ik} \in \{0,1\} \qquad \forall i, k \qquad (11)$

$X_{kk} \in \{0,1\} \qquad \forall k \qquad (12)$

The objective consists of three components: the cost of collecting from origin *i* to hub *k*, the cost of distributing from the hub *k* to destination *j*, and the cost of transferring between two hubs. Constraint set (5) requires a node to be allocated to one hub while constraint (6) specifies the number of hubs to be opened. Constraint set (7) ensures that *k* is a hub before a node can be allocated to the hub. Constraint sets (8) and (9) ensure that the flow from the origin to the destination will not be routed via hubs *k* and *m* unless origin *i* is assigned to hub *k* and destination *j* is assigned to hub *m*. Finally, constraint sets (10), (11) and (12) define the sign and integer restrictions.

Ernst and Krishnamoorthy (1996) proposed an integer programming formulation to solve USApHMP which requires fewer variables and constraints. The authors use different discount factors for flows from nodes to hubs and hubs to nodes in the objective function.

Problem 3: Single allocation *p*-hub median model by Ernst (1996)

The new notation is shown below.

χ is the discount factor for collection (node to hub)

δ is the discount factor for distribution (hub to node)

Decision variables

X_{ik} 1 if node *i* is allocated to a hub located at node *k*

0 otherwise

Y^i_{km} is the total amount of flow emanating from node *i* that is routed between hubs *k* and *m*.

Minimize $\sum\sum C_{ik} X_{ik} (\chi O_i + \delta D_i) + \sum_i \sum_k \sum_l \alpha C_{km} Y^i_{km}$

s.t.

$$\sum_k X_{ik} = 1 \qquad \forall i \qquad (5)$$

$$\sum_k X_{kk} = p \qquad (6)$$

$$X_{ik} - X_{kk} \leq 0 \qquad \forall i,k \qquad (7)$$

$$\sum_m Y^i_{km} - \sum_m Y^i_{mk} = O_i X_{ik} - \sum_j W_{ij} X_{jk} \qquad \forall i,k \qquad (13)$$

$$Y^i_{km} \geq 0 \qquad \forall i,k,m \qquad (14)$$

$$X_{ik} \in \{0,1\} \qquad \forall i,k \qquad (15)$$

Constraint sets (5) to (7) are the same as in Problem 2. The only change is the flow balance constraint set (13) which states that the flow balances for flow commodity *i* at node *k* where demand and supply at the node is determined by the arc X_{ik}.

2.3.2 Heuristics for USApHMP

O'Kelly (1987) proposed two heuristics to find the number of hubs to be selected for USApHMP based on the Skorin-Kapov model (Problem 2) . In HEUR1, each node is assigned to the nearest hub. In HEUR2, O'Kelly evaluated all possible ways to allocate non-hub nodes to their nearest and second nearest hubs. HEUR2 evaluates all of the 2^{N-P} possible ways of assigning the non-hub nodes to the nearest and second nearest hubs. O'Kelly found that the runing time of HEUR2 increases corresponding to the number of nodes and found that HEUR1 performs well in term of cost. Further, O'Kelly found that when the discount increases (α small), the difference between the objectives of both heuristics get smaller.

Klincewicz (1991) developed an exchange heuristic for USApHMP based on local improvement by considering both single and double exchange procedures. This exchange heuristic exchanges hub locations based on a measure of local improvement. This is compared with the heuristics proposed by O'Kelly (1987). The exchange heuristic is superior in terms of computational time and a small between best solution found by the heuristic and the optimal solution.

Klincewicz (1992) presented a tabu search and a greedy randomized search procedure (GRASP) heuristic; in both of these heuristics, the demand nodes are allocated to their nearest hubs. The tabu search performed better in CPU time while GRASP reached optimal solutions more frequently. Klincewicz (1991, 1992) used the CAB (Civil Aeronautics Board) data set, based on the airline passenger interactions evaluated by the Civil Aeronautics Board, and a larger problem size with 52 demand points and up to 10 hubs to measure the performance of the heuristics.

Skorin-Kapov (1994) developed a tabu search heuristic, TABUHUB, and compared the heuristic with HEUR1, HEUR2 and the tabu search developed by Klincewicz (1992). The results were better, but the CPU times were longer. Ernst and Krishnamoorthy (1996) developed a simulated annealing heuristic and compared it with the tabu search presented by Skorin-Kapov (1994), which showed that their simulated annealing heuristic is comparable for both computer time and solution quality.

Chen (2007) developed a hybrid heuristic based on the simulated annealing method and the tabu list for a large sized problem with 100 and 200 nodes in the AP (Australian Post) data set by comparing it with the genetic algorithms and the simulated annealing. This heuristic outperformed the genetic algorithms and the simulated annealing in CPU time and solution quality.

Perez et al. (2004) proposed a path-relinking algorithm for the USApHMP. The representation of the path-relinking consists of two parts in one array. The path-relinking was run on the AP data set up to 100 nodes and compared with TABUHUB. The authors found that the path-relinking performs better than TABUHUB both computational times and quality of solutions.

Kratica et al. (2007) developed GAs in two representations, GAHUB1 and GAHUB2, to solve the USApHMP based on the CAB and the AP data set. In the selection procedure, fine grained tournament was used for both representations. Caching was used in the evaluation function to improve running time. The authors found that GAHUB2 reached all best known solutions.

2.3.2 Multiple allocation *p*-hub median problems

Recall that in the uncapacitated multiple allocation *p*-hub median problem (UMApHMP), the flow from a node may be routed through different hub nodes, depending on the destination node.

Problem 4: UMApHMP by Campbell (1992)

This model uses notation and decision variables previously introduced in Problem 2.

Min $\sum_i \sum_j \sum_k \sum_m W_{ij} X_{ijkm} (C_{ik} + C_{mj} + \alpha C_{km})$

s.t. $\sum_k X_{kk} = p$ (6)

$\sum_k \sum_m X_{ijkm} = 1 \quad \forall i, j$ (16)

$\sum_m X_{ijkm} \le X_{kk} \quad \forall i, j, k, m$ (17)

$\sum_k X_{ijkm} \le X_{mm} \quad \forall i, j, k, m$ (18)

$X_{ijkm} \ge 0 \quad \forall i, j, k, m$ (10)

$X_{kk} \in \{0,1\} \quad \forall k$ (12)

The objective is to minimize the transportation cost of the network which is the same as in problem 2. Constraint set (6) is the same as in the problem 2. Constraint set (16) ensures that the flow between every origin – destination (*i*,*j*) is routed through some hubs. Constraint sets (17) and (18) ensure that the flow from the origin to the destination is only routed by locations that are hubs. The sign restrictions are shown in constraint sets (10) and (12).

Problem 5: UMApHMP by Ernst (1998)

Ernst‘s 1998 formulation for the UMApHMP was introduced as an improved alternative to Campbell’s 1988 model and retains the idea of three different discount factors as used in Problem 2 (Ernst 1996). The model requires fewer variables than Campbell (1998).

Decision variables

Z_{ik} the flow from node *i* to hub *k*

X_{lj}^{i} the flow from node *i* through hub *l* to node *j*

H_k 1 if node *k* is a hub

0 otherwise

Y_{kl}^{i} is the total amount of flow from node *i* that is routed between hubs *k* and *l*.

Minimize $\sum_i \left[\sum_k \chi C_{ik} Z_{ik} + \sum_k \sum_l \alpha C_{kl} Y_{kl}^{i} + \sum_l \sum_j \delta C_{lj} X_{lj}^{i} \right]$

s.t. $\sum_k H_k = p$ (19)

$\sum_k Z_{ik} = O_i \qquad \forall i$ (20)

$$\sum_{l} X_{lj}^{i} = W_{ij} \qquad \forall i, j \qquad (21)$$

$$\sum_{l} Y_{kl}^{i} - \sum_{j} X_{kj}^{i} - \sum_{l} Y_{kl}^{i} - Z_{ik} = 0 \qquad \forall i, k \qquad (22)$$

$$Z_{ik} \le O_i H_k \qquad \forall i, k \qquad (23)$$

$$\sum_{i} X_{lj}^{i} \le D_j H_l \qquad \forall l, j \qquad (24)$$

$$H_k \in \{0,1\} \qquad \forall i, k \qquad (25)$$

$$Y_{kl}^{i}, X_{lj}^{i}, Z_{ik} \ge 0 \qquad \forall i, j, k, l \qquad (26)$$

Constraint (19) represents the number of hubs to be assigned. Constraints set (20) – (22) represent the flow balance for node *i*. Constraint set (20) ensures that total flow emanating from node *i* routed by every hub *k* is equal to total amount of flow originating at node *i*. Constraint set (21) ensures that total flow from node *i* flowing from every hub *l* to node *j* is equal to flow between node *i* and *j*. Constraint set (22) states that the flow balances for node *i* to node *j* must be routed via a feasible path. Constraint set (23) states that flow from node *i* flows through hub *k* only if node *i* is allocated to hub *k*. Constraint set (24) states that total flow from node *i* flowing from every hub *l* to node *j* only if node *j* is allocated to hub *l*. The sign restrictions are shown in constraint sets (25) and (26).

2.3.3 Heuristics for UMApHMP

Some research using metaheuristics to solve the UMApHMP has been performed. Stanimirovic (2008) proposed a genetic algorithm with binary representation to solve the UMApHMP. The methods are evaluated using the CAB and the AP data with up to 200 nodes.

The author also implemented caching to reduce the computational time in the evaluation function. For example, for a problem with 50 nodes and 4 hubs, the optimal solution has been found within 0.885 sec and for a problem with 200 nodes and 3 hubs, the optimal solution has been found within 174.9 sec.

Kang (2008) implemented the ant colony optimization model for solving the UMApHMP on the CAB and the AP data set. The author proposed and evaluated different multiple allocation methods: nearest hub and all pair shortest path. From the result, the combination policy, combining nearest hub and all pair shortest path, is the most effective allocation method to solve larger instances in reasonable computational time. For example, for a problem with 50 nodes and 4 hubs, nearly a optimal solution has been found within 143.48 sec and for a problem with 200 nodes and 3 hubs, a nearly optimal solution has been found within 3636.64 sec.

Marija (2010) proposed an evolutionary algorithm with permutation representation to solve the UMApHMP on the AP data set. From the results, the optimal solutions are almost found for every problem instances and some new best solutions for problem sizes of 200 nodes are found. For example, for a problem with 50 nodes and 4 hubs, the optimal solution has been found within 0.551 sec and for a problem with 200 nodes and 3 hubs, the optimal solution has been found within 73.947 sec.

In this dissertation, Multiple TS is proposed to solve the UMApHMP on the AP data set. We identify multiple nodes using the Convex Hull and Single Node Exchange. We use Single Reallocation Exchange procedures to find optimal allocations. For example, for a problem with 50 nodes and 4 hubs, the optimal solution has been found within 526.25 sec and for a problem with 200 nodes and 3 hubs, the optimal solution has been found within 5322.46 sec.

2.4 Work for other related network problems

Campbell and Ernst (2005a) presented four models to locate hub arcs as links between hubs. Their evaluation used the Civil Aeronautics Board (CAB) data, based on air passenger traffic in the United States, with n = 10, 20, and 25 cities. Then, the costs from four models were compared. The cost per unit flow for collection and distribution on the access arcs was set to 1.0. The cost per unit flow for transfer on a bridge arc, an arc connected between two hubs without a reduced unit flow cost, was 1.0. For transfer on hub arcs, five different levels of the discount factor were considered: α = 0.2, 0.4, 0.6, 0.8, and 1.0. The cost results suggest that strong economies (smaller α) and fully connected hub arcs with fewer hubs and more hub arcs are better than unconnected hub arcs with more hubs and fewer hub arcs. With weak economies of scale (large α), then many hubs with less well-connected hub arcs are preferable.

Hatice and Sibel (2008) proposed an integer programming model for incomplete hub networks and proposed a tabu-based heuristic algorithm to solve a realistically sized problem in a Turkish network. The incomplete hub network only connects necessary terminals in the hub network to decrease the investment cost. In contrast, in a complete hub network every hub pair is interconnected with a hub link. The tabu-based heuristic constructs feasible solutions for the hub-covering problem with tight time bounds. Three different allocation strategies are used in constructing feasible solutions. The heuristic algorithm was tested on the CAB data set for 10, 15 and 20 nodes and the Turkish network was tested with 81 nodes. The performance of the heuristic was compared with CPLEX on the CAB data set. The heuristic obtained good solutions with less CPU time than CPLEX.

3. Research Problem

The dissertation focuses on hybrid metaheuristics for large-sized problems (100 – 200 or more nodes) for various *p*-hub median models. Although metaheuristics have been widely examined for hub locations, hybrid metaheuristics for large-sized problems has not been analyzed in depth. In general, as the size of the problem increases, the quality of the metaheuristics solution decreases.

Based on the previous literatures, research has shown that pure metaheuristics do not run as fast as hybrid heuristics. This research develops metaheuristics for several large p- hub median problems that generate good solutions.

4. Research Overview

The objective of this research is to design metaheuristics to generate high quality solution. To outline this work, there will be three chapters in this dissertation, consisting of the following topics.

1. Comparison of different types of metaheuristics for the uncapacitated single allocation *p*-hub median model.
2. Comparison of different types of hybrid metaheuristics for the uncapacitated single allocation *p*-hub median model.
3. Development of metaheuristics for the uncapacitated multiple allocation *p*-hub median model.

CHAPTER TWO

METAHEURISTICS FOR THE USApHMP

1. Introduction

The uncapacitated single allocation *p*-hub median problem (USApHMP) belongs to the class of NP-hard problems (O'Kelly, 1987). In the first section of this chapter, A Mathematical Programming Language (AMPL) and Optimization Programming Language (OPL) are used to illustrate the limitations of current formulations for USApHMP on small to large instances.

To investigate the difficulty of solving USApHMP on large data sets, exact optimization tools are applied. Ernst (1996) suggests that when the problem includes more than 100 nodes, his formulation (Problem 3 in Chapter 1) is too large to be solved optimally; however, metaheuristic methods may produce reliable results in reasonable computational times. Ernst (1996) introduced the Australian Postal (AP) data set as a real world hub location data set. The entire set contains 200 nodes, but subsets with 10, 20, 40, 50, and 100 nodes can be generated. In order to provide justification for the application of metaheuristics, we evaluate the computational effort required by CPLEX 11.2 with AMPL and OPL5.2 on an Intel Core Duo 1.66 GHz with 1 GB RAM to solve the Ernst 1996 formulation in Table 2.1.

For USApHMP, computational time increased enormously as the number of nodes increased. When the number of nodes is more than 120, AMPL ran out of memory. Two strategies exist for resolving this problem: developing more effective math programming formulations or heuristic / metaheuristic approaches. We focus on the metaheuristic approach in this dissertation.

Table 2.1 Results from Ernst's formulation for USApHMP (Problem 3)

Number of Nodes	*P*	Obj	AMPL time(s)	OPL time(s)
10	2	167493	0.0625	0.053
	3	136008	0.1875	0.28
	4	112396	0.23438	0.51
	5	91105.4	0.17188	0.5
20	2	172817	0.40625	2.17
	3	151533	1.0625	4.09
	4	135625	1.73438	3.51
	5	123120	2.15625	6.29
50	2	178484	39.0156	621
	3	158570	42.9375	342
	4	143378	49.6875	960
60	2	179920	64.42	-
	5	132850	297.484	-
	10	102940	475.562	-
80	2	180182	172.016	-
	4	145810	977.984	-
100	2	180224	1217	-
	4	145897	3693.22	-

2. Metaheuristics

Optimization software, such as AMPL, can solve USApHMP for instances based on the AP data set up to 100 nodes. Therefore, we anticipate that metaheuristics can be applied to solve problems in a reasonable time, especially when the size of problems is large. Genetic algorithms, tabu search, and simulated annealing are the specific metaheuristics applied in the dissertation. We first describe the elements that are common to all the metaheuristics examined in this dissertation. In this chapter we focus on USApHMP while Chapter 4 is concerned with UMApHMP.

2.1 Representation

The solution representation is the data structure that the metahueristic operates upon. It can be a direct representation or some sort of indirect representation. A variety of representations are possible for USApHMP and will be described with the help of the following seven node, three hub solution. In this example, nodes 3, 4 and 7 are hubs. Nodes 1, 2, and 3 are allocated to node 3; nodes 4 and 6 are allocated to node 4; and nodes 5 and 7 are allocated to 7 as shown in Figure 2.1.

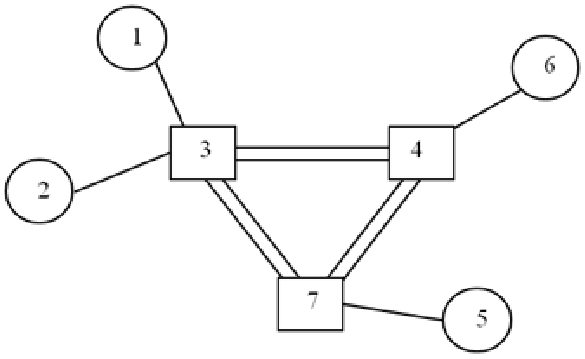


Figure 2.1: Solution Represented

For example, a permutation representation of 4334477 denotes that nodes 3, 4, and 7 are hubs with the non-hub nodes 1, 2, 5, and 6 being connected to hubs 4, 3, 4, and 7, respectively. In another representation structure (Topcuoglu, 2005), two sets of arrays are designed separately: the first set expresses the location as a binary array (0011001): bits 3, 4 and 7 are one; the second set represents allocation (4334477). In a third alternative, two parts of the array are combined in one chromosome (Vladimir, 2009). The first bit identifies the hubs as a binary array (0011001), while the second bit specifies which hub is used (first, second, etc) for each node; the first hub is hub 3, and the next hubs are hub 4 and 7, respectively. The combined representation is shown as 02|01|11|12|02|03|13|. In a fourth alternative, a two-part combination was proposed in a different structure. The first bit represents the location as a binary, and the second bit expresses hub priority or distance as proposed in GAHUB2 (Josef, 2007). For example, the first section denotes opened hub (0,0,1,1,0,0,1) and the second section indicates assignment (0,0,0,0,0,0,0,0). The representation is 00|00|11|10|00|00|10|.

In considering which representation to use, important factors of the representations should be considered, such as the simplicity of modifying the solution, the steps needed for arrangement, and the reasonability of adjusting the hub allocation. To study the effects of different representations on algorithm efficiency, two well-known representations are investigated.

2.1.1. Permutation representation

The permutation representation consists of two parts in arrays: the hub part and the allocation part. The hubs are shown as a partial permutation of the *n* nodes having length *p*. The allocation part has one element for each node in the network indicating the hub to which the node is allocated. For the example solution, the representation is 347|3334747. The first part of the array indicates that 3, 4, and 7 are the set of hubs, with nodes 1, 2, and 3 served by the hub at node 3, etc.

2.1.2. Binary representation

The binary representation we consider has two parts, each of which has an element for each node in the network. The first part, called the location part, determines the hub nodes in the network, and the second part, called the allocation part, represents a set of hubs to which the non-hub nodes are allocated. We call a set of nodes allocated to the same hub a “cluster”. In the location part, the value 1 indicates a node is a hub node, while the value 0 indicates the node is not a hub node. The allocation part represents the connections of the nodes to the hubs in the cluster. In the allocation part, the representation structure is identified as a cluster. The same cluster is defined as the set of nodes allocated to the same hub.

In the example solution, the location part is 0011001. The allocation part is 0001212. The first cluster, represented by value 0, consists of nodes 1, 2, and 3, which are assigned to node 3. In an alternative expression, nodes 1, 2, and 3 of the allocation array represent cluster 0 (served by the hub at node 3). The second cluster, represented by value 1, consists of nodes 4 and 6, assigned to hub 2, which is node 4. Finally, the last cluster, represented by value 2, indicates node 5 and 7 are assigned to hub 3, which is node 7. Consequently, the solution is represented by the array 0011001|0001212.

2.2 Generating initial feasible solutions

To prevent infeasibility of the initial solutions generated, random keys are applied to encode the solutions. The random key is selected from a uniform distribution in the interval [0,1). To decode the solutions, the random keys are sorted into ascending order and the first p nodes are selected as the hubs.

For example: $n = 7$, $p = 4$

Table 2.2 Decoding random keys

Order before sorting	1	2	3	4	5	6	7
Random key	0.25	0.52	0.32	0.08	0.94	0.16	0.03
Sorted random key	0.03	0.08	0.16	0.25	0.32	0.52	0.94
Order after sorting	7	4	6	1	3	2	5

Decode as hub set: 7, 4, 6, 1

In both representations we use random keys to encode hub sets to avoid generating and repairing infeasible initial solutions. The Mersenne Twister algorithm is utilized to generate uniform random numbers.

2.3 Reallocation Procedure

Once a hub set is known, it is possible to consider allocating each non-hub node to each hub. We implement an iterative heuristic method to find good allocations of non-hub nodes to hubs given an initial solution, called the Reallocation Procedure. In the Reallocation Procedure, each non-hub node changes its allocation to each of the different hubs and the resulting objective function is computed. Each of the (n-p)(p-1) neighbors or possible new solutions are created for each iteration. The best solution from the iteration is chosen as the current solution for the next iteration. This process is continued until no further improved solution can be generated.

Let P be the set of hubs and P' be the set of non-hub nodes. Let $h(i)$ be the hub allocated to node i and the n-tuple $\boldsymbol{h}$ be the allocation for all n nodes. A new allocation for a single hub is indicated by $h(i)'$ and a new allocation for all n nodes is indicated by $\mathbf{h}'$. The Reallocation Procedure is described as follows:

```
$z$ = objective for the solution $P$ and $\boldsymbol{h}$
done = FALSE
while done = FALSE
    done = TRUE
    For $i \in P'$,
        For $j \in P \setminus \{h(i)\}$
```

```
                    $h(i)'=j$
                    $z'$ = objective for the solution $P$ and $\mathbf{h}'$
                    if $z'<z$, then $\mathbf{h} = \mathbf{h}'$, $z = z'$, and done = FALSE
          end for
     end for
end while
report **h**
```

Numerical example:

Assume *P*= {3, 4, 7} and **h=**{7, 4, 3, 4, 7, 4, 7, 7,7, 7}. There are (*n*-*p*)(*p*-1) = 14 neighbors. Two iterations of the Reallocation Procedure are shown below, with the new hub allocations for the non-hub nodes underlined the hub nodes bolded.

Iteration 1

											Resultant Objective (current or better)
H	7	4	**3**	**4**	7	4	**7**	7	7	7	163341
	<u>3</u>	4	**3**	**4**	7	4	**7**	7	7	7	
	<u>4</u>	4	**3**	**4**	7	4	**7**	7	7	7	
	7	<u>7</u>	**3**	**4**	7	4	**7**	7	7	7	
	7	<u>3</u>	**3**	**4**	7	4	**7**	7	7	7	
	7	4	**3**	**4**	<u>3</u>	4	**7**	7	7	7	147205*
	7	4	**3**	**4**	<u>4</u>	4	**7**	7	7	7	
	7	4	**3**	**4**	7	<u>7</u>	**7**	7	7	7	
	7	4	**3**	**4**	7	<u>3</u>	**7**	7	7	7	
	7	4	**3**	**4**	7	4	**7**	<u>3</u>	7	7	
	7	4	**3**	**4**	7	4	**7**	<u>4</u>	7	7	
	7	4	**3**	**4**	7	4	**7**	7	<u>3</u>	7	
	7	4	**3**	**4**	7	4	**7**	7	<u>4</u>	7	
	7	4	**3**	**4**	7	4	**7**	7	7	<u>3</u>	
	7	4	**3**	**4**	7	4	**7**	7	7	<u>4</u>	

Iteration 1 results in a new hub allocation being defined as **h=**{7, 4, 3, 4, 7, 4, 7, 7 ,7 ,7} with a new objective value *z*=147205.

Iteration 2

											Resultant Objective (current or better)
H	7	4	**3**	**4**	3	4	**7**	7	7	7	147205
	$\underline{3}$	4	**3**	**4**	3	4	**7**	7	7	7	136671*
	$\underline{4}$	4	**3**	**4**	3	4	**7**	7	7	7	
	7	$\underline{7}$	**3**	**4**	3	4	**7**	7	7	7	
	7	$\underline{3}$	**3**	**4**	3	4	**7**	7	7	7	
	7	4	**3**	**4**	$\underline{7}$	4	**7**	7	7	7	
	7	4	**3**	**4**	$\underline{4}$	4	**7**	7	7	7	
	7	4	**3**	**4**	3	$\underline{7}$	**7**	7	7	7	
	7	4	**3**	**4**	3	$\underline{3}$	**7**	7	7	7	
	7	4	**3**	**4**	3	4	**7**	$\underline{3}$	7	7	
	7	4	**3**	**4**	3	4	**7**	$\underline{4}$	7	7	
	7	4	**3**	**4**	3	4	**7**	7	$\underline{3}$	7	
	7	4	**3**	**4**	3	4	**7**	7	$\underline{4}$	7	
	7	4	**3**	**4**	3	4	**7**	7	7	$\underline{3}$	
	7	4	**3**	**4**	3	4	**7**	7	7	$\underline{4}$	

Iteration 2 results in a new hub allocation being defined as **h=**{3, 4, 3, 4, 7, 4, 7, 7 ,7 ,7} with a new objective value *z*=136671. This procedure is repeated until no improved hub allocation is found.

3. Genetic algorithms

Genetic algorithms (GAs), which were first proposed by Holland (1975), are a powerful type of algorithm for optimization that imitates natural selection and crossover to produce highly fit offspring. Genetic algorithms are based on genetic processes to combine the best properties of the parent entities.

GAs perform by first generating initial populations depending on their representation structures, and then selecting the chromosomes of the next generation based on their fitness; namely, high fitness chromosomes have a higher chance of surviving and reproducing. Crossover occurs by recombining a portion of a pair of parents to produce offspring, while mutation is applied to individual chromosomes after crossover to avoid entrapment in local optima. Mutation is performed by randomly altering a portion of the chromosome to generate new chromosomes with a small probability. Figure 2.2 illustrates the GA procedure in this research with details provided in the following sections.

The GA for USApHMP begins by randomly creating the initial population by randomly generating a hub set for each chromosome. Once the hub sets have been defined, every node in the network is allocated to its nearest hub. Then, the Reallocation Procedure is applied to reassign each non-hub node to a single hub.

```
GA Procedure
  Generate initial population;
  Allocate each node to the nearest hub in each chromosome;
  Evaluate the fitness of each chromosome;
  Perform Reallocation Procedure for each chromosome
  While (stopping criterion is not achieved) do
        Elite Reproduction 5% of Population;
        Selection;
            40% from Elite parents;
            60% of randomly selected parents;
        Crossover 95% of Population;
            Cut and splice crossover for Binary representation;
            Single Point Crossover for Permutation representation;
        Mutation 20% of Population
            Swap (hub, non-hub node)
        Choose the BestHubSet;
        Perform Reallocation Procedure for each chromosome ;
```

Figure 2.2 Genetic algorithm procedure

3.1 Selection

The population for the next generation is created by selecting parents to crossover, followed by crossover and mutation operations. We utilize elite reproduction, in which the top 5% parent solutions are kept in the next generation to ensure the current best solution will not be lost. The next step involves choosing a high fitness top 5% parent with a probability 0.4 to mate with another parent that is randomly selected with a probability 0.6 to generate high fitness offspring. .

3.2 Crossover

Due to the different structures of each chromosome for each representation, two types of crossovers are applied in each representation: the cut and splice crossover for the binary representation and the single point crossover for the permutation representation. We only perform crossover on the hub section and use a reallocation scheme to fill the allocation section.

3.2.1 Cut and splice crossover for binary GA

In the USApHMP problem, the number of hubs is fixed. After crossover is implemented, the number of hubs must remain the same. The cut and splice crossover is used for the binary representation to address this concern.

We arbitrarily designate one chromosome as Parent 1 and the other as Parent 2. Cut and splice crossover begins by selecting the cut point of parent 1 deterministically at the position c_1 just after half of the number of hubs are detected. Then, the remaining hubs of parent 2 at the position c_1+1 are used to produce child 1 until p hubs are filled in child 1 as shown in Figure 2.3. Therefore, extra hubs are prevented in cut and splice crossover. However, if the number of hubs from position c_1+1 to n in parent 2 is less than p in the network, the remaining hubs are selected randomly from the non-hub nodes. Then, select the cut point of parent 2 at the position c_2 and repeat the same procedure as parent 1.

The following example illustrates the Cut and splice crossover.

Example $p = 4$

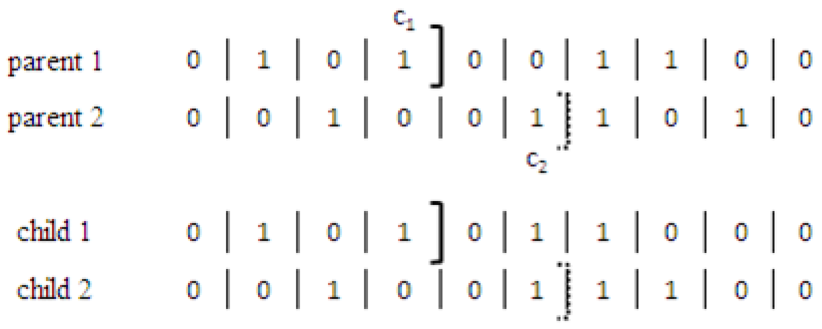


Figure 2.3 Cut and splice crossover for a binary GA

In this example, p=4 so c_1 is set to be position 4. Child 1 therefore keeps the first 4 values from parent 1. The rest of Child 1 is taken from Parent 2, but only the first three values are taken from Parent 2 after c_1, as 4 hubs are determined by that point.

3.2.2 Single point crossover for the permutation representation GA

In the permutation representation, we implement a traditional single point crossover. A single point is randomly selected along the hub section. The offspring are generated by combining the left and the right parts of the parents as shown in Figure 2.4. If the number of hubs of the offspring is not equal to p, it is discarded and another attempt is made.

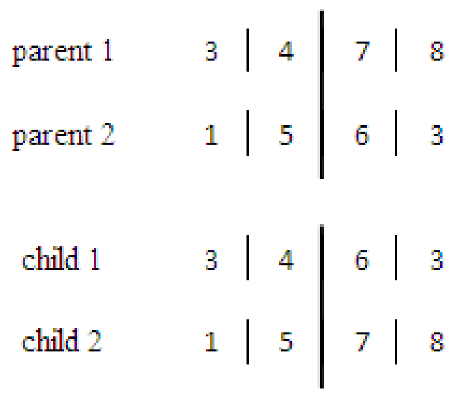


Figure 2.4 Single point crossover for the permutation representation

3.3 Mutation

After crossover, a mutation is applied to explore the search space more fully. Chromosomes are randomly selected with the probability 20% of the population and mutated by swapping one randomly selected non-hub node with a randomly selected hub nodes.

4. Simulated Annealing

Simulated annealing (SA) is a probabilistic hill climbing algorithm and was developed by Metropolis in 1953 to simulate the annealing process in metals. SA has been widely implemented for combinatorial optimization problems. SA operates by accepting moves to improved solutions while allowing non-improving moves with some probability to avoid local minima. In each move, the acceptance probability is examined when deciding whether or not to accept the non-improving solution. If the new objective value is better than the current objective, the new solution will automatically be accepted. However, if the objective value is worse than the current objective function value, some decision parameters, such as the current temperature, the cooling schedule, and the magnitude of the objective difference, are considered to allow moving. According to the Metropolis criterion, the new solution is accepted if a random number, *r*, generated from uniform distribution [0,1] is no larger than the acceptance probability. Figure 2.5 illustrates the SA procedure in this research with details provided in the following sections.

```
Procedure Simulated Annealing
  begin
        Initial solution creation $s_0$;
        Set initial temperature $T_0$ and cooling rate $\rho$;
        SingleHubExchange;
        SelectBestHubSet;
    While (stopping criterion is not achieved) do
                generate random number $r$ unif(0,1)
                If ($r > 0.40$),
                      then MoveSameCluster;
                Else  MoveDifferentCluster;
                Perform Reallocation Procedure;
                If  $f(s) < f(s_0)$,
                      then accept solution;
                Else generate random number $r$ unif(0,1);
                If  $r < e^{\frac{-(f(s)-f(s_0))}{T_i}}$,
                      then accept solution;
                Update temperature  $T_{i+1} = \rho T_i$
    End while
```

Figure 2.5 Simulated annealing procedure

4.1 SingleHubExchange

The initial solution is generated by choosing a hub set from the set of nodes where the highest amount of flow occurs. Next, SingleHubExchange is performed by replacing one of the current hubs with a non-hub node; there are $(n\text{-}p)p$ possible hub sets. Then, non-hub nodes in each hub set are allocated to the closest hub. Next, evaluate $(n\text{-}p)p$ solutions and select the best objective value or the best hub set from SingleHubExchange to perform the reallocation. For example, $n = 5$, $p = 2$. The first two nodes having highest amount of flow are node 3 and 5. The number of solutions is 6. The possible hub sets are (1,5) , (2,5), (4,5), (3,1), (3,2),and (3,4).

4.2 Neighborhood definitions

The neighbors are defined based on the hub sets. Recall the concept of a *cluster*: a set of non-hubs nodes allocated to the same hub. To generate a new hub set in the neighborhood, transitions are defined in two ways:

(1) MoveSameCluster: Change the location of the hub to be a different node in the same cluster.

(2) MoveDifferentCluster: Replace one of the current hubs with a non-hub node from another cluster.

We determine which type of neighbor to generate based on a random number, selecting MoveSameCluster with 40% probability and MoveDifferentCluster with 60% probability.

4.3 SA: temperature and schedule

The acceptance probability to move into a non-improving solution is $e^{\frac{-\Delta E}{T_{i-1}}}$ where ΔE is the difference between current and new objective value. The temperature T_i in iteration i is reduced by a cooling schedule of $\rho = 0.97$, as shown in Figure 2.5 (Jeng, 2008). The initial temperature is 2000, based on initial empirical results.

Following Osman (1993), when the same solutions appear in *2np* iterations, reheating is used to avoid local optima. The reheated initial temperature is set to be $max(T_{r-1}, T_{incumbent})$, where $r = 1,\ldots,4$.

5. Tabu Search

Tabu search (TS) is a heuristic methodology that was originally proposed by Fred Glover in 1986. By using a memory structure, tabu search can forbid revisiting previously visited solutions, and it accepts non-improving solution to avoid local optima. The decision factors of tabu search are defined by neighborhood structure and memory structure. In tabu search implementation, recently visited solutions are recorded as a list in short-term memory to prevent cycling in the same local optimum. In long-term memory, the frequency of the hub set solutions obtained is also considered to generate a new initial solution and to differentiate it from the visited solutions. In USApHMP, TabuHub was developed to solve the problem for the CAB data set (Skorin, 1994). We implement a different tabu search methodology based on efficient processes consisting of single location exchange for hub and the assignment reallocation for allocation part. Figure 2.6 illustrates the TS procedure in this research with details provided in the following sections.

```
Procedure Tabu search
    begin
      Generate initial hub set
      SingleHubExchange;
      Shortest allocation;
      SelectBestHubSet;
      While (stopping criterion is not achieved) do
          Repeat
            Reallocation;
            SelectBestAllocation;
            UpdateTabuList;
            Update the current solution;
            end;
          until(stopping criteria);
      End While;
End
```

Figure 2.6 Tabu search algorithm

Neighborhood definitions

For neighborhood of TS, SingleHubExchange is performed and the best objective value or the best hub set is selected; this is the same as SA.

5.1 Tabu List Representation

The tabu list is based on the elements of movement from the current solution to its selected neighbor, which will be recorded in the tabu list to prevent cycling. Due to neighborhood structures consisting of location and allocation parts, the location and allocation lists are created separately (Skorin, 96). In the location part, when a hub is replaced with a non-hub node. The recently deleted hub will be recorded in the tabu list to forbid the same hub being selected during the tabu tenure length. For the allocation part, after the reallocation is performed, the best reallocation of non-hub node *n* with hub *h* is determined. The tabu list for the allocation part is composed of the pair *(h, n)*. To diversify the initial hub set, long-term memory is applied when the number of iterations is equal to the long-term memory length, after which the initial hub set is regenerated. Selecting the new initial hub set is based on the proportion of the amount of flow divided by the frequency of the nodes designated as hubs.

5.2 Stopping criteria

In tabu search, the algorithm will stop when no new incumbent has been identified for at least a fixed number of consecutive iterations.

5.3 Difference between TabuHub and Tabu search

In tabu search for USApHMP, the single location exchange and reallocation procedure as first introduced in TabuHub were applied to find the best hub set and the best allocation of the hub network. In contrast with TabuHub, we find the best reallocations after each hub set has been produced from single location exchange. For the reallocation process, we consider only the allocation part as input. First, the reallocation process is only used for the best hub set from the single location exchange procedure while the reallocation procedure of TabuHub is called after every hub set is generated, and second, tabu search only applies in the last stage after the best hub set is generated. The reallocation process will continue until no further improved solution is found.

Tables 2.3 and 2.4 illustrate the difference between TabuHub and this tabu search. In Table 2.3, the different set of hubs, (*n-p*)*p* neighbors, are generated by the single location exchange procedure and every non-hub node is aloocated to its nearest hub. The possible representations are shown in the single location exchange column. Then, the best hub set is chosen.

Numerical example: 10 nodes and 3 hubs. Assume the initial hub set is 3, 7, and 8.

Table 2.3 Set of neighborhoods generated from TABUHUB

Single Loc Exchange Hub set			Shortest Allocation Results	Possible Reallocation Neighborhoods *(n-p)(p-1)*
1	7	8	1 8 7 8 1 1 7 8 7 7	14
2	7	8	2 2 8 7 2 8 2 8 7 7	14
.	7	8	.	14
.	7	8	.	14
10	7	8	7 7 10 8 7 10 7 8 7 10	14
3	1	8	1 8 3 1 1 3 3 8 8 8	14
3	2	8	3 2 3 3 2 3 3 8 8 8	14
3	.	8	.	14
3	.	8	.	14
3	10	8	3 3 3 8 8 10 8 8 8 10	14
3	7	1	1 3 3 7 1 3 7 7 7 7	14
3	7	2	2 2 3 7 3 3 7 7 7 7	14
3	7	4	7 4 3 4 7 4 7 7 7 7	14
3	7	.	.	14
3	7	.	.	14
3	7	10	7 7 3 3 7 10 7 7 7 10	14

Next, reallocation for *each* potential hubset is performed. In each iteration, the number of neighbors generated from TabuHub is (*n*-*p*)*p* * (*n*-*p*)(*p*-1) as numerical example in table 2.3. On the other hand, tabu search only performs the reallocation procedures on the best hubsets. The number of neighbors generated from the Tabu search algorithm is (*n*-*p*)*p* + 2 (*n*-*p*)(*p*-1) as illustrated in the numerical example in table 2.4.

Table 2.4 Number of neighborhoods generated from Tabu search algorithm

Single Loc Exchange Hub set			Shortest Allocation Representation	Reallocation I (neighborhoods)	Reallocation II (neighborhoods)
1	7	8	1 8 7 8 1 1 7 8 7 7	choose the best hub set to reallocate 14	14
2	7	8	2 2 8 7 2 8 2 8 7 7		
.	7	8	.		
.	7	8	.		
10	7	8	7 7 10 8 7 10 7 8 7 10		
3	1	8	1 8 3 1 1 3 3 8 8 8		
3	2	8	3 2 3 3 2 3 3 8 8 8		
3	.	8	.		
3	.	8	.		
3	10	8	3 3 3 8 8 10 8 8 8 10		
3	7	1	1 3 3 7 1 3 7 7 7 7		
3	7	2	2 2 3 7 3 3 7 7 7 7		
3	7	4	7 4 3 4 7 4 7 7 7 7		
3	7	.	.		
3	7	10	7 7 3 3 7 10 7 7 7 10		

5.6 Stopping criteria

To measure the quality of solution and control the computational time, two types of stopping criteria are combined: percent of gap and number of evaluations. First, the algorithm will be stopped when the gap is less than the setting value. The gap is defined as the percent above the optimal solution, when it is known, or the best known solution when the optimal solution is not known.

$$\%Gap = \left(\frac{Incumbent}{BestKnown(or\ Optimal\ Sol)} - 1\right) * 100$$

The setting value is 0.5 %, Second, the number of objective function evaluations is used as a stopping criterion if the gap is outside the setting value. For small problems ($n \leq 50$), we allow up to 1,000 iterations while we allow up to 2,000 iterations for large problems ($n \geq 100$). These values were chosen to allow the running times to be reasonable.

5.7 Number of replications

From the central limit theorem, the sample means are normally distributed when the sample size approached infinity and accepted approximation when $n \leq 30$. Therefore, the number of replications is 30. We set the population size as 100 in the genetic algorithm,

6. Caching

Due to the complexity in computing an objective evaluation, caching is applied to speed up the algorithms and reduce some repetitive computations. Caching is a technique intended to improve the running time of an algorithm by calling the objective value in the caching list to avoid repeatedly calling the objective evaluation function. In this work, the cache size is based on the number of hubs and the number of nodes in each problem, and is set as 2**np*, based on empirical experience. To call the objective from the repeatedly generated solution, the caching list consists of two sections: a representation of the hub set and an objective list.

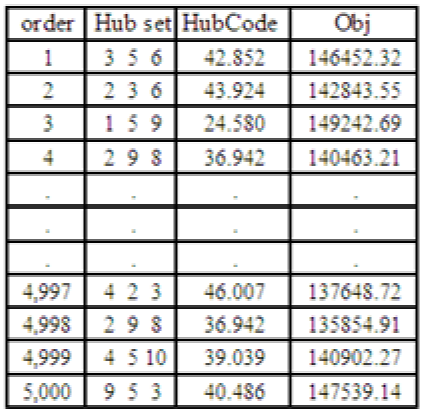
In the caching process, hub set solutions are converted to hub codes, keeping them and their objective values in the caching list. To represent each hub set, the encoding solutions of the hub sets that will become the hub codes are based on $\sum_{i=hub} \frac{1}{i} \log i \times 100$ where i is the hub node. For example, for hub set (3 5 6), we compute the hub code as follows $\left(\frac{1}{3}\log 3 + \frac{1}{5}\log 5 + \frac{1}{6}\log 6\right) \times 100 = 42.852$

```
Caching procedure,
  If the objective is found in the caching list,
     then objective value = Call (hub code);
  else
     Call Evaluate objective function;
     Add ( hub code, objective value);
     If cache list is full
          then Remove (the oldest hub code, objective value)
     end if
  end if
```

Figure 2.7 Caching procedure

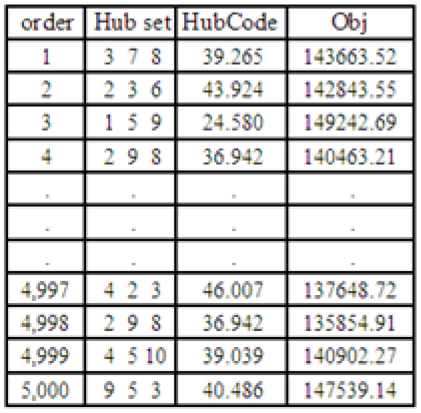
The hub codes and objective values are stored in a data structure called a red-black tree. When a hub set is generated, its hub code is generated. If its hub code is not found in the caching list, the evaluate function is called to calculate the objective value and to store the items into the caching list in the red-black tree. Whenever the caching list is full, the oldest entry of the list is replaced by a new hub set and the objective value.

Table 2.5 Updating the cache list



order	Hub set	HubCode	Obj
1	3 5 6	42.852	146452.32
2	2 3 6	43.924	142843.55
3	1 5 9	24.580	149242.69
4	2 9 8	36.942	140463.21
.	.	.	.
.	.	.	.
.	.	.	.
4,997	4 2 3	46.007	137648.72
4,998	2 9 8	36.942	135854.91
4,999	4 5 10	39.039	140902.27
5,000	9 5 3	40.486	147539.14

← oldest posittion



order	Hub set	HubCode	Obj
1	3 7 8	39.265	143663.52
2	2 3 6	43.924	142843.55
3	1 5 9	24.580	149242.69
4	2 9 8	36.942	140463.21
.	.	.	.
.	.	.	.
.	.	.	.
4,997	4 2 3	46.007	137648.72
4,998	2 9 8	36.942	135854.91
4,999	4 5 10	39.039	140902.27
5,000	9 5 3	40.486	147539.14

← oldest posittion

In order to update the cache list, the oldest position of the hub set (3 5 6) and the objective at the oldest position in the list are replaced with a new hub set and the objective value. The computational time of an algorithm may be sped up by caching. To search and retrieve objective values in caching, the operation time such as creating, storing, and searching in the caching list must be fast when compared with directly computing an objective function. Accordingly, the efficiency of caching procedures depends on the data structure. The objective values are stored in the caching list using the red-black tree (see in Appendix) as a data structure of the caching list to guarantee that each access time is within *O(log(n))*.

7. Computational Experiments

To evaluate the computational results of GA, SA, and TS, we use many sizes of the AP data set. Each node represents a post district in the Australian postal system. The data set contains 200 nodes, with smaller problems created by selecting subsets of those 200 nodes. For each problem size, the number of hubs is equal to 5, 10, 15, or 20 hubs. Experimentations with each type of metaheuristic consist of four versions: permutation, permutation with caching, binary, and binary with caching. Each algorithm is run 30 times with independent random number streams except TS which has no randomness. Nearly all algorithms obtain optimal solutions for small problems (10, 20, 40, and 50 nodes). For large problems (100 and 200 nodes), TS finds nearly optimal solutions for every problem except the problems with 200 nodes and 20 hubs. The computational results of SA, GA, and TS are shown in table 2.6, 2.7 and 2.8 respectively. In each table, the best known solution is shown. They are from Ernst (1996), with the exception of those marked with an asterisk, which are from Kratica (2007).

Table 2.6 Results of the SA (Averages over 30 replications)

			Permutation SA no caching				Permutation SA with caching				Binary SA no caching				Binary SA with caching			
n	*p*	Best known	%gap	stdGap	time(s)	stdTime	%gap	stdGap	time(s)	stdTime	%gap	stdGap	time(s)	stdTime	%gap	stdGap	time(s)	stdTime
10	2	167493.06	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	3	136008.13	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	4	112396.07	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20	2	172816.69	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	3	151533.08	0.00	0.00	0.03	0.01	0.00	0.00	0.01	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.01	0.00
	4	135624.88	0.00	0.00	0.04	0.04	0.00	0.00	0.03	0.01	0.00	0.00	0.04	0.01	0.00	0.00	0.02	0.10
40	2	177471.67	0.00	0.00	0.01	0.00	0.00	0.00	0.01	0.01	0.00	0.00	0.01	0.00	0.00	0.00	0.01	0.00
	3	158830.54	0.00	0.00	0.23	0.14	0.00	0.00	0.16	0.10	0.00	0.00	0.24	0.09	0.00	0.00	0.17	0.10
	4	143968.88	0.00	0.00	0.29	0.32	0.00	0.00	0.30	0.01	0.00	0.00	0.35	0.12	0.00	0.00	0.32	0.30
50	2	178484.29	0.03	0.04	2.50	0.00	0.03	0.06	2.09	0.01	0.04	0.06	2.78	0.01	0.04	0.05	2.26	0.06
	3	158569.93	0.00	0.00	0.55	0.00	0.00	0.00	0.47	0.06	0.00	0.00	0.59	0.00	0.03	0.02	0.51	0.00
	4	143378.05	0.01	0.06	1.68	0.02	0.00	0.00	1.14	0.12	0.03	0.01	1.25	0.05	0.01	0.03	1.14	0.01
100	5	136929.44	0.00	0.01	123.43	0.63	0.00	0.00	112.85	3.50	0.02	0.01	137.00	0.77	0.00	0.00	115.62	3.08
	10	106469.57	2.87	1.01	206.10	1.20	1.80	1.13	202.44	1.06	2.60	0.82	230.48	1.02	2.62	1.42	187.82	1.27
	15	90534.00 [J]	4.65	1.54	120.66	1.05	4.46	1.36	113.82	1,54	5.71	0.94	172.53	1.52	4.73	0.87	114.62	1.36
	20	80270.10 [J]	6.96	1.39	122.29	4.63	6.30	0.63	115.00	3.33	6.60	2.19	310.60	1.60	6.42	1.22	285.90	1.62
200	5	140175.65[J]	0.29	0.12	486.73	2.97	0.20	0.15	455.82	4.23	0.31	0.12	538.85	3.52	0.23	0.08	462.58	3.04
	10	110147.66[J]	1.46	1.39	458.77	4.62	2.02	1.08	446.69	4.72	5.05	1.33	506.06	3.72	3.37	0.85	447.66	6.29
	15	94496.406[J]	4.22	2.35	462.38	6.18	5.43	1.77	434.96	6.83	3.27	2.01	492.95	8.62	3.87	2.33	439.00	7.01
	20	85129.3[J]	4.54	2.38	480.64	7.43	4.37	2.19	457.57	10.95	6.05	2.26	535.05	12.16	6.34	2.35	470.88	7.67

*J. Kratica (2007)

Table 2.7 Results of the GA (Averages over 30 replications)

			Permutation GA no caching				Permutation GA with caching				Binary GA no caching				Binary GA with caching			
n	*p*	Best known	%gap	stdGap	time(s)	stdTime	%gap	stdGap	time(s)	stdTime	%gap	stdGap	time(s)	stdTime	%gap	stdGap	time(s)	stdTime
10	2	167493.06	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	3	136008.13	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	4	112396.07	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20	2	172816.69	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	3	151533.08	0.00	0.00	3.06	0.01	0.00	0.00	1.39	0.01	0.00	0.00	0.03	0.01	0.00	0.00	0.02	0.01
	4	135624.88	0.00	0.00	9.35	0.01	0.00	0.00	3.32	0.01	0.00	0.00	0.03	0.01	0.00	0.00	0.03	0.01
40	2	177471.67	0.00	0.00	12.24	0.03	0.00	0.00	5.34	0.01	0.00	0.00	0.16	0.00	0.00	0.00	0.02	0.00
	3	158830.54	0.00	0.00	15.06	0.08	0.00	0.00	6.59	0.06	0.00	0.00	0.42	0.02	0.00	0.00	0.02	0.00
	4	143968.88	0.00	0.00	22.33	0.34	0.00	0.00	12.68	0.04	0.00	0.00	1.65	0.08	0.00	0.00	0.08	0.01
50	2	178484.29	0.00	0.00	22.64	0.52	0.00	0.00	15.45	0.16	0.00	0.00	0.55	0.03	0.00	0.00	0.35	0.02
	3	158569.93	0.05	0.00	28.26	0.78	0.01	0.00	23.23	0.84	0.00	0.00	0.84	0.07	0.00	0.00	0.71	0.01
	4	143378.05	0.31	0.04	51.26	0.63	0.02	0.00	40.83	0.95	0.00	0.00	12.06	0.18	0.00	0.03	4.73	0.02
100	5	136929.44	1.69	1.08	441.18	0.44	1.46	0.82	392.04	0.95	0.01	0.00	429.91	1.05	0.18	0.02	426.75	2.89
	10	106469.57	7.61	1.47	461.83	1.21	7.57	1.38	412.16	2.10	1.07	0.34	433.19	1.49	0.89	0.32	430.52	2.25
	15	90534.00 [J]	9.98	2.28	484.55	1.63	11.36	2.47	431.93	2.17	0.63	0.30	438.08	2.46	1.42	0.44	435.68	3.23
	20	80270.10 [J]	10.83	3.03	512.33	1.98	12.73	2.72	453.87	2.23	0.85	0.38	444.09	3.06	1.26	0.50	439.32	3.4
200	5	140175.65[J]	0.91	0.78	1731.55	2.16	2.13	0.94	1472.85	2.57	0.33	0.17	1700.03	20.84	0.44	0.12	1688.35	22.9
	10	110147.66[J]	7.96	2..27	1767.44	2.24	8.50	2.60	1704.59	3.08	0.55	0.24	1733.64	12.46	1.52	0.66	1708.50	8.51
	15	94496.406[J]	12.59	1.60	1813.35	1.28	12.61	1.93	1651.08	3.16	1.58	0.76	1732.49	7.96	1.55	0.74	1713.68	6.91
	20	85129.3[J]	14.16	2.09	1880.37	2.23	14.57	2.15	1695.28	2.21	1.96	1.26	1740.69	8.37	1.64	1.23	1688.35	9.18

*J. Kratica (2007)

Table 2.8 Results of the TS (Averages over 30 replications)

			Permutation TS no caching		Permutation TS with caching		Binary TS no caching		Binary TS with caching	
N	*P*	Best known	%gap	time(s)	%gap	time(s)	%gap	time(s)	%gap	time(s)
10	2	167493.06	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	3	136008.13	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	4	112396.07	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20	2	172816.69	0.00	0.03	0.00	0.00	0.00	0.02	0.00	0.00
	3	151533.08	0.00	0.03	0.00	0.02	0.00	0.11	0.00	0.02
	4	135624.88	0.00	0.03	0.00	0.02	0.00	0.02	0.00	0.02
40	2	177471.67	0.00	0.08	0.00	0.02	0.00	0.05	0.00	0.02
	3	158830.54	0.00	0.14	0.00	0.05	0.00	0.08	0.00	0.05
	4	143968.88	0.00	0.16	0.00	0.08	0.00	0.11	0.00	0.08
50	2	178484.29	0.00	0.20	0.00	0.13	0.00	0.11	0.00	0.27
	3	158569.93	0.00	0.30	0.00	0.20	0.00	0.23	0.00	0.16
	4	143378.05	0.00	1.11	0.00	0.56	0.00	1.09	0.00	0.72
100	5	136929.44	0.00	0.94	0.00	0.81	0.00	1.99	0.00	1.02
	10	106469.57	0.00	22.67	0.00	19.40	0.00	45.50	0.00	20.12
	15	90534.00 [J]	0.00	27.29	0.00	23.98	0.00	53.34	0.00	24.15
	20	80270.10 [J]	0.00	10.63	0.00	9.74	0.00	68.82	0.00	9.84
200	5	140175.65[J]	0.00	94.86	0.00	56.87	0.00	116.23	0.00	57.45
	10	110147.66[J]	0.00	299.47	0.00	254.23	0.00	381.57	0.00	243.05
	15	94496.406[J]	0.00	273.08	0.00	219.25	0.00	328.21	0.00	222.25
	20	85129.3[J]	0.23	323.46	0.23	258.42	0.23	405.21	0.23	260.93

*J. Kratica (2007)

8. Analysis

To evaluate the impact of caching on computational time and quality of each algorithm, we consider the use of statistical methods. If normality assumption are not satisfied, and computational results of algorithms are continuous variables, then Wilcoxon Signed Rank test is used. Wilcoxan signed rank test are tested in section 8.1 and 8.2. The results from each algorithm in the same representation are compared to investigate the effect of caching and type of representation in tables 2.10 and 2.11. Gap and computational time of GA, SA, and TS are shown in figure 2.8 and 2.9 respectively.

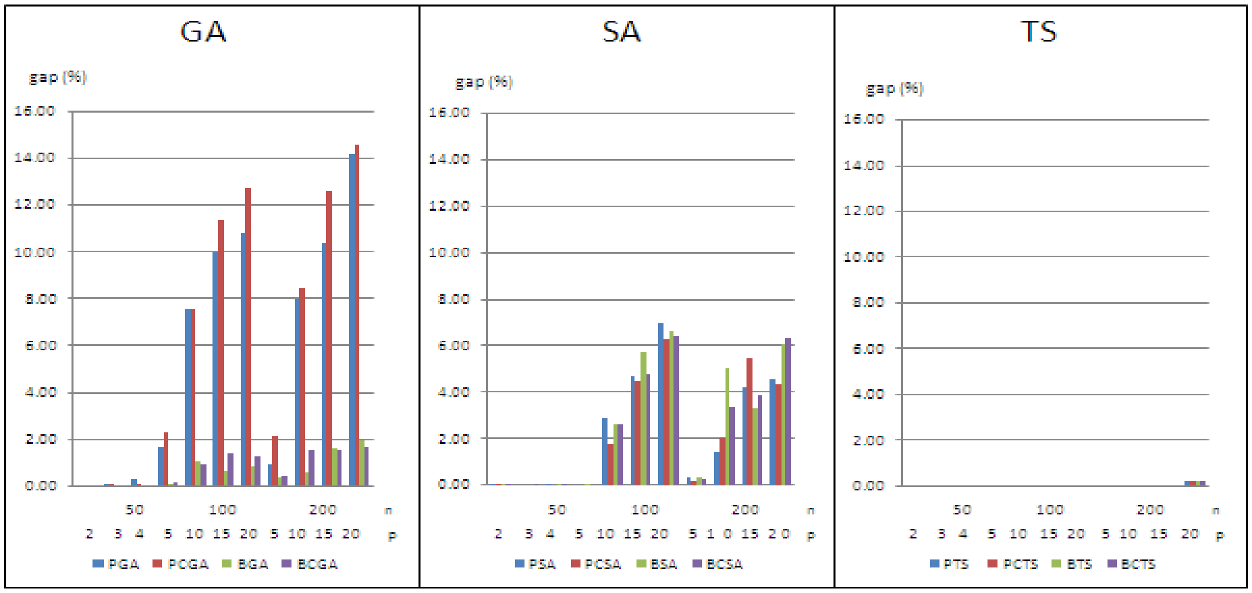


Figure 2.8 Gap of GA, SA, and TS

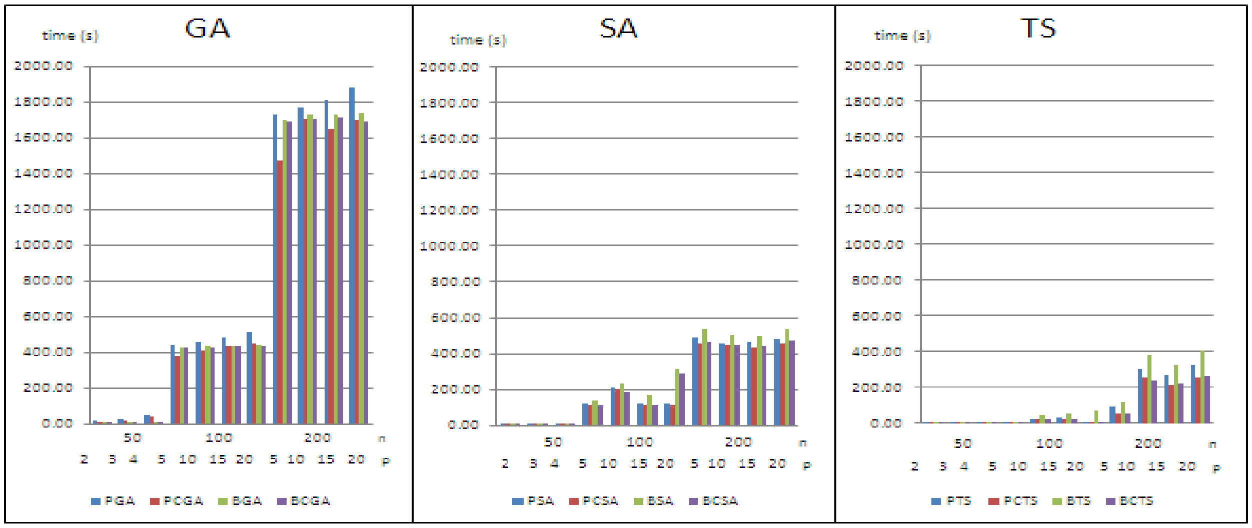


Figure 2.9 Computational time of GA, SA, and TS

8.1 Evaluating the impact of caching on running time

If the quality of solutions is similar, caching should only be used if the running time is no longer with caching. To investigate the impact of using caching on running time, the Wilcoxon signed rank test is applied for comparisons on computational time between no caching (nC) and caching version (C) for each algorithm. We illustrate the Wilcoxon signed rank test comparing the running time of the non-caching and caching binary GA in table 2.9.

Table 2.9 Wilcoxon signed rank test for the time of binary GA with (C) and without caching (nC)

		Binary GA				Signed
n	*P*	nC time(s)	C time(s)	abs diff	Rank	Rank
10	2	0	0	0.00	-	-
10	3	0	0	0.00	-	-
10	4	0	0	0.00	-	-

Table 2.9 (continued). Wilcoxon signed rank test for the time of binary GA with (C) and without caching (nC)

		Binary GA				Signed
n	*P*	nC time(s)	C time(s)	abs diff	Rank	Rank
20	2	0	0	0.00	-	-
20	4	0.03	0.03	0.00	-	-
20	3	0.03	0.02	0.01	1	1
50	3	0.84	0.71	0.13	2	2
40	2	0.16	0.02	0.14	3	3
50	2	0.55	0.35	0.20	4	4
40	3	0.42	0.02	0.40	5	5
40	4	1.65	0.08	1.57	6	6
100	15	438.08	435.68	2.40	7	7
100	10	433.19	430.52	2.67	8	8
100	5	429.91	426.75	3.16	9	9
100	20	444.09	439.32	4.77	10	10
50	4	12.06	4.73	7.33	11	11
200	5	1700.03	1688.35	11.68	12	12
200	15	1732.49	1713.68	18.81	13	13
200	10	1733.64	1708.5	25.14	14	14
200	20	1740.69	1688.35	52.34	15	15

Hypothesis

H_0: means of the times of binary GA with and without caching are the same

H_1: means of the times of binary GA without caching are more than that from binary GA with caching

In this example, we have N_{test} = 15 different running times over the set of N=20 experimental results. The sum of the ranks W = 120. To compute the value of the test statistic with α = 0.05, we use the following formula.

$$W_{1-\alpha} = Z_{1-\alpha}\sqrt{\frac{n(n+1)(2n+1)}{6}} \quad = \quad (1.65) * \sqrt{1240} = 58.10$$

In this case W > $W_{1-\alpha}$, so we reject H_0, and conclude that the means of the times from binary GA with no caching solutions are more than that from binary GA with caching solutions.

The Wilcoxon signed rank test is also applied for time comparisons between the caching and non-caching versions of all algorithms, as shown in table 2.10. We find that implementing caching implementation in each algorithm can speedup running time significantly.

Table 2.10 Time comparisons between no caching and caching algorithm for each algorithm

Comparison		N	N_{test}	Wilcoxon statistic	$W_{1-\alpha}$
GA time	BGA – BCGA	20	15	120	58.1*
	pGA – pCGA	20	16	136	61.9*
SA time	BSA – BCSA	20	15	120	58.1*
	pSA – pCSA	20	15	118.5	58.1*
TS time	BTS – BCTS	20	16	129	61.9*
	pTS – pCTS	20	17	153	69.71*

8.2 Evaluating metaheuristics performance

To evaluate performance of each algorithm, the percentage of gaps obtained from GA and SA in each representation with and without caching are analyzed by Wilcoxon signed rank test and are displayed as statistical results in table 2.11. TS gaps are the same due to the fact that it has no randomness.

Table 2.11 % Gap comparisons

Comparison		N	N_{test}	Wilcoxon statistic	$W_{1-\alpha}$
GA gap	BCGA - BGA	20	8	16	23.56
	pCGA - pGA	20	10	27	32.37
	pGA - BCGA	20	10	55	32.37*
SA gap	BSA – BCSA	20	10	28	32.37
	pSA – pCSA	20	8	23	23.56
	pSA- BSA	20	10	18	32.37

*Significantly different

Table 2.12 Time comparisons

Comparison		N	N_{test}	Wilcoxon statistic	$W_{1-\alpha}$
GA time	pGA – pCGA	20	16	136	61.9*
	BGA - BCGA	20	15	120	58.1*
	pCGA - BCGA	20	16	4	61.9
SA time	BSA – BCSA	20	15	120	58.1*
	pSA – pCSA	20	15	118.5	58.1*
	BCSA – pCSA	20	13	77.5	47.22*
TS time	BTS – BCTS	20	16	129	61.9*
	pTS – pCTS	20	17	153	69.71*
	BCTS – pCTS	20	11	54	37.12*

*Significantly different

Table 2.13 Winner comparisons

Comparison		N	N test	Wilcoxon statistic	$W_{1-\alpha}$
Gap	PCSA - BCGA	20	9	40	27.86*
	BCGA - PCTS	20	8	36	23.56*
Time	PCSA - BCGA	20	15	13	58.1
	PCSA - PCTS	20	13	89	47.22*

*Significantly different

8.3 Discussion

Based on Table 2.11, we see that there is not a statistically significant difference in performance between the binary GAs with and without caching. Since there is a statistically significant difference in running time between the binary GAs with and without caching, with the binary caching GA being faster (as shown in Table 2.12), we prefer the binary caching GA.

In the permutation GA analysis, there is not a statistically significant difference between the performance of the permutation GAs with and without caching. Since there is a statistically significant difference in running time between the binary SAs with and without caching, the permutation GA with caching is faster, we prefer the permutation caching GA.

When comparing the binary caching GA and the permutation GA without caching based on performance, we find the binary caching GA performs better. However, there is not a statistically significant difference in running time between the binary caching and the permutation caching GAs. Therefore, we conclude that the recommended GA is the binary caching GA.

Based on Table 2.11, we see that there is not a statistically significant difference in performance between the binary SAs with and without caching. Since there is a statistically significant difference in running time between the binary SAs with and without caching, with the binary caching SA being faster (as shown in Table 2.12), we prefer the binary caching SA.

In the permutation SA analysis, we find the same results: there is not a statistically significant difference between the performance of the permutation SAs with and without caching while there is a statistically significant difference in the running times, with the permutation caching SA being faster. Therefore, we prefer the permutation caching SA

When comparing the binary caching SA and the permutation caching SA based on performance, we find no statistically significant difference. However, there is a statistically significant difference in running time between the binary caching SA and the permutation caching SA. Therefore, we conclude that the recommended SA is the permutation caching SA.

Since the TS has no randomness, there is no difference between the performance of the permutation TSs with and without caching while there is a statistically significant difference in the running times, with the permutation caching TS being faster. Therefore, we prefer the permutation caching TS

In the binary TS analysis, there is no difference between the performance of the binary TSs with and without caching. However, there is a statistically significant difference in running time between the binary TSs with and without caching, with the binary caching TS being faster (as shown in Table 2.12), we prefer the binary caching TS.

When comparing the binary caching TS and the permutation caching TS based on computational time, we find statistically significant difference in the running times between the binary caching TS and the permutation caching TS, with the permutation caching TS being faster, we prefer the permutation caching TS.

Now, we compare the recommended GA, SA and TS versions to each other. When comparing the binary caching GA and the permutation caching SA based on performance, we find the binary caching GA performs better.

Based on Table 2.13, there are statistically significant differences between the running times of the binary caching GA and the permutation caching SA: the permutation SA with caching is faster but the binary caching GA performs better, we prefer the binary caching GA.

There is a statistically significant difference between the running time of the permutation caching TS with and the binary caching GA, we find the permutation caching TS being faster. In the performance analysis, we find there is a statistically significant difference between the performance of the permutation caching TS and the binary caching GA, with the permutation caching TS performs better. Therefore, we conclude that the recommended algorithm is the permutation caching TS.

Comparative performance of TABUHUB and tabu search

Based on computational results of TABUHUB (Skorin. 1994 and Perez, 2004), the results clearly demonstrate that TS performs better than TABUHUB both solution quality and computational times. In addition, TS is capable to solve larger scale AP instances up to 200 nodes.

9. Conclusions

GA, SA, and TS were proposed to solve the USApHMP problem. The results show that caching greatly affects the computational time of GA, SA, and TS. Consequently, TS performance is competitive among other algorithms in terms of computational time and solution quality. We hypothesize a few reasons why TS outperforms the other algorithms. First, TS has strength in its deterministic scheme, which focuses on necessary stages by using adaptive memory to avoid local optima and long-term memory to expand the search space. Second, the single exchange location method is effective because it considers the initial hub set based on the amount of flow to find the optimal location of hubs in a short time. Accordingly, all TS algorithms reach smaller gaps in shorter periods of time compared with GAs and SAs. GA has gains in its ability to both randomly search to explore more space and to hill climb to select highly fit individuals to create the next generation. Therefore, GA can produce good quality solutions in a reasonable time as well. However, the representation as binary structures provide more diversity of solutions from the crossover procedure than from those developed from permutation, so the percentage of gaps of binary GA are smaller than that of permutation. There are some advantages for SA implementation, as well. First, SA allows non-improving moves via probabilistic acceptance to avoid local optimum. Second, the structure of SA is based on simplification and modification, so it only needs a few decision parameters compared to GA and TS. In these problems, the reheating procedure is applied to improve the quality of the solution when the instant temperature is low and SA converges to local optimal. For small problems, SA reaches optimality very fast; however, in large-sized problems, SA tends to be restricted in terms of search space.

In USApHMP, to choose p hubs from n nodes, the number of possible hub sets is equal to $\frac{n!}{(n-p)!\,p!}$. After one hub set is obtained, (n-p)(p-1) possible solutions are generated along with reallocation procedure. In this case, we speculate that the caching technique is effective because the sub-problem is an assignment problem.

CHAPTER THREE

PATH RELINKING FOR HUB LOCATION

1. Introduction

Basic metaheuristics for USApHMP perform very effectively in some cases. In general, when the size of the problem is small (10, 20, 40, and 50 nodes), many metaheuristics perform competitively. However, when the size of the problem is larger (more than 100 nodes), the quality of the solutions tends to be decreased. Therefore, hybrid metaheuristics are introduced to increase the effectiveness of the algorithms. Path-relinking metaheuristics are implemented with GA, SA and TS algorithms as hybrid algorithms. In this chapter, Path Relinking metaheuristics are applied to investigate large instances of USApHMP to generate high quality of solutions in a reasonable computational time.

2. Path Relinking approach

Path Relinking (PR) was introduced by Glover in 1996 to better explore a specified space to intensify and diversify the solution space (Glover and Laguna,1997). The boundary of the search space is defined by two solutions: the initial solution and guiding solution (or reference set). In general, the path moves from one of the solutions, generates a neighborhood, and gradually selects solutions from the neighborhood toward to another solution. PR has been applied to solve complex optimization problems. Juan and Elena (2006) investigated the capacitated *p*-median problem, and implemented PR to enhance the performance of scatter search. Perez et al. (2004) adapted PR to solve large instance problems in the USApHMP. They found that their algorithm efficiently performs up to 100 nodes. The same authors also implemented PR as a hybrid algorithm based on neighborhood of variable neighborhood search which are shaking, local search and neighborhood exchange. The algorithm is applied to solve the USApHMP up to 200 nodes (Perez et al., 2007).

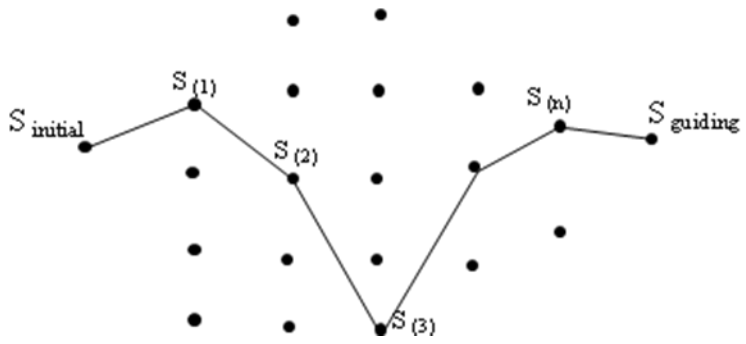


Figure 3.1 Path Relinking

3. Path Relinking approach for USApHMP

Path Relinking performs on a reference set that consists of the initial solution and guiding solutions to identify the solution space. Important factors to design effective PR method include choosing the initial and guiding solutions and generating neighborhood structure.

3.1 Choosing the initial and guiding solutions

Several considerations to select the reference set are discussed in the literature (Ghamlouche et al., 2004). First, the generating reference set should consist of the local optima derived from construction algorithms to share common characteristics with the optimal solution. Second, good solutions have characteristics that should be retained while allowing diversified solution in relation to the reference set. Finally, diversification and intensification is ensured by starting with a good quality solution from base algorithms and moving to a diversified solution. In PR implementation, the guiding solution is chosen from the best solution obtained from the base algorithm to intensify the attractive solution and the initial solution is selected from a local optimal solution derived from a base algorithm.

3.2 Generating neighborhood structure

PR begins with a local optimal solution derived from local search and designated as the initial solution, which is then converted to the best solution found from local search or the guiding solution. The neighborhood structure is based on position-based path relinking, which moves the index of the initial solution corresponding to the position in the target solution. The tunneling is built by gradually converting the initial solution into the guiding solution and backward moving from the guiding solution into the initial solution to explore more solutions in the search space.

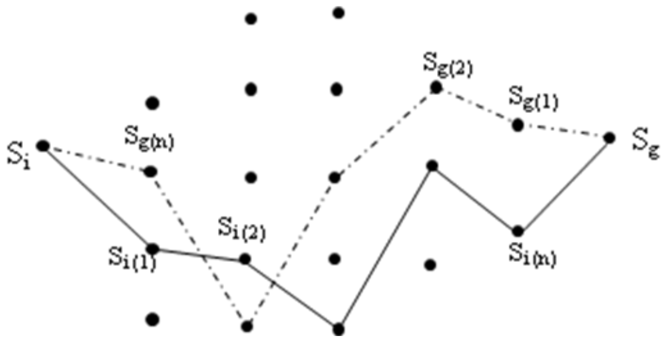


Figure 3.2 Path Relinking for USApHMP

In this research, we consider two neighbors for each solution: change the *i*th hub to a random non-hub node and change the *i*th hub to the ith hub in the reference solution. This procedure is continually repeated toward the guiding solution. Later, the path gradually moves backward from the guiding solution to be the initial solution to explore new solutions in the same path that share common characteristics with the optimal solution, by reversing the roles of the initial and guiding solutions

3.4 Numerical example

To illustrate, the initial solution is (14, 28, 32, 35) and the guiding solution is (33, 34, 35, 38).

initial solution	14	28	32	35	33	34	38	35
	14	28	32	**34**				
	14	28	32	**35**				
	14	28	**33**	35				
	14	28	**38**	35				
	14	**33**	38	35				
	14	**34**	38	35				
	12	34	38	35				
guiding solution	**33**	34	38	35				

Figure 3.3 Numerical example of Path relinking for hub location

The process begins with inserting the single random hub 34 in the 4^{th} position, the last position of the current solution. The current solution is (14, 28, 32, 34). Next, insert hub 35, the last hub from the last position of the guiding solution, into the current solution. Now, the current solution is (14, 28, 32, 35). Then, the index of the position of the hub to be replaced is shifted and investigated in the same pattern, insert random single hub 33 to generate one neighborhood (14, 28, 33, 35) and insert the hub 38 of the same position in the guiding solution. Then, the current solution is (14, 28, 38, 35). This procedure is repeated until the neighborhood includes the guiding solution.

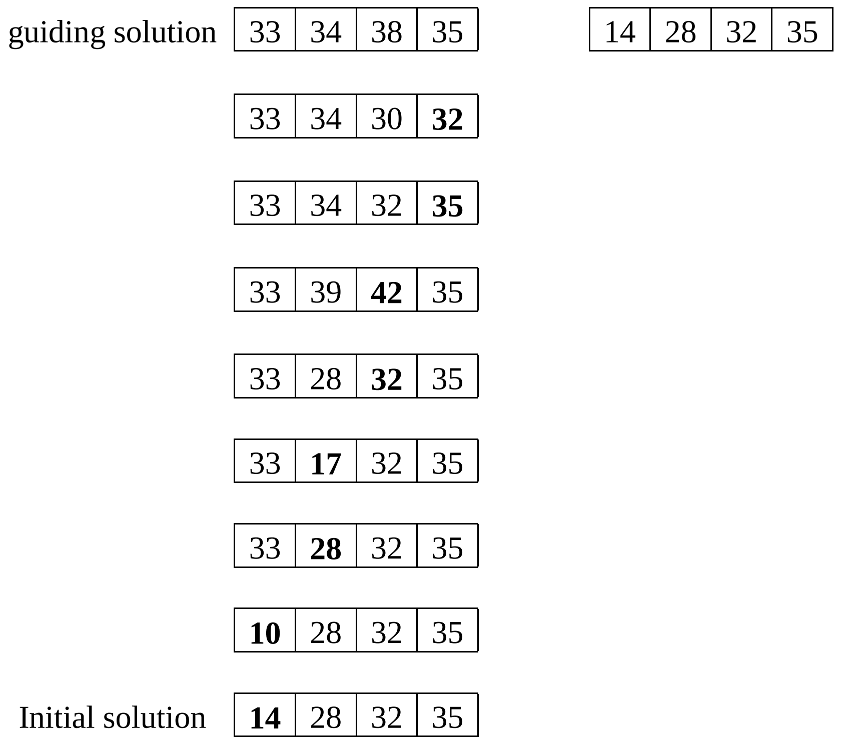


Figure 3.4 Numerical example of Path relinking for hub location (continue)

After the path moves through the tunneling, backward moving is consequently operated as shown in Figure 3.4, starting with the guiding solution (33, 34, 38, 35), the neighborhood moves are applied in reverse until the neighborhood includes the initial solution (14, 28, 32, 35).

```
Procedure Path Relinking-local search Algorithm
  Begin
        Build the reference set, $R$
          While (iterations < Max_ iter) do
              Randomly select local optimal solution $X_{j,}$
              $R = R$U $X_j$;
        Identify the initial solution, $S_0$ ;
        Identify the guiding solution, $S_g$ ;
    While($R \geq 1$) do
        for  $i = 1$ to $N$  do
            Generate neighborhood, $S_i$ , with position-based PR;
                Shortest allocation;
                Reallocation;
                If Z$(S_i) < Z(S^*)$,
                Then   $S^* = S_i$;
              If  $\Delta(S_i, S_g) = 0$,
              Then generate neighborhood, $S_{gi}$ ,to move backward;
              If  $\Delta(S_{gi}, S_0) = 0$, Then update reference set;
        Until($R = \varnothing$)
     End
```

Figure 3.5 PR-local search algorithms for USApHMP procedure

The path relinking procedure is shown in Figure 3.4. The algorithm starts with performing the local search algorithm iteration to obtain local optimal solution. Then, it identifies the initial solution, S_i and the guiding solution, S_g, from the best solution. Later, it performs position-based path relinking by generating elements in the neighborhood by sharing one of hubs of the target solution and inserting a single random hub. After the new hub set is generated, a non-hub node is connected to the nearest hub and evaluated. To reduce computational time, the reallocation function is only used when the objective value of the new hub set is within 5% of the current incumbent. Finally, the algorithm moves along the path toward to the guiding solution, and then moves backward to the original initial solution.

4. Computational experiment

In this section, we implement GAPR, SAPR, and TSPR to evaluate the performance of PR on each base algorithm. The problems are composed of 10, 20, 40, 50, 100, and 200 nodes based on AP data set. Each type of hybrid algorithm consists of four versions: permutation, permutation with caching, binary, and binary with caching. The path relinking hybrid algorithm enhance the quality of solution of almost all algorithms except TSPR. However, the qualities of solutions derived from path relinking are dependent on the quality of each construction algorithm. In addition, the improving percentage of gaps by implementing the path relinking is not completely high because characteristic of problem, USApHMP, is necessary to find every hub matches up to the hub set. The computational results of permutation and binary GA, permutation and binary SA, and permutation and binary TS are shown in table 3.1-3.6 respectively. Each algorithm was applied to each data set 30 times, and the averages are shown.

Table 3.1 Results of the Binary GAPR (Averages over 30 replications)

N	*P*	Best known	Binary GA no caching		Binary GAPR no caching		Binary GA with caching		Binary GAPR with caching	
			%gap	time(s)	%gap	time(s)	%gap	time(s)	%gap	time(s)
10	2	167493.06	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	3	136008.13	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	4	112396.07	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20	2	172816.69	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	3	151533.08	0.00	0.03	0.00	0.01	0.00	0.02	0.00	0.01
	4	135624.88	0.00	0.03	0.00	0.01	0.00	0.03	0.00	0.00
40	2	177471.67	0.00	0.16	0.00	0.14	0.00	0.02	0.00	0.04
	3	158830.54	0.00	0.42	0.00	1.07	0.00	0.02	0.00	0.04
	4	143968.88	0.00	1.65	0.00	2.37	0.00	0.08	0.00	1.52
50	2	178484.29	0.00	0.55	0.00	1.56	0.00	0.35	0.00	1.34
	3	158569.93	0.00	0.84	0.00	2.44	0.00	0.71	0.00	1.45
	4	143378.05	0.00	12.06	0.51	21.63	0.00	4.73	0.00	22.01
100	5	136929.44	0.01	429.91	0.16	437.22	0.18	426.75	0.15	214.03
	10	106469.57	1.07	433.19	0.08	360.82	0.89	430.52	0.41	353.42
	15	90534.00 [J]	0.63	438.08	0.73	584.39	1.42	435.68	0.84	573.72
	20	80270.10 [J]	0.85	444.09	0.84	535.35	1.26	439.32	1.05	527.50
200	5	140175.65[J]	0.33	1700.03	0.28	2182.70	0.44	1688.35	0.64	2044.20
	10	110147.66[J]	0.55	1733.64	0.81	2203.08	1.52	1708.50	1.57	2159.04
	15	94496.406[J]	1.58	1732.49	1.52	2264.63	1.55	1713.68	1.48	2223.90
	20	85129.3[J]	1.96	1740.69	1.92	2887.06	1.64	1688.35	1.92	2714.56

*J. Kratica (2007)

Table 3.2 Results of the Permutation GAPR (Averages over 30 replications)

N	P	Best known	Permutation GA no caching		Permutation GAPR no caching		Permutation GA with caching		Permutation GAPR with caching	
			%gap	time(s)	%gap	time(s)	%gap	time(s)	%gap	time(s)
10	2	167493.06	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	3	136008.13	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	4	112396.07	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20	2	172816.69	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	3	151533.08	0.00	3.06	0.00	3.32	0.00	1.39	0.00	3.19
	4	135624.88	0.00	9.35	0.00	10.05	0.00	3.32	0.00	7.04
40	2	177471.67	0.00	12.24	0.00	14.05	0.00	5.34	0.00	10.25
	3	158830.54	0.00	15.06	0.00	16.24	0.00	6.59	0.00	14.21
	4	143968.88	0.00	22.33	0.00	20.70	0.00	12.68	0.00	21.78
50	2	178484.29	0.00	22.64	0.00	23.90	0.00	15.45	0.45	15.80
	3	158569.93	0.05	28.26	0.05	32.57	0.01	23.23	0.46	30.79
	4	143378.05	0.31	51.26	0.52	63.52	0.02	40.83	1.40	46.94
100	5	136929.44	1.69	441.18	1.63	472.31	2.29	377.38	2.42	374.07
	10	106469.57	7.61	461.83	7.28	482.72	7.57	412.16	6.43	324.33
	15	90534.00 [J]	9.98	484.55	6.66	517.19	11.36	431.93	9.73	342.69
	20	80270.10 [J]	10.83	512.33	10.83	572.54	12.73	453.87	11.65	482.14
200	5	140175.65[J]	0.91	1731.55	1.24	1865.31	2.13	1472.85	2.63	1706.63
	10	110147.66[J]	7.96	1767.44	5.74	2157.53	8.50	1704.59	8.52	2104.96
	15	94496.406[J]	10.36	1813.35	9.98	2298.26	12.61	1651.08	11.07	2164.24
	20	85129.3[J]	14.16	1880.37	12.39	2916.00	14.57	1695.28	14.45	2723.28

*J. Kratica (2007)

Table 3.3 Results of the Binary SAPR (Averages over 30 replications)

N	*P*	Best known	Binary SA no caching		Binary SAPR no caching		Binary SA with caching		Binary SAPR with caching	
			%gap	time(s)	%gap	time(s)	%gap	time(s)	%gap	time(s)
10	2	167493.06	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	3	136008.13	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	4	112396.07	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20	2	172816.69	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	3	151533.08	0.00	0.02	0.00	0.01	0.00	0.01	0.00	0.01
	4	135624.88	0.00	0.04	0.00	0.03	0.00	0.02	0.00	0.02
40	2	177471.67	0.00	0.01	0.00	0.02	0.00	0.01	0.00	0.02
	3	158830.54	0.00	0.24	0.00	0.18	0.00	0.17	0.00	0.06
	4	143968.88	0.00	0.35	0.00	0.44	0.00	0.32	0.00	0.36
50	2	178484.29	0.04	2.78	0.05	3.81	0.04	2.26	0.00	2.63
	3	158569.93	0.00	0.59	0.00	5.32	0.03	0.51	0.03	3.24
	4	143378.05	0.03	1.25	0.01	1.40	0.01	1.14	0.01	1.31
100	5	136929.44	0.02	137.00	0.00	128.93	0.00	115.62	0.00	111.42
	10	106469.57	2.60	230.48	2.56	240.61	2.62	187.82	1.87	213.56
	15	90534.00 ^{J}	5.71	172.53	3.23	183.59	4.73	114.62	3.12	139.63
	20	80270.10 ^{J}	6.60	310.60	4.81	412.54	6.42	285.90	5.93	247.39
200	5	140175.65^{J}	0.31	538.85	0.16	872.89	0.23	462.58	0.23	765.78
	10	110147.66^{J}	5.05	506.06	3.39	1865.53	3.37	447.66	2.73	1273.24
	15	94496.406^{J}	3.27	492.95	3.20	1822.57	3.87	439.00	3.28	1075.33
	20	85129.30^{J}	6.05	535.05	4.58	2153.42	6.34	470.88	4.89	1286.47

*J. Kratica (2007)

Table 3.4 Results of the Permutation SAPR (Averages over 30 replications)

N	*P*	Best known	Permutation SA no caching		Permutation SAPR no caching		Permutation SA with caching		Permutation SAPR with caching	
			%gap	time(s)	%gap	time(s)	%gap	time(s)	%gap	time(s)
10	2	167493.06	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	3	136008.13	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	4	112396.07	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20	2	172816.69	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	3	151533.08	0.00	0.03	0.00	0.01	0.00	0.01	0.00	0.01
	4	135624.88	0.00	0.04	0.00	0.02	0.00	0.03	0.00	0.02
40	2	177471.67	0.00	0.01	0.00	0.01	0.00	0.01	0.00	0.01
	3	158830.54	0.00	0.23	0.00	0.12	0.00	0.16	0.00	0.05
	4	143968.88	0.00	0.29	0.00	0.48	0.00	0.30	0.00	0.36
50	2	178484.29	0.03	2.50	0.00	3.44	0.03	2.09	0.00	2.74
	3	158569.93	0.00	0.55	0.00	0.62	0.00	0.47	0.00	0.34
	4	143378.05	0.01	1.68	0.00	2.63	0.00	1.14	0.00	2.30
100	5	136929.44	0.00	123.43	0.00	124.30	0.00	112.85	0.00	113.49
	10	106469.57	2.87	206.10	2.36	214.50	1.80	202.44	1.78	211.56
	15	90534.00 ^{J}	4.65	120.66	3.23	142.34	4.46	113.82	3.08	140.92
	20	80270.10 ^{J}	6.96	122.29	4.75	202.53	6.30	115.00	4.26	200.65
200	5	140175.65^{J}	0.29	486.73	0.24	687.87	0.20	455.82	0.16	573.92
	10	110147.66^{J}	1.46	458.77	1.44	1344.28	2.02	446.69	2.02	1562.14
	15	94496.406^{J}	4.22	462.38	3.65	1208.62	5.43	434.96	3.04	1266.43
	20	85129.3^{J}	4.54	480.64	4.21	1945.77	4.37	457.57	4.30	1646.08

*J. Kratica (2007)

Table 3.5 Results of the Binary TSPR (Averages over 30 replications)

			Binary TS no caching		Binary TSPR no caching		Binary TS with caching		Binary TSPR with caching	
N	*P*	Best known	%gap	time(s)	%gap	time(s)	%gap	time(s)	%gap	time(s)
10	2	167493.06	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	3	136008.13	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	4	112396.07	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20	2	172816.69	0.00	0.02	0.00	0.02	0.00	0.00	0.00	0.00
	3	151533.08	0.00	0.11	0.00	0.10	0.00	0.02	0.00	0.00
	4	135624.88	0.00	0.02	0.00	0.10	0.00	0.02	0.00	0.00
40	2	177471.67	0.00	0.05	0.00	0.10	0.00	0.02	0.00	0.06
	3	158830.54	0.00	0.08	0.00	0.10	0.00	0.05	0.00	0.10
	4	143968.88	0.00	0.11	0.00	0.40	0.00	0.08	0.00	0.10
50	2	178484.29	0.00	0.11	0.00	0.40	0.00	0.27	0.00	0.40
	3	158569.93	0.00	0.23	0.00	0.42	0.00	0.16	0.00	0.39
	4	143378.05	0.00	1.09	0.00	1.36	0.00	0.72	0.00	0.94
100	5	136929.44	0.00	1.99	0.00	2.48	0.00	1.02	0.00	1.16
	10	106469.57	0.00	45.50	0.00	49.53	0.00	20.12	0.00	20.19
	15	90534.00 [J]	0.00	53.34	0.00	54.22	0.00	24.15	0.00	25.41
	20	80270.10 [J]	0.00	68.82	0.00	70.42	0.00	9.84	0.00	12.38
200	5	140175.65[J]	0.00	116.23	0.00	118.56	0.00	57.45	0.00	58.66
	10	110147.66[J]	0.00	381.57	0.00	383.54	0.00	243.05	0.00	274.26
	15	94496.406[J]	0.00	328.21	0.00	329.40	0.00	222.25	0.00	223.48
	20	85129.3[J]	0.23	405.21	0.23	427.26	0.23	260.93	0.23	304.33

*J. Kratica (2007)

Table 3.6 Results of the permutation TSPR (Averages over 30 replications)

N	*P*	Best known	Permutation TS no caching		Permutation PRTS no caching		Permutation TS with caching		Permutation PRTS with caching	
			%gap	time(s)	%gap	time(s)	%gap	time(s)	%gap	time(s)
10	2	167493.06	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	3	136008.13	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	4	112396.07	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20	2	172816.69	0.00	0.03	0.00	0.03	0.00	0.00	0.00	0.00
	3	151533.08	0.00	0.03	0.00	0.06	0.00	0.02	0.00	0.06
	4	135624.88	0.00	0.03	0.00	0.06	0.00	0.02	0.00	0.06
40	2	177471.67	0.00	0.08	0.00	0.10	0.00	0.02	0.00	0.10
	3	158830.54	0.00	0.14	0.00	0.20	0.00	0.05	0.00	0.10
	4	143968.88	0.00	0.16	0.00	0.20	0.00	0.08	0.00	0.10
50	2	178484.29	0.00	0.20	0.00	0.24	0.00	0.13	0.00	0.14
	3	158569.93	0.00	0.30	0.00	0.32	0.00	0.20	0.00	0.25
	4	143378.05	0.00	1.11	0.00	1.27	0.00	0.56	0.00	0.69
100	5	136929.44	0.00	0.94	0.00	1.02	0.00	0.81	0.00	1.87
	10	106469.57	0.00	22.67	0.00	22.80	0.00	19.40	0.00	20.53
	15	90534.00 [J]	0.00	27.29	0.00	28.57	0.00	23.98	0.00	24.37
	20	80270.10 [J]	0.00	10.63	0.00	12.05	0.00	9.74	0.00	10.63
200	5	140175.65[J]	0.00	94.86	0.00	97.24	0.00	56.87	0.00	58.50
	10	110147.66[J]	0.00	299.47	0.00	302.90	0.00	254.23	0.00	254.92
	15	94496.406[J]	0.00	273.08	0.00	289.96	0.00	219.25	0.00	220.47
	20	85129.3[J]	0.23	323.46	0.23	438.41	0.23	258.42	0.23	272.45

*J. Kratica (2007)

8. Analysis

To evaluate the impact of path relinking on quality of each algorithm and running time, Wilcoxan signed rank test are tested in section 8.1. The gaps from SAPR and GAPR in two representations are shown in figure 3.6 - 3.7. The results from comparing percentage of gap and computational time are shown in table 3.7.

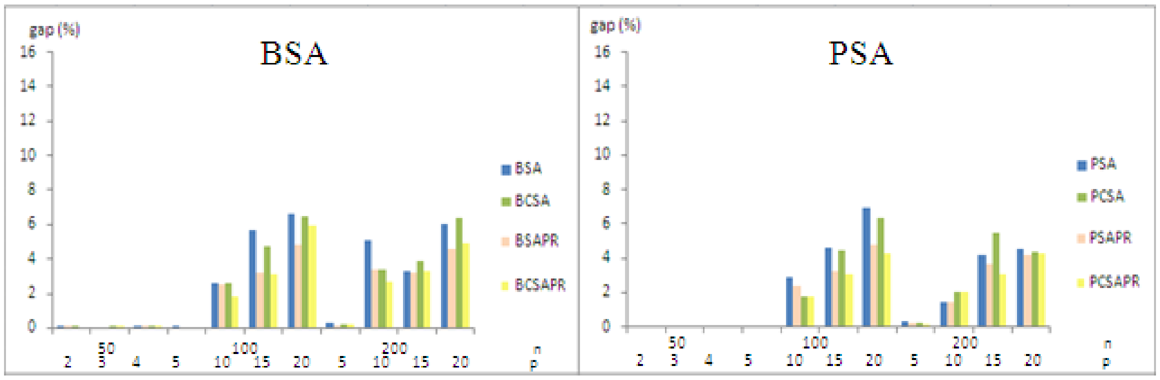


Figure 3.6 Gap of SAPR

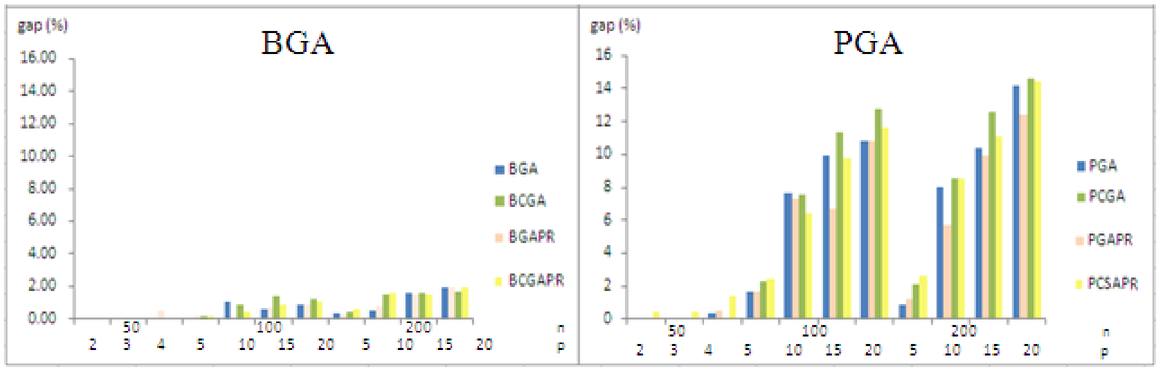


Figure 3.7 Gap of GAPR

8.1 Evaluating metaheuristics performance

To evaluate performance of each algorithm, the percentages of gaps obtained from GAPR and SAPR, in each representation with and without caching are analyzed by Wilcoxan signed rank test and are displayed as statistical results in table 3.7. The methodology is the same as in the previous chapter.

Table 3.7 Time and performance comparisons

Comparisons		N	N_{test}	Wilcoxon statistic	$W_{1-\alpha}$
GAPR time	BGAPR - BCGAPR	20	15	116	58.1*
	pGAPR - pCGAPR	20	16	134	61.9*
	pCGAPR - BCGAPR	20	16	60	61.9
GAPR gap	BCGAPR – BGAPR	20	8	26	23.56*
	pCGAPR - pGAPR	20	11	61	37.11*
	pGAPR – BGAPR	20	10	55	32.37*
SAPR time	BSAPR – BCSAPR	20	14	105	58.1*
	pSAPR – pCSAPR	20	13	69	47.22*
	BCSAPR – pCSAPR	20	13	37	47.22
SAPR gap	BSAPR -BCSAPR	20	9	22	27.85
	pSAPR – pCSAPR	20	7	20.5	23.56
	BCSAPR - pCSAPR	20	9	45	27.85*
TSPR time	BTSPR – BCTSPR	20	15	120	58.1*
	pTSPR – pCTSPR	20	14	98	52.56*
	BCTSPR – pCTSPR	20	14	82	52.56*

TS gaps are the same due to the same procedure.
*Significantly difference

8.2 Discussion

In the permutation GAPR analysis, there are statistically significant differences between the performance and running times of the binary GAPRs with and without caching: the binary GAPR with caching is faster (as shown in table 3.7) but the binary GAPR without caching performs faster.

In the binary GAPR analysis, we see that there is a statistically significant difference in performance between the permutation GAPRs with and without caching. The permutation GAPR without caching performs better, while the permutation caching GAPR performs faster. We prefer the permutation GAPR without caching.

When comparing the binary caching GAPR and the permutation GAPR without caching based on performance, we find the binary GAPR without caching performs better.

Based on table 3.7, we see that there is no a statistically significant difference in performance between the binary SAPRs with and without caching. Since there is a statistically significant difference in running time between the binary SAPRs with and without caching, with the binary caching SA being faster , we prefer the binary caching SAPR.

In the permutation SAPR analysis, we find the same results: there is not a statistically significant difference between the performance of the permutation SAPRs with and without caching while there is a statistically significant difference in the running times, with the permutation caching SAPR being faster. Therefore, we prefer the permutation caching SAPR.

When comparing the binary caching SAPR and the permutation caching SAPR based on performance, we find the permutation caching SAPR performs better. Therefore, we conclude that the recommended SAPR is the permutation caching SAPR.

In the TSPRs analysis, there is no difference among the performance of the TSPRs with and without caching. While optimal solutions are already found in all problems of the TSs except for the 200 nodes 20 hubs problem, the TSPRs do not find the best known solution.

However, there is a statistically significant difference in running time between the binary TSPRs with and without caching, with the binary caching TSPR being faster (as shown in Table 3.7), we prefer the binary caching TSPR.

In the permutation TSPR analysis, we find the same results: there is a statistically significant difference in the running times, with the permutation caching TSPR being faster. Therefore, we prefer the permutation caching TSPR.

When comparing the binary caching TSPR and the permutation caching TSPR based on running time, there is a statistically significant difference, with the permutation caching TSPR being faster. Therefore, we conclude that the recommended TSPR is the permutation caching TSPR.

Table 3.8 The recommended algorithms comparisons

Comparisons		N	N test	Wilcoxon statistic	$W_{1-\alpha}$
time	BGAPR - pCSAPR	20	15	115	58.1*
	pCSAPR - pCTSPR	20	16	125.5	61.9*
gap	pCSAPR -BGAPR	20	9	39	27.85*
	BGAPR - pCTSPR	20	9	45	27.85*

*Significantly difference

In table 3.8, we compare the recommended GAPR, SAPR and TSPR versions to each other. When comparing the binary GAPR without caching and the permutation caching SAPR based on performance, we find the binary GAPR caching performs better. There is a statistically significant difference between the running time of the binary GAPR without caching and the permutation caching SAPR, we find the permutation caching SAPR being faster.

Based on table 3.8, there are statistically significant differences between the running times of the permutation caching SAPR and the permutation caching TSPR, we prefer the permutation caching TSPR.

When comparing the binary GAPR without caching and the permutation caching TSPR based on performance, we find statistically significant difference in the performance between the binary GAPR without caching and the permutation caching TSPR. Therefore, we conclude that the recommended PR is the permutation caching TSPR.

9. Conclusion

In this research, Path Relinking is implemented with the four types of representations: permutation, permutation with caching, binary, and binary with caching. The computations are based on the AP data set. Since PR is based on a random approach, the percentages of gaps are not directly related to the initial solution construction algorithm. In addition, selecting good quality shortest network for reallocation can affect to computational time. However, computational times from PR based on caching algorithms are faster than algorithms without caching. In summary, using Path Relinking to hybridize heuristics can improve the quality of solutions around their construction algorithms. Random methods are not well suited for problem where lots of hub sets have to match up due to high interaction between elements.

CHAPTER FOUR

METAHEURISTICS FOR THE UMA*p*HMP

1. Introduction

In large transportation networks, many cities function as origins and destinations. For example, establishing and operating several airport hubs leads to more efficient performance, reduced travelling time and costs. In large airline networks many flights route from a city to different hub airports depending on passenger destination. In this case, the uncapacitated multiple allocation p-hub median location model is suitable to minimize total transportation. The problem of allocating non-hub node to hubs is NP-hard as is the *p*-hub median problem (Kyra, 1999). The multiple allocation *p*-hub median problem is concerned with locating hubs and allocating non-hub nodes to hub nodes. Non-hub nodes are allowed to be allocated to more than one hub depending on the destination of the flows originating at that node. We call non-hub nodes which are allocated to more than one hub the *multiple nodes*.

2. Metaheuristics

Since UMApHMP belongs to the class of NP-hard problems, at some point, exact optimization methods will have difficulty solving certain instances. The optimization software, AMPL, can solve the UMA*p*HMP instances based on the AP data set up to 30 nodes.

Ernst (1998) formulated an ILP for UMApHMP, resulting in the run times displayed in table 4.1 for the same computational environment. These problems require more memory than USApHMP. For the 20-node problem, UMApHMP begins to consume two to three times the amount of computational time of USApHMP as shown in table 4.1. Additionally, when the number of nodes is larger than 40, AMPL reported that too much memory was used.

Table 4.1 Result from mathematic programming for UMApHMP

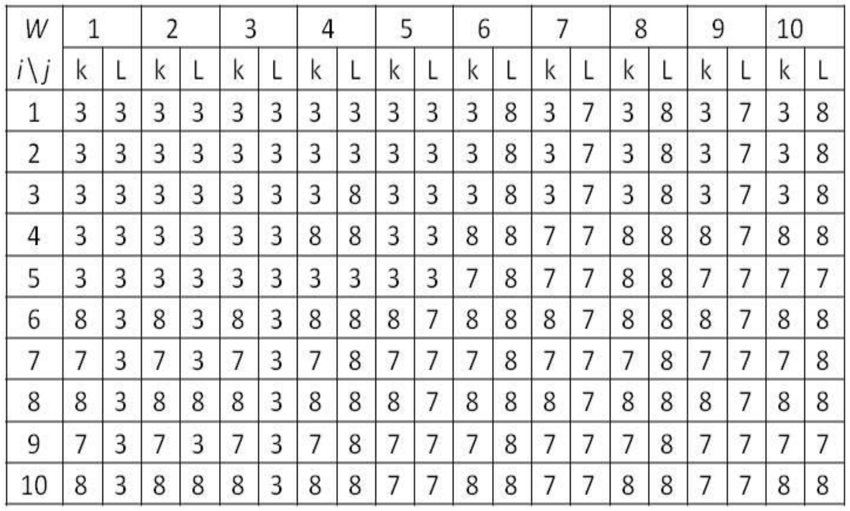
n	*p*	Obj	AMPL time(s)
10	2	163603.9436	0.1875
	4	107354.7300	0.3906
20	2	168599.7873	3.2967
	4	131665.4302	11.2800
30	2	170906.7072	9.1875
	4	138035.9339	41.2031
40	2	too much memory used	
	4	too much memory used	
50	2	too much memory used	

We attempt to solve the UMApHMP using the formulation by Ernst in 1998 implemented in CPLEX 11.2 with AMPL on an Intel Core Duo 1.66 GHz with 1 GB RAM. The run times are displayed in table 4.1. For 20 nodes problem, UMApHMP consumes 2-3 times of computational time of USApHMP. Additionally, when the number of nodes is more than 30, AMPL reported too much memory used. Therefore, a metaheuristic is proposed to solve the problem especially when size of problem is large up to 200 nodes. In chapter 2 and chapter 3, we have investigated the USA*p*HMP with metaheuristics and hybrid metaheuristics. Based on the computational experiment in section 7 of chapter 2, we found that the most effective algorithm for the USA*p*HMP is the permutation tabu search. Due to the complexity of the UMA*p*HMP, very little research about metaheuristics for the UMA*p*HMP is available . Therefore, in chapter 4 we will extend the permutation tabu search to solve the UMA*p*HMP.

2.1 Solution representation

A two dimensional array is used to represent the solution. The size of the array is equal to $n \times 2n$. The two dimensional array expresses the path of each O-D pair. The row represents the origin node and the column identifies the destination node. Recall that in the UMApHMP, flow originates at a node *i*, passes through hubs *k* and possibly *l*, and then arrives at its destination *j*. The intermediate hubs of for O-D pairs represented by the row and column combination *i* and *j* are represented in column *k* and *l* respectively.

For example, consider a problem with *n* = 10, *p* = 3 in Figure 4.1. The route of flows from origin node 2 to destination node 6 is 2-3-8-6. Therefore, 3 appears in column *k* of origin node 2 and 8 is in column *l* of destination node 6. If a flow only passes through one hub, that hub is recorded in both columns *k* and *l*, such as in the case of the flow from node 1 to 2; the path is 1-3-3-2, or 1-3-2.



W	1		2		3		4		5		6		7		8		9		10	
$i \backslash j$	k	L	k	L	k	L	k	L	k	L	k	L	k	L	k	L	k	L	k	L
1	3	3	3	3	3	3	3	3	3	3	3	8	3	7	3	8	3	7	3	8
2	3	3	3	3	3	3	3	3	3	3	3	8	3	7	3	8	3	7	3	8
3	3	3	3	3	3	3	3	8	3	3	3	8	3	7	3	8	3	7	3	8
4	3	3	3	3	3	3	8	8	3	3	8	8	7	7	8	8	8	7	8	8
5	3	3	3	3	3	3	3	3	3	3	7	8	7	7	8	8	7	7	7	7
6	8	3	8	3	8	3	8	8	8	7	8	8	8	7	8	8	8	7	8	8
7	7	3	7	3	7	3	7	8	7	7	7	8	7	7	7	8	7	7	7	8
8	8	3	8	8	8	3	8	8	8	7	8	8	8	7	8	8	8	7	8	8
9	7	3	7	3	7	3	7	8	7	7	7	8	7	7	7	8	7	7	7	7
10	8	3	8	8	8	3	8	8	7	7	8	8	7	7	8	8	7	7	8	8

Figure 4.1 Solution representation for the UMA*p*HMP

The tabu search procedure is shown in Figure 4.2.

```
**Procedure** Tabu Search for the UMA$p$HMP
    Single Location Exchange
    Generate multiple nodes
            Create Convex Hull
            Recognize the position of each hub in convex hull
            Create straight lines to divide zone of each hub
            Calculate distance from each node to its middle arc
            Set priorities of multiple nodes
    Single Node Exchange
            Allocate multiple nodes to the shortest origin hub
            Evaluate
    Choose the best set of multiple nodes
    Single Reallocate destination hubs
    Update tabu list
    Update priorities of multiple nodes

**End**
```

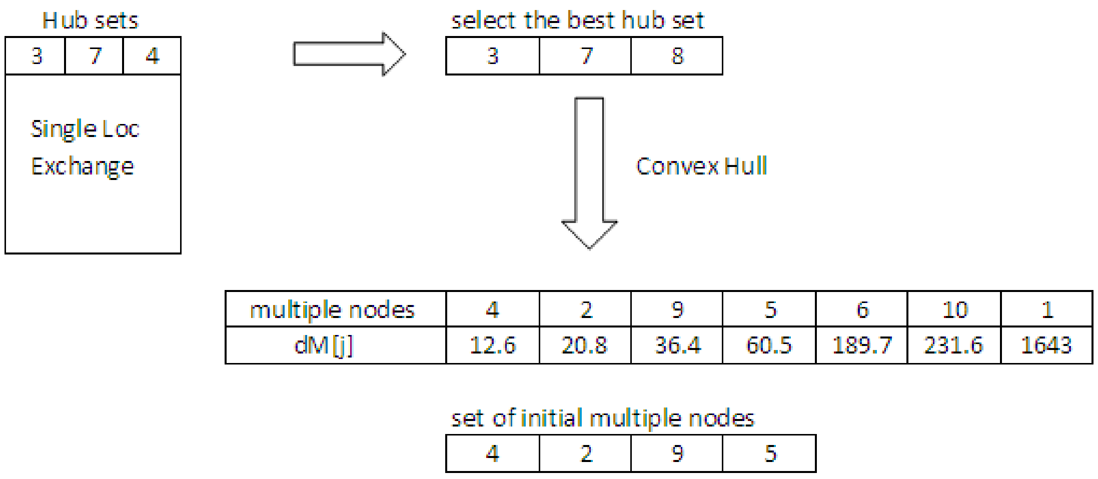
Figure 4.2 Tabu Search for the UMA*p*HMP procedure

The algorithm starts with receiving the best hubs from the tabu search for the USApHMP and generating the initial set of multiple nodes from convex hull. Then, it identifies the initial solution. Later, perform the Single Node Exchange procedure to find the best set of multiple nodes. Next, perform the shortest allocation for origin hub of multiple nodes (K_{mj}). Then, perform Single Reallocation for destination hubs of multiple nodes to determine the optimally multiple allocation network.

2.2 Generate initial multiple nodes

Multiple nodes are mostly aligned between two hubs in a network. Therefore, a convex hull is created to help sort the priorities of multiple nodes. Multiple nodes are selected from priorities of candidate multiple nodes or distance of each candidate multiple node in descending order. The distance of candidate multiple nodes is derived from the distance between the location of node and the middle line of its area in the convex hull. Let *m* be the number of multiple nodes that a problem may have. Number of multiple nodes, m, is defined by tuning. A numerical example for n = 10 p= 3, m = 4 is shown in figure 4.3.

Numerical Example for n = 10 p= 3, m = 4



Hub sets		
3	7	4

select the best hub set		
3	7	8

multiple nodes	4	2	9	5	6	10	1
dM[j]	12.6	20.8	36.4	60.5	189.7	231.6	1643

set of initial multiple nodes			
4	2	9	5

Figure 4.3 generating initial multiple nodes from convex hull

2.3 Convex Hull

The convex hull, the smallest enveloping polygon of *p* hubs, is implemented to determine nodes that align between hubs. In this research, the initial input of the convex hull is a set of *p* hubs. The procedure to generate multiple nodes from convex hull as displayed in Figure 4.4, is below.

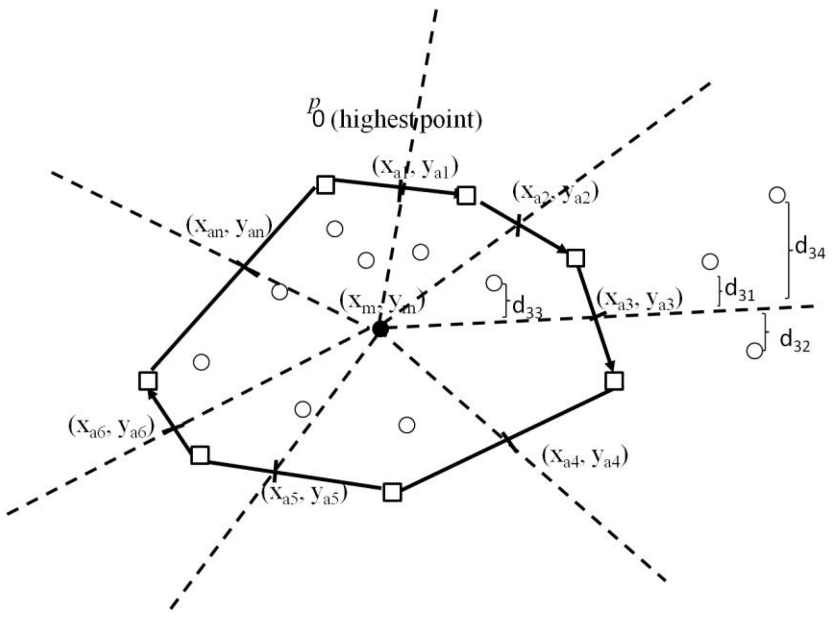


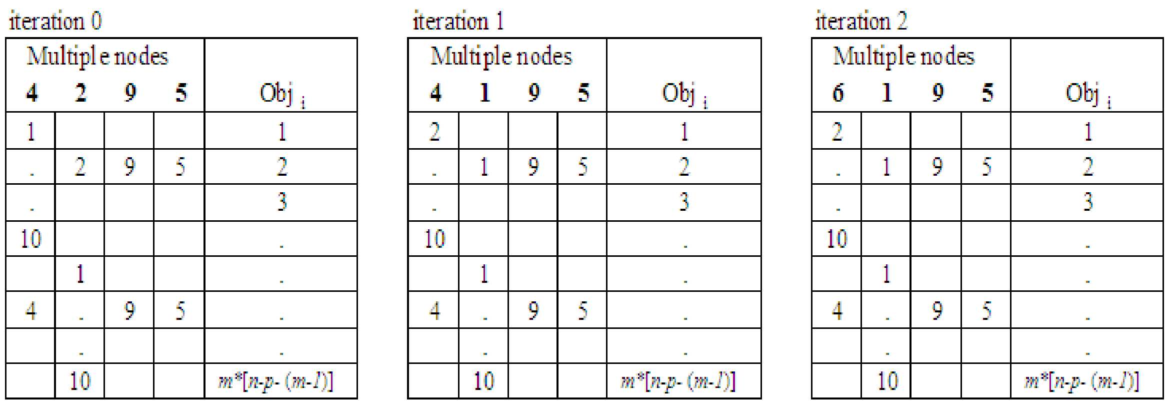
Figure 4.4 Convex hull

Firstly, create a convex hull for which some/all of *p* hubs are chosen to be boundary hubs. Use the position of the boundary hubs in the convex hull to create arcs from hub to hub in clockwise order. Next, create the straight lines from the middle point of the hull to the middle of arcs and calculate distance from node to its middle arc. Set priorities of multiple nodes (dM_j) corresponding to their distances in ascending order. Finally, choose multiple nodes to construct multiple allocation networks from their priorities. Note that it is possible that fewer than p hubs are used to create the convex hull.

2.4 Single Node Exchange procedure

To seek the best set of multiple nodes for the solution, the Single Node Exchange procedure is implemented by changing exactly one node in each solution. Therefore, the neighborhoods consist of $m \times [n - p - (m - 1)]$ solutions. For n = 10, p = 3 and m = 4, the neighborhood has 16 solutions. Then, the shortest allocation procedure is applied to find the best origin hubs of each O-D pair by fixing destination hub corresponding to the initial solution derived from the USApHMP. Next, evaluate and choose the best solutions to be the initial solution of the next iteration. These steps are repeatedly continued until no further improvement.

Table 4.2 Single Node Exchange procedure to seek the best set of multiple node



iteration 0

Multiple nodes				
4	**2**	**9**	**5**	Obj $_i$
1				1
.	2	9	5	2
.				3
10				.
	1			.
4	.	9	5	.
	.			.
	10			m*[n-p- (m-1)]

iteration 1

Multiple nodes				
4	**1**	**9**	**5**	Obj $_i$
2				1
.	1	9	5	2
.				3
10				.
	1			.
4	.	9	5	.
	.			.
	10			m*[n-p- (m-1)]

iteration 2

Multiple nodes				
6	**1**	**9**	**5**	Obj $_i$
2				1
.	1	9	5	2
.				3
10				.
	1			.
4	.	9	5	.
	.			.
	10			m*[n-p- (m-1)]

For Tabu list, the deleted node from each iteration will be recorded in the tabu list. In this numerical example, 2 and 4 are kept in the list for iteration 1 and 2, respectively.

2.5 Single Reallocation Hub for multiple nodes

Multiple allocation has flexibility that allows multiple nodes connect to more than one hub node for minimizing transportation cost. After a set of multiple nodes is obtained, Single Reallocation Hub for multiple nodes is applied to find a good allocation. To reallocate the destination hub over a list of potential hubs indicated by *l1* to *lp*, the transportation cost of flow from *i* to *j* is defined below.

$$\min(Cost_{i-k-l1-j}, Cost_{i-k-l2-j}, Cost_{i-k-l3-j}, ..., Cost_{i-k-lp-j})$$

where $\forall i \in n$, $\forall j \in m$.

We have applied Single Reallocation Hub for multiple nodes to keep the lowest transportation cost. There are p potential destination hubs for a path. Therefore, number of possible solutions is $p \times m \times n$.

2.6 Long term memory

In order to obtain diversified solutions, long term memory is applied to be a criterion to update a new starting solution. After multiple nodes are selected, the distances will be penalized by doubling their values, so that different sets of multiple nodes will be selected, as displayed in table 4.3.

Table 4.3 Numerical example: distance of selected multiple nodes.

iteration	selected Multiple nodes	Multiple nodes						
		4	2	9	5	6	10	1
0	4 2 9 5	12.6	20.8	36.4	60.5	190	232	1643
1	4 1 9 5	**25.2**	**41.6**	**72.8**	**121**	190	232	1643
2	6 1 9 5	**50.4**	41.6	**146**	**242**	190	232	**3286**
3	6 10 9 5	50.4	41.6	**291**	**484**	**379**	232	**6573**

For example, multiple nodes 4, 2, 9, and 5 are selected in iteration 0. Their distances are multiplied by two, as shown in the row for iteration 1. The same steps are repeated. For example, assume the long term memory length is 3. When the current iteration is equal to the setting long term memory, a new starting solution is generated by updating priorities of candidate multiple nodes as shown in table 4.4.

Table 4.4 Numerical example: updating priorities of candidate multiple nodes.

ltm = 3	Multiple nodes						
	2	4	10	9	6	5	1
dM_j	41.6	50.4	232	291	379	484	6573

Priorities of candidate multiple nodes are updated based on their distance in ascending order.

3. Computational environment

The algorithms are coded in C Programming language. The computation is performed in the high throughput computing pool of Clemson’s Palmetto cluster in Linux environment.

4. Stopping criteria

This procedure will stop when no new incumbent has been found for at least a fixed number of consecutive iterations. In addition, the TS will be stopped when the gap is less than a preset value.

$$\%Gap = \left(\frac{Incumbent}{BestKnown\,(or\ Optimal\ Sol)} - 1\right) * 100$$

The preset value is approximately 0.5% of the difference between the incumbent and the optimal or best known solution.

5. Computational Experiments

To evaluate the computational results of TS for the UMA*p*HMP, many sizes of the AP data set have been used. The data set contains 200 nodes, with smaller problems created by selecting subsets of those 200 nodes. For each problem size, the number of hubs is equal to 2, 3, 4, or 5 hubs. Nearly all algorithms obtain optimal solutions for small problems (10, 20, 40, and 50 nodes). Note that each problem is only solved once, since the proposed tabu search has no randomness. The gap and computational time of MultipleTS are displayed in figure 4.5 and 4.6.

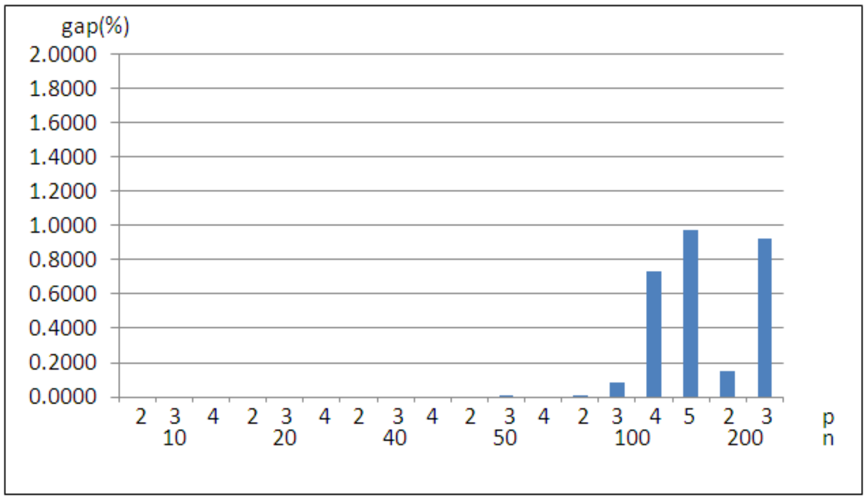


Figure 4.5 Gap of MultipleTS

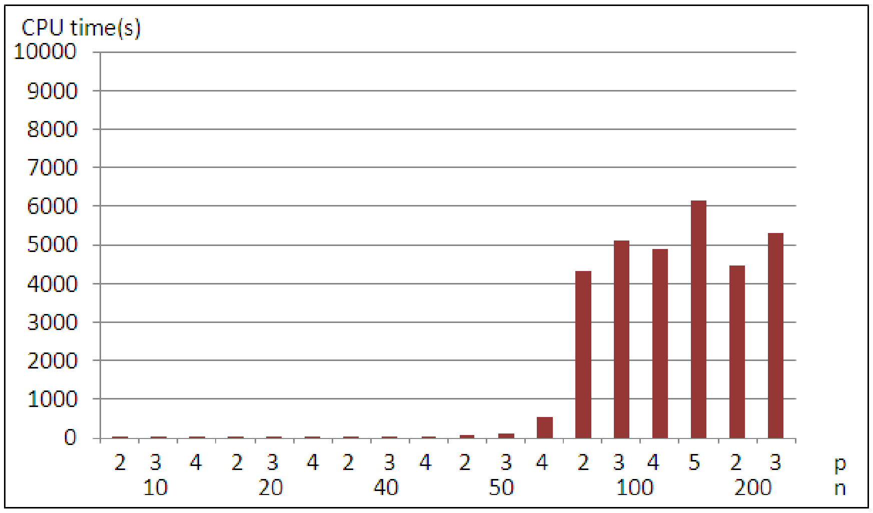


Figure 4.6 Computational time of MultipleTS

Table 4.5 Computational results

n	*p*	Optimal	Best found by TS	Gap (%)	Time (s)
10	2	163603.94	163603.94	0.0000	0.35
	3	131581.79	131581.79	0.0000	0.41
	4	107354.73	107354.73	0.0000	0.48
20	2	168599.79	168599.79	0.0000	0.34
	3	148048.30	148048.30	0.0000	0.54
	4	131665.43	131665.43	0.0000	3.34
40	2	173415.96	173415.96	0.0000	28.63
	3	155458.61	155458.61	0.0000	33.90
	4	140682.74	140682.74	0.0000	43.64
50	2	174390.03	174390.03	0.0000	86.44
	3	156014.72	156016.60	0.0001	120.30
	4	141153.38	141153.38	0.0000	526.25
100	2	176245.38	176268.02	0.0100	4303.02
	3	157869.93	158008.37	0.0870	5136.20
	4	143004.31	144056.79	0.7300	4886.83
	5	133482.57	134777.99	0.9700	6135.96
200	2	178094.99	178364.00	0.1500	4452.50
	3	159725.11	161207.20	0.9200	5322.46

For large problems, 100 and 200 nodes, TS finds nearly optimal solutions for every size of problems. The computational results of TS algorithm are shown in table 4.5. In comparison with the benchmark techniques Ant Colony Algorithm proposed by Kang, 2008 and the evolutionary based approach proposed by Marija, 2010, we find our tabu search performs better than Ant Colony Algorithm in quality of solution. Up to now, the most effective metaheuristic is the evolutionary based approach.

A few reasons that tabu search perform effectively is described now. First, Tabu search has strength in its deterministic process to avoid revisiting the same set of multiple nodes and long term memory to diversify solutions. Second, the heuristic methods are suitable for complex algorithm to focus on only necessary stages. It considers the multiple nodes based on the ascending distance from convex hull to find the optimal multiple allocation network.

6. Conclusion

In this chapter, we proposed the MultipleTS for the UMApHMP. The algorithm has been developed from the permutation tabu search for the USApHMP obtained from Chapter 2 by using the heuristic method to find optimal allocation part. The heuristic methods are defining multiple nodes from convex hull, Single Node Exchange to find best set of multiple nodes and Single Reallocate Exchange to find the best allocation of the network. The AP data set up to 200 nodes was used in the computational experiment. From the computational results, the MultipleTS is capable for solving small sized of problems and provide good solutions for large size of problems.

CHAPTER FIVE

CONCLUSION AND FUTURE WORK

5.1 Conclusion

This dissertation has addressed two variants of the *p*-hub median problem: the uncapacitated single allocation (USApHMP) and the uncapacitated multiple allocation (UMApHMP) versions. We first considered the USApHMP and then the UMApHMP, based on the results of the USApHMP.

In Chapter 2, we learned that USApHMP is NP-hard, and so developed three metaheuristics, namely GA, TS, SA, for the USApHMP for two types of solution representations. Each algorithm was implemented with caching and without caching. The results distinctly show that caching can speed up running times of all algorithms. The performance of TS with the permutation representation is highest among other algorithms. The deterministic procedures of TS such as single location exchange to find good hub sets and reallocation to find best allocation are applied to find optimal solution. The strategy of finding the best hub set first and allocating non-hub nodes to the correct hub afterwards appears to contribute to the performance. For small size instances (10, 20, 40, and 50 nodes), TS achieved optimal solutions in very short times. For large problem sizes (100-200 nodes), TS found nearly optimal solutions.

In Chapter 3, we investigate how path-relinking may impact the performance of various base metaheuristics. Path-relinking is implemented with GA, SA and TS algorithms as hybrid algorithms, GAPR, SAPR, and TSPR. Each type of hybrid algorithm consists of four versions: permutation, permutation with caching, binary, and binary with caching. The path relinking algorithm enhances the performance of almost all algorithms except TSPR A good base metaheuristic does not require PR.

In Chapter 4, we proposed Multiple TS for solving the UMApHMP based on the results for USApHMP in Chapters 2 and 3. The UMApHMP allow flexibility to allocate non-hub node more than one hub which results in lower transportation cost than the USApHMP. Given a set of hubs, we identify multiple nodes (those nodes that receive flows or send flows to multiple hubs) using the Convex Hull, Single Node Exchange and Single Reallocation Exchange procedures. For small problems (10, 20, 40, and 50 nodes), TS finds nearly all optimal solutions. TS finds nearly optimal solutions for every large problem (100 and 200 nodes). Using a USApHMP initial solution combine with the geometric interpretation of the problem can provide good results.

5.2 Future research

Future work can focus on either problem-specific extensions or methodological extension. The future work on the pHMP could investigate the capacitated versions, include fixed cost to open hubs and route, or the use of incomplete hub networks (in which the hubs are not complete connected). Additionally, larger real world situations could be investigated such as distribution network, optical fiber network, communication network, and relief centers. For example, in an evacuation situation, a network may connect up to 1000 homes. Thus, data sets and algorithms for extremely large number of nodes could be developed.

The conclusions drawn on the use of PR in hybridizing GA, SA and TS should be tested on other problem types as well. This can begin with the variants of the pHMP discussed above and continue with network and non-network problems. By investigating the performance of PR when combined with other base heuristics, we can learn about the strengths and weaknesses of the base heuristics, and thereby strengthen them.

APPENDICES

Appendix A

Red-black tree

The binary search tree is among the most well-known data structure for searching, deleting, and inserting . The binary search tree stores sorted data. The basic structure of the binary search tree consists of a parent node, a right child node, and a left child node. The data arrangement structure in the binary search tree is in ascending order from left to right. In other words, the data in the left side is always less than the data in the right side. To search a node in the tree, it is compared with the left and the right nodes in the tree and is traced down a path on the tree until the bottom of the tree is reached. Therefore, the performance of the binary search tree depends on its height. If the data structure is arranged randomly, inserting a new node into the tree tends to keep the tree balanced, which results in a fast operation time. In contrast, if the data structure is arranged linearly, accessing the data sequentially is the worst case, or *O(n)* time.

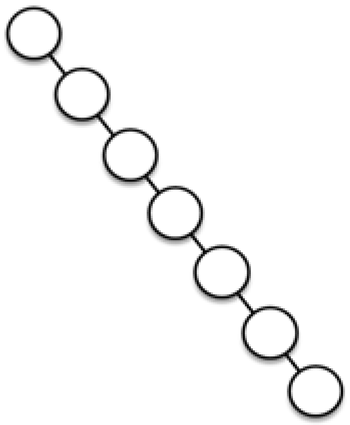


Figure 1 The worst case *O(n)* time

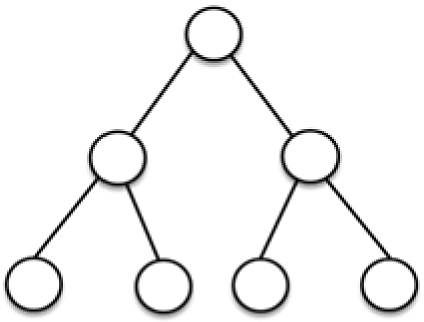


Figure 2 The best case *O(log(n))* time

A red-black tree is a self-balanced tree that keeps the binary search tree balanced. All operations in the red-black tree, such as search, insertion, and deletion, are guaranteed in *O(log(n))* time.

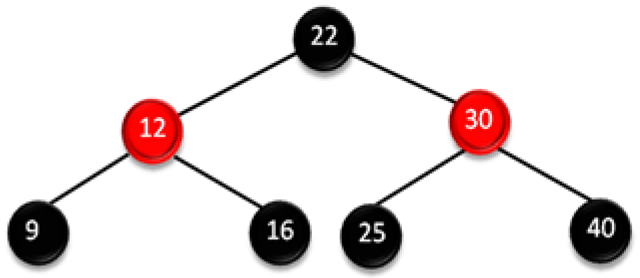


Figure 3 Red-black tree

To maintain a balanced binary search tree, these properties are necessary:

1. A node in the tree is either red or black.
2. The root is black
3. If a node is red, then its parent is black
4. Every path has the same number of black nodes

To insert a new node, it has to be red. When one of the characteristics of the red-black tree is violated, such as a double-red violation, the violated node is colored to be black. Or, if the value of a left node is more than that of a right node, either the left or right node rotates to maintain the characteristics of the red-black tree.

REFERENCES

Baha K. and Sibel A. (2008). Network hub location problems: the state of the art. *European Journal of Operational Research, 190*(1), 1-21.

Campbell, J. F. (1992). Location-allocation for distribution systems with transshipments and transportation economies of scale. *Annals of Operations Research, 40*, 77-99.

Campbell, J. F. (1994). Integer programming formulations of discrete hub location problems. *European Journal of Operational Research, 72*, 1-19.

Campbell, J. F. (1994). A survey of network hub location. *Studies in Locational Analysis, 6*, 31-49.

Campbell, J. F. (1996). Hub location and the p-hub median problem. *Operations Research, 44*(6), 1-13.

Campbell, J. F., Ernst AT and Krishnamoorthy M (2002). *Facility Location: Applications and Theory*. Berlin: Springer.

Campbell, J. F., Ernst, A.T and Krishnamoorthy M. . (2005). Hub arc location problems: part I—introduction and results. *Management Science 51*(10), 1540–1555.

Campbell, J. F., Ernst, A.T. and Krishnamoorthy, M. . (2005). Hub arc location problems: part II-formulations and optimal algorithms. *Management Science, 51*(10), 1556 - 1571.

Campbell, J. F. G. S., Andreas T. Ernst, and Mohan Krishnamoorthy. (2003). Solving hub arc location problems on a cluster of workstations *Parallel Computing, 29*(5), 555-574.

Ebery, J. (2001). Solving large single allocation p-hub problems with two or three hubs *European Journal of Operational Research, 128*(2), 447-458.

Filipović, V. (2003). Fine-grained tournament selection operator in genetic algorithms. *Computing and Informatics 22*, 143–161.

Hansen, P. and Mladenovic, N. (2001). Variable Neighborhood Search: Principles and Applications. *European Journal of Operations Research, 130*, 449–467.

Hatice C. K, A. S., Kara B.K., and Karasan O. (2009). A tabu-search based heuristic for the hub covering problem over incomplete hub networks. *Computers and OperationsResearch, 36*, 3088 - 3096.

Jeng.F., C. (2007). A hybrid heuristic for the uncapacitated single allocation hub location problem. *Omega, 35*, 211-222.

Klincewicz, J. G. (1991). Heuristics for the p-hub location problem. *European Journal of Operational Research, 53*, 25-37.

Klincewicz, J. G. (1992). Avoiding local optima in the p-hub location problem using tabu search and GRASP. *Annals of Operations Research, 40*, 283–302.

Krishnamoorthy, E. a. (1996). Efficient algorithms for the uncapacitated single allocation p-hub median problem. *Location Science, 4*(3), 139–154.

Krishnamoorthy, E. a. (1998a). Exact and heuristic algorithms for the uncapacitated multiple allocation p-hub median problem. *European Journal of Operational Research 104*, 100-112.

Krishnamoorthy, E. a. (1998b). An exact solution approach based on shortest-paths for p-hub median problems. *Informs Journal on Computing, 10*(2), 149–162.

Ma, K. T. (2008). *An Ant Colony Optimization Algorithm for Solving the Uncapacitated Multiple Allocation P-Hub Median Problem.* Paper presented at the The Ninth Asia-Pacific Industrial Engineering & Management Systems.

Milanović, M. (2010). A New Evolutionary Based Approach for Solving the Uncapacitated Multiple Allocation p-Hub Median Problem *Soft Computing in Industrial Applications 75* 81-88.

O'Kelly, M., Skorin-Kapov D. and Skorin-Kapov J. (1995). Lower bounds for the hub location problem. *Management Science 41,* 713-721.

O'Kelly, M. E., Bryan.D., Skorin-Kapov and Skorin-Kapov J. (1996). Hub network design with single and multiple allocation: A computational study *Location Science, 4*(3), 125-138.

O’Kelly, M. E. (1986). The location of interacting hub facilities. *Transportation Science, 20*(2), 92-105.

O’Kelly, M. E. (1987). A quadratic integer program for the location of interacting hub facilities. *European Journal of Operational Research, 32,* 393-404.

Osman, I. (1993). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operational Research 41*, 421–451.

Pérez, M. A. F. a. M. V. J. M. (1998). *Genetic algorithm with multistart search for the p-hub median problem.* Paper presented at the the EUROMICRO Conference – EUROMICRO'98.

Pérez, M. A. F. a. M. V. J. M. (2004). On the use of the path relinking for the p-hub median problem. *Lecture Notes in Computer Science*, 155-164.

Pérez, M. A. F. a. M. V. J. M. (2007). A hybrid VNS–path relinking for the p-hub median problem. *IMA J Management Math 18*(2), 157-171.

Sue., A.-H. (1998). A hybrid heuristic for the uncapacitated hub location problem. *European Journal of Operational Research 106*, 489-499.

Sue., A.-H. (2001). Using simulated annealing to solve the p-hub median problem. *International Journal of Physical Distribution & Logistics Management, 31*(3), 203-220.

Silva, C. a. M. R. (2007). A genetic algorithm for the problem of configuring a hub-and-spoke network for a LTL trucking company in Brazil. *European Journal of Operational Research, 179*, 747–758.

Skorin-Kapov, D., Skorin-Kapov, J. (1994). On tabu search for the location of interacting hub facilities. *European Journal of Operational Research, 73*, 502-509.

Skorin-Kapov, D., Skorin-Kapov, J., O'Kelly, M. (1996). Tight linear programming relaxations of uncapacitated p-hub median problems. *European Journal of Operational Research, 94*, 582–593.

Stanimirovic, Z. (2008). An efficient genetic algorithm for the uncapacitated Multiple Allocation p-hub Median Problem. *Control and Cybernetics, 37*(3), 669-692.

Topcuoglu, H., Corut, F., Ermis, M., Yilmaz, G. (2005). Solving the uncapacitated hub location problem using genetic algorithms. *Computers & OR, 32*(4), 967–984.

White, S. (1984). *Concepts of scale in simulated annealing.* Paper presented at the IEEE Int. Conference on Computer Design, Port Chester, NY.