DISTRIBUTION PLANNING WITH SELECTIVE TRAVELING SALESMAN AND VEHICLE ROUTING PROBLEMS

A Dissertation by

Elham Kookhahi

Master of Science, Isfahan University Technology, 2012

Bachelor of Science, Isfahan University Technology, 2009

Submitted to the Department of Industrial and Manufacturing Engineering
and the faculty of the Graduate School of
Wichita State University
in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

December 2016

DISTRIBUTION PLANNING WITH SELECTIVE TRAVELING SALESMAN AND VEHICLE ROUTING PROBLEMS

The following faculty members have examined the final copy of this dissertation for form and content, and recommend that it be accepted in partial fulfillment of the requirement for the degree of Doctor of Philosophy with a major in Industrial Engineering.

_____
Mehmet Bayram Yildirim, Committee Chair

_____
Krishna Krishnan, Committee Member

_____
Laila Cure, Committee Member

_____
Deepak Gupta, Committee Member

_____
Ramazan Asmatulu, Committee Member

Accepted for the College of Engineering

_____
Royce Bowden, Dean

Accepted for the Graduate School

_____
Dennis Livesay, Dean

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deep gratitude to my dissertation chair, Professor Mehmet Bayram Yildirim. He patiently guided me through my dissertation research, never accepting less than my best efforts. His wisdom, knowledge, and commitment to the highest standards have inspired and motivated me.

I acknowledge my committee members—Dr. Krishnan Krishna, Dr. Laila Cure, Dr. Deepak Gupta, and Dr. Ramazan Asmatulu—for agreeing to serve on my dissertation committee.

I also give my heartfelt appreciation to my parents, who have always supported me and encouraged me to study and get an advanced degree.

ABSTRACT

This dissertation introduces a new class of selective traveling salesman problem (TSP) and vehicle routing problem (VRP) in which the goal is to maximize the served demand. Using solution frameworks developed for these selective TSPs and VRPs, the decision-maker has the opportunity to implement efficient planning tools to schedule vehicle tours with respect to the budget and time constraints.

In this dissertation, mixed-integer linear mathematical models are developed and solved for each problem, where the goal is maximization of served demand. The problems considered in this dissertation are the following: traveling salesman problem with partial coverage (TSPWPC), capacitated vehicle routing problem with partial coverage (CVRPWPC), multiple traveling salesman problem with partial coverage (MTSPWPC), and multiple depot capacitated vehicle routing problem with partial coverage (MDCVRPWPC).

The proposed models were solved using exact solution methods and metaheuristic methods such as hybrid genetic algorithms. The proposed hybrid genetic algorithms (HGAs) consider the polar angle of the nodes and incorporate clustering algorithms to determine and improve the solutions, which can be obtained by regular genetic algorithms (GAs). In the proposed selective multiple vehicle mathematical models, we also utilize sweep and assignment methods to generate structured solutions.

This dissertation also presents detailed numerical experimentation, as well as computational results to assess the performance of the proposed algorithms.

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

Chapter Page

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

Chapter Page

LIST OF TABLES

LIST OF TABLES (continued)

LIST OF FIGURES

LIST OF FIGURES (continued)

LIST OF FIGURES (continued)

LIST OF FIGURES (continued)

Figure Page

LIST OF ABBREVIATIONS

CGA	Cellular Genetic Algorithm
CPU	Central Processing Unit
CSP	Covering Salesman Problem
CVRP	Capacitated Vehicle Routing Problem
CVRPWPC	Capacitated Vehicle Routing Problem with Partial Coverage
DFJ	Dantzig-Fulkerson-Johnson
GA-2OPT	Genetic Algorithm Based on Greedy 2-OPT Mutation
GA	Genetic Algorithm
GA-AS	Genetic Algorithm Based on Assignment Method
GA-Cluster	Genetic Algorithm Based on Clustering Method
GA-IR	Genetic Algorithm Based on In-Route Mutation
GA-MDAS	Genetic Algorithm Based on Multiple Depot Assignment Method
GAMS	General Algebraic Modeling System
GA-MDSW	Genetic Algorithm Based on Multiple Depot Sweep Method
GA-Polar	Genetic Algorithm Based on Polar Angle Method
GA-SW	Genetic Algorithm Based on Sweep Method
HGA	Hybrid Genetic Algorithm
HVRP	Heterogeneous Vehicle Routing Problem
ILP	Integer Linear Program
LIFO	Last-In First-Out
MDVRP	Multiple Depot Vehicle Routing Problem

LIST OF ABBREVIATIONS (continued)

MDCVRPWPC	Multiple Depot Capacitated Vehicle Routing Problem with Partial Coverage
MILP	Mixed-Integer Linear Program
MIP	Mixed-Integer Program
MPSRPTW	Multiple Depot Petrol Station Replenishment Problem with Time Windows
MTSP	Multiple Traveling Salesman Problem
MTZ	Miller-Tucker-Zemlin
MTSPWPC	Multiple Traveling Salesman Problem with Partial Coverage
MVCTP	Multiple Vehicle Covering Tour Problem
NN	Neural Network
NNS	Nearest Neighborhood Search
NP-Hard	Non-Deterministic Polynomial-Time Hard (No Known Algorithm Can Solve This Problem in Polynomial Time)
NP-Complete	Non-Deterministic Polynomial-Time Complete
PMX	Partially Mapped Single Point Crossover
PSO	Particle Swarm Optimization
PVRP	Periodic Vehicle Routing Problem
RE	Relative Error
SA	Simulated Annealing
SPP	Set Partitioning Problem
SS	Sum of Squares
SVRAP	Single Vehicle Routing Allocation Problem

SVRP	Stochastic Vehicle Routing Problem
TS	Tabu Search
TSP	Traveling Salesman Problem
TSPWPC	Traveling Salesman Problem with Partial Coverage
UTSA	Unified Tabu Search Algorithm
VRP	Vehicle Routing Problem
VRPPD	Vehicle Routing Problem with Pickup and Delivery
VRPTW	Vehicle Routing Problem with Time Windows

LIST OF NOTATIONS

B	Budget Amount
B_k	Budget Assigned to Vehicle *k*
c_{ij}	Cost of Traveling between Vertex *i* and Vertex *j*
c_{ij}^k	Travel Cost between Node *i* and Node *j* by Vehicle *k*
c_p^1	First Parent Chromosome
c_p^2	Second Parent Chromosome
c_{off}^1	First Offspring Chromosome
c_{off}^2	Second Offspring Chromosome
cap_k	Capacity of Vertex *k*
CAP_i	Capacity of Node *i*
$capv_k$	Capacity of Vehicle *k*
d_k	Demand of Vertex *k*
DIS_j	Maximum Distance People Will Travel to Come to City *j*
E	Edge Set
E_j	Number of People Who Travel to City *j*
FC	Fixed Cost of Vehicle Assignment
fc_k	Fixed Cost of Assigning Vehicle *k*
G	Undirected Graph
G_r	Fleet of Vehicles in Depot *r*
h_i	Binary Variable Showing If Node *i* Is on Any Route
I	Set of Nodes

i	Node *i*
J	Set of Nodes
j	Node *j*
K	Set of Vehicles
K_r	Set of Vehicles at Depot *r*
k	Vehicle Index
L_{ij}	Distance between Node *i* and Node *j*
$lengthq$	Length of Tour
N_{max}	Number of Computational Cycles
obj	Maximum Number of Served Customers on Current Tour
p_k	Profit of Vertex *k*
p_s	Selection Probability
p_m	Mutation Probability
p_e	Elitism Probability
q_{ij}	Binary Variable Showing If Node *j* Is Assigned to Node *i*
R	Set of Depots
r	Depot Index
r_k	Binary Variable Showing If Vehicle *k* Is on Route
sp	Size of Population
T_{tour}	Tour Time Limit
T_{tour}^k	Maximum Time That Vehicle *k* Can Be on Route
t_{ij}	Travel Time between Vertex *i* and Vertex *j*

t_{ij}^{k}	Time of Traveling Between Node *i* and Node *j* by Vehicle *k*
tp_i	Service Time at Node *i*
tp_i^k	Time of Getting Service at Node *j* by Vehicle *k*
$u_{\ i}$	Auxiliary Variable
u_{ik}	Auxiliary Variable
V	Set of Nodes Served Directly
V^K	Set of Nodes Served Directly by Vehicle *k*
V_i	Binary Variable Showing If Node *i* Belongs to Set *V*
v_i	Vertex *i*
v_i^k	Vertex *i* Served by Vehicle *k*
W	Set of Nodes Served Indirectly
W^k	Set of Nodes Served Indirectly
W_i	Binary Variable Showing If Node *i* Belongs to Set *W*
w_i^k	Binary Variable Showing If Node *i* Is Served Indirectly by Vehicle *k*
X_{ij}	Binary Variable Showing If There Is a Vertex between Node *i* and Node *j*
x_{ij}^k	Binary Variable Showing If There Is a Vertex between Node *i* and Node *j* by Vehicle *k*
Y_{ij}	Binary Variable Showing If Node *j* Is Assigned to Node *i*
y_{ij}^k	Binary Variable Showing If Node *j* Is Assigned to Node *i* and Get Served by Vehicle *k*
Z	Set of Nodes Not Being Served

Z_i	Binary Variable Showing If Node *i* Belongs to Set *Z*
$\bar{Z}_i$	Parameter Shows If Node *i* Is on Route
z_i^k	Binary Variable Showing If Node *i* Is Served Directly by Vehicle *k*
ρ	K-Means Threshold

CHAPTER 1

INTRODUCTION

1.1 Overview

The traveling salesman problem (TSP) is a well-studied combinatorial optimization problem with applications ranging from routing to scheduling. It can be classified based on several factors. For example, the TSP can be classified based on the objective function: minimizing the tour cost, maximizing the profit, or a combination of both; or it can be classified based on vehicle capacity, etc.

The TSP is non-deterministic polynomial-time complete (NP-complete) [1]. Exact algorithms, such as integer programming methods [2], can find the solution to this problem in a reasonable amount of time when its size is small [3]. However, when the problem size increases, exact algorithms may take a significant amount of time to solve to optimality. In this case, heuristic methods such as intelligent algorithms and metaheuristics can find good solutions in a reasonable amount of time. Theoretically, these algorithms can find the optimal solution if they are run for a long enough period of time. Some examples of these algorithms are the nearest neighborhood search (NNS) [4], genetic algorithm (GA) [5], simulated annealing (SA) [6], particle swarm optimization (PSO) [7], neural network (NN) [8], Tabu search (TS) [9], and ant colony [10].

This dissertation introduces a new class of traveling salesman problem that does not require visiting all potential nodes because of budget and time constraints. In addition, each node has a service capacity. A node that is visited by a vehicle may serve other surrounding nodes that are not directly visited, if the distance between nodes is not greater than a predetermined distance (i.e., customers are willing to travel within this distance). The objective of the proposed TSP is maximizing the total served demand. This new TSP with side constraints is also extended to

vehicle routing problems (VRPs) (i.e., many capacitated vehicles with a single depot, many capacitated vehicles with many depots, and many vehicles with many depots, etc.).

One real-life application of this problem can be in planning a tour for doctors in an underdeveloped region or at a time of crisis to serve the maximum number of patients. In addition, in health economics, registrars visit different hospitals to collect patient documents while maximizing the total number of served hospitals with a limited budget [11]. In disaster-management problems, an objective can be to maximize the number of served injured people by trained medical staff with limited time and budget constraints who must travel to different locations affected by the disaster, assuming that injured people could be transferred to locations on the route [12].

1.2 Objective of Dissertation

The main objective of this dissertation is to introduce a new class of selective TSP and VRP in which the goal is to maximize the served demand. The work here proposes mathematical models for these problems and provides customized solution algorithms, including metaheuristics, to solve them. In addition, the impact of several factors (e.g., how nodes are dispersed in the plane, impact of clusters, time limit, and budget, etc.) will be assessed. This dissertation will provide decision-makers or tour planners with efficient planning tools to schedule their tours with respect to budget and time constraints.

1.3 Contributions of Dissertation

The contributions of this dissertation, shown in Table 1.1, are summarized in in two main areas: modeling and algorithmic contributions. The abbreviations in Table 1.1 are defined as follows:

- **CPU**: Central Processing Unit

- **CVRPWPC**: Capacitated Vehicle Routing Problem with Partial Coverage
- **GA**: Genetic Algorithm
- **GA-2OPT**: Genetic Algorithm Based on Greedy 2-OPT Mutation
- **GA-AS**: Genetic Algorithm Based on Assignment Method
- **GA-Cluster**: Genetic Algorithm Based on Clustering Method
- **GA-IR**: Genetic Algorithm Based on In-Route Mutation
- **GA-MDAS**: Genetic Algorithm Based on Multiple Depot Assignment Method
- **GA-MDSW**: Genetic Algorithm Based on Multiple Depot Sweep Method
- **GA-Polar**: Genetic Algorithm Based on Polar Angle Method
- **GA-SW**: Genetic Algorithm Based on Sweep Method
- **MDCVRPWPC**: Multiple Depot Capacitated Vehicle Routing Problem with Partial Coverage
- **MILP**: Mixed-Integer Linear Programming
- **MTSPWPC**: Multiple Traveling Salesman Problem with Partial Coverage
- **RE**: Relative Error
- **TSPWPC**: Traveling Salesman Problem with Partial Coverage

Table 1.1. Dissertation Contributions

Problems	Objective	Mathematical Model	Solution Algorithm	Performance Measurement
TSPWPC	Total Served Demand	MILP	GA-Polar GA-Cluster	RE CPU
CVRPWPC	Total Served Demand	MILP	GA-SW GA-AS	RE CPU
MTSPWPC	Total Served Demand	MILP	GA-2OPT GA-IR	RE CPU
MDCVRPWPC	Total Served Demand	MILP	GA-MDSW GA-MDAS	RE CPU

1.3.1 Modeling Contributions

The main contribution of this dissertation is the introduction of a new class of traveling salesman problem, which is a combination of prize collecting and the covering salesman problem (CSP): the prize that can be collected from each node depends not only on nodes that are on the travel route but also nodes that are within a travel distance of those nodes on the route.

Our work is inspired from different selective traveling salesman problems existing in the literature. One of the problems is covering salesman problem (CSP) with the objective of minimization of the cost of the tour, in which all cities should lie within the covering distance of a city on the salesman route. The team orienting problem has an objective of defining a subset of customers to maximize the profit while time spent on visiting nodes is limited. Median cycle problem determines a cycle of vertices with the goal of minimizing the routing cost and the cost of assisting vertices which are not in the cycle. In all of these problems, the nodes which are not on the route will get served as long as they lie within the covering distance (i.e. covering salesman problem and median cycle problem) or they will not get served at all (i.e. team orienting problem). In the problems that we introduce in this dissertation, the nodes which are not on the route can be partially served by the nodes on the route as long as the capacity of the nodes on the route are big enough and the distance between these cities is less than a predetermined distance. The cost of assigning those nodes to the routes is assumed to be zero and each city can get service from only one node. Therefore, the prize that can be collected from visiting a node is dependent on the nodes that are visited previously. As those nodes might serve some nodes which are not on the route and those served nodes cannot be served by any other node on the route.

The problems addressed in this dissertation are variants based on the number of vehicles, the capacity of vehicles, the number of depots and the assignment cost of a salesman to a route. In

the first problem, there is only one vehicle to serve the demand. In the second problem, there is a fleet of vehicles with a certain capacity to serve a given set of nodes from a single depot. The third mathematical model considers that one decision is the number of vehicles in the fleet to serve the demand. Finally, the last mathematical model considers a multiple depot selective vehicle routing problem in which there is a fleet of vehicles in each depot to serve the demand of all customers. The number of vehicles are fixed at each depot, and they must return to the same depot they are departing from.

1.3.2 Algorithmic Contributions

TSPs and VRPs are very difficult problems (i.e., NP-complete). As a result, when the problem size increases, the time to solve these problems increases exponentially. The problems modeled in this dissertation are generalizations of the TSP and VRP problems. Thus, metaheuristic methods are proposed to solve these problems efficiently. We hope to obtain near-optimal solutions in a reasonable amount of central processing unit (CPU) time using the proposed metaheuristics.

We propose hybrid genetic algorithms (HGAs) to solve each problem. These HGAs consider the polar angle of the nodes and incorporate clustering algorithms to determine good solutions compared to regular GAs. In VRP-inspired models, we also utilize sweep and assignment methods to generate structured solutions.

To evaluate the efficiency of the proposed metaheuristics, medium-size problems are solved using either the CPLEX solver or a proposed dynamic cut algorithm. We analyze the performance of each metaheuristic based on several factors that are incorporated into the problem. Thereafter, the relative error is calculated for each GA.

1.4 Outline of Dissertation

The organization of this dissertation is as follows: Chapter 2 provides a review of the literature. Chapter 3, the first paper, introduces the traveling salesman problem with partial coverage. This problem can be considered a combination of the prize collecting problem and the CSP. Visiting each node results in some profit in terms of served demand at each node. Each node has the capacity to serve other nodes; therefore, other nodes can be assigned to nodes on the route, as long as the distance between the serving node and served node is not greater than a predetermined distance. Since there is a limited budget and tour duration, all cities may not be visited. The goal is to find the tour with maximum profit.

A mixed-integer linear program (MILP) is developed, and a dynamic cut algorithm is proposed to solve larger-size problems to optimality. Since the problem is non-deterministic polynomial-time hard (NP-hard), an HGA with second-level optimization is proposed, where the second level of optimization takes advantage of clustering algorithms. Furthermore, another genetic algorithm with insertion mutation based on the polar angles of nodes is designed.

Results of both GAs and the dynamic cut algorithm are compared based on the optimality gap. In addition, the impact of clustering algorithms and clusters on the efficiency of the proposed mathematical model and algorithms is analyzed in detail.

In Chapter 4, the second paper, we introduce a new vehicle routing problem called the capacitated vehicle routing problem with partial coverage (CVRPWPC). This problem is an extension of the traveling salesman problem with partial coverage (TSPWPC), in which there is a fleet of vehicles originating from and returning to a depot, as well as a limited budget, time limit, and capacity for each vehicle. The profit definition at each node is the same as the profit definition for TSPWPC. The objective is to find a set of routes with maximal served demand. To solve the

CVRPWPC, a genetic algorithm in which some structured solutions are generated using some modifications of sweep and assignment methods is proposed. A crossover operation is proposed resulting in a better order of nodes. Here the insertion mutation is implemented based on the polar angles of nodes.

In Chapter 5, the third paper, a multiple traveling salesman problem with partial coverage (MTSPWPC) is introduced. Here, there is a fixed cost of using a vehicle, i.e., the number of vehicles to serve the demand is not fixed. The objective here is similar to that of the TSPWPC and CVRPWPC. The goal is to maximize the served demand with a fleet of vehicles while having a limited time limit and budget for each vehicle, as well as a limited overall budget. To solve the MTSPWPC, two genetic algorithms are designed: a genetic algorithm based on in-route mutation (GA-IR) and a genetic algorithm based on the greedy 2-OPT mutation (GA-2OPT). In the GA-IR, two mutations, "in-route" and "cross-route," are performed. In GA-2OPT, "in-route" and "greedy 2-OPT" mutations are implemented.

In Chapter 6, the fourth paper, a multiple depot capacitated vehicle routing problem with partial coverage (MDCVRPWPC) is introduced. Here, a fleet of vehicles is assigned to each depot, and there is a limited budget and time limit for each vehicle. The goal here is to maximize the overall served demand, i.e., the objective is similar to the TSPWPC and CVRPWPC. To solve the MDCVRPWPC, two genetic algorithms are designed: genetic algorithm based on multiple depot assignment method (GA-MDAS) and genetic algorithm based on multiple depot sweep method (GA-MDSW). In GA-MDAS, the initial solutions are constructed cone by cone. In GA-MDSW, the initial solutions are constructed based on the sweep method at each depot. Before implementing the GAs, a clustering method is implemented to separate nodes based on the number of depots. An

inter-depot mutation is implemented to swap some nodes among depots at each generation in order to reduce the effect of the clustering method on the final solution.

1.5 Summary

This chapter presents the motivation for the dissertation work, and the problems addressed are defined. Next, the main algorithmic contributions of this dissertation work are discussed. Finally, an outline of the dissertation is presented.

1.6 References

[1] Zhang, W. X., & Korf, R. E. (1996). A study of complexity transitions on the asymmetric traveling salesman problem. *Artificial Intelligence, 81*(1–2), 223–239.

[2] Climer, S., & Zhang, W. X. (2006). Cut-and-solve: An iterative search strategy for combinatorial optimization problems. *Artificial Intelligence, 170*(8–9), 714–738

[3] Johnson, D. S., & McGeoch, L. A. (2004). *The traveling salesman problem and its variations, combinatorial optimization*. London: Springer Press, pp. 445–487.

[4] Burke, E. K., Cowling, P. I., & Keuthen, R. (2001). Effective local and guided variable neighbourhood search methods for the asymmetric travelling salesman problem. In *Applications of evolutionary computing* (pp. 203-212). Berlin Heidelberg: Springer.

[5] Chatterjee, S., Carrera, C., & Lynch, L. A. (1996). Genetic algorithms and traveling salesman problems. *European Journal of Operational Research, 93*(3), 490-510.

[6] Munakata, T., & Nakamura, Y. (2001). Temperature control for simulated annealing, *Physical Review E, 64*, 046127.

[7] Clerc, M. (2004). Discrete particle swarm optimization, illustrated by the traveling salesman problem. In *New optimization techniques in engineering* (pp. 219-239). Berlin Heidelberg: Springer.

[8] Budinich, M. (1996). A self-organizing neural network for the traveling salesman problem that is competitive with simulated annealing. *Neural Computation, 8*(2), 416-424.

[9] Knox, J. (1994). Tabu search performance on the symmetric traveling salesman problem. *Computers & Operations Research, 21*(8), 867-876.

[10] Dorigo, M., & Gambardella, L. M. (1997). Ant colonies for the travelling salesman problem. *BioSystems, 43*(2), 73-81.

[11] Lupsa, L., Chiorean, I., Lupsak, R., & Neamtiu, L. (2010). Some special traveling salesman problems with applications in health economics. In D. Davendra (Ed.), *Traveling salesman problem, theory and applications*. InTech Open Access Publisher. DOI: 10.5772/13365.

[12] Arulselvan, A. (2009). Network model for disaster management (Doctoral dissertation). University of Florida, Gainesville.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

The traveling salesman problem and vehicle routing problem have both been well studied. Several extensions of each problem can be modeled using different objectives and/or additional constraints.

In a regular TSP, the objective is to find a tour that covers all cities while minimizing the total cost of traveling. A variation of the TSP that has a profit associated with each node can be seen as a bi-criteria problem, where the objective is to maximize the collected profit from visiting each node while minimizing the cost of traveling. The solutions include a set of non-inferior feasible solutions, which none of the objectives can improve except by deteriorating the other objective.

According to Bektas [1], the TSP can be different based on the following:

- Number of travelers: If there is more than one traveler who can serve the nodes, then more than one tour can exist in the problem. This problem is called a multiple traveling salesman problem (MTSP). Some variations of this problem are where the number of travelers is not fixed. In the case of a varying number of travelers, there is a cost associated with each route (i.e., to each traveler). If there is a capacity associated with each traveler, then the problem is called a vehicle routing problem.
- Number of depots: If there is more than one depot having a fleet of vehicles, then the problem is called a multiple depot vehicle routing problem (MDVRP). In one variation of the problem, vehicles departing from a depot must return to the same depot (i.e., this is a

fixed-destination problem). In a different variation, vehicles departing from a depot can return to any other depot (i.e., a non-fixed-destination problem).

- Time windows: The TSP with time windows has a time window associated with each node, which requires that each node be served during its corresponding time interval.

The vehicle routing problem is a generalization of the TSP in which there is more than one traveler, and each traveler has a capacity limitation. The VRP can be categorized into more specific groups as follows [2]:

- Heterogeneous vehicle routing problem (HVRP): In this problem, there is a fleet of different types of vehicles, whereby each one must start its trip from the depot and must terminate its trip at the depot. Each node can be visited only once by only one vehicle, and the objective is to minimize the total cost.[3]
- Capacitated vehicle routing problem (CVRP): This problem is a VRP in which there is a fixed fleet of vehicles with a uniform capacity of a single commodity [4].
- Multiple depot vehicle routing problem: This problem is an extension of the VRP, whereby there is more than one depot and also a fleet of vehicles at each depot starting their trip from the corresponding depot. If the vehicle returns to the same depot, then the problem is referred to as a fixed-destination MDVRP; otherwise, it is referred to as a non-fixed-destination MDVRP.[5]
- Vehicle routing problem with time windows (VRPTW): In this problem, there is a time window for each node in which a visit can occur. Each vehicle must visit each node within the time frame, and the service cannot begin before opening the time window or after closing the time window [6].

- Stochastic vehicle routing problem (SVRP): This problem is an extension of the VRP in which there can be one or more parameters considered to be stochastic or unknown. The random parameters can be the nature of the customer demand, travel time, or time window for each customer [7].
- Vehicle routing problem with pickup and delivery (VRPPD): In this problem, each vehicle can pick up some items at each node as well as deliver items at that node. This problem is more complicated because the capacity utilization of vehicles can change over a trip [8].

The TSP and VRP are NP-hard problems [9, 10], and any generalization of these problems is NP-hard as well. Several algorithms can be used to solve the TSP and VRP, including exact, approximation, heuristic, and metaheuristic.

This chapter discusses some of the exact algorithms as well as metaheuristics used to solve new classes of the TSP, MTSP/VRP, and MDVRP that are addressed in the chapters that follow.

2.2 Exact Algorithms to Solve TSP and VRP

This section addresses some of the different exact methods used to solve the TSP, MTSP/VRP, and MDVRP. The scope of algorithms found in the literature is quite large. Some of most popular ones used to solve these problems successfully are mentioned here.

2.2.1 Exact Algorithms to Solve TSP

There are different exact algorithms to solve the TSP. Laporte [11] categorizes exact algorithms to solve the TSP into different groups in the context of an integer linear program (ILP). According to Laporte, there are two earliest formulations of TSP. Dantzig, Fulkerson, and Johnson (DFJ) [12] present the first formulation of TSP as an ILP. In this formulation, they introduce DFJ subtour elimination constraints. Later, to solve the TSP, Miller, Tucker, and Zemlin (MTZ) [13] introduce some new ILPs that reduce the number of subtour elimination constraints by using MTZ

subtour elimination constraints. One of the most common algorithms to solve TSP is the branch-and-bound algorithm. The quality of this algorithm depends on the quality of the boundary that results from the problem relaxation. Some authors have developed a branch-and-bound algorithm for the TSP based on relaxation of the subtour elimination constraints [14, 15].

Other algorithm extensions are based on the shortest spanning arborescence bound [16, 17], shortest spanning tree bound [18, 19], and the two-matching lower bound [20, 21].

Another method to solve the TSP is a cutting-plane technique [12]. A combination of this method and the branch-and-bound method can result in a successful solution approach and is referred to as the branch-and-cut algorithm. In this algorithm, the crucial key is to identify proper valid inequalities and have a good algorithm to separate them efficiently. [22]

Proper valid inequalities can be facet-defining inequalities such as subtour elimination constraints [23], comb inequalities [24], clique tree inequalities [25], path inequalities [26], and crown inequalities [27].

Different branch-and-cut algorithms use a variety of facet-defining valid inequalities and different methods to implement the TSP to optimality. Here we refer to Crowder and Padberg [28] who solve a 318-city instance; Padberg and Rinaldi [29] who solve instances having 532 cities; Grötschel and Holland [30] who solve an instance with 1,000 cities; and Applegate, Bixby, Chvátal and Cook [31] who solve instances with 7,397 and 13,509 cities. One of the largest TSP instances involving 24,978 cities is solved by using Concorde, a computer code for the TSP, developed by Applegate, Bixby, Chvátal, and Cook [32]

2.2.2 Exact Algorithms to Solve MTSP/VRP

The MTSP is a relaxation of the VRP, and the two have a very close connection. Hence, most algorithms that can solve the MTSP can solve the VRP, and vice-versa. Different exact

algorithms can be used to solve the MTSP. The Laporte and Nobert [33] algorithm is the first algorithm that solves the MTSP without transforming it into a TSP. This algorithm is performed based on relaxation of some subtour elimination constraints. The algorithm is designed in such a way that a subtour elimination constraint is applied to the problem if it is violated after achieving an integer solution. This approach is called a straight algorithm. Another approach, called the reverse algorithm, is where the existence of illegal subtours are checked before achieving an integer solution, which has a better performance than the straight algorithm.

Ali and Kennington [34] propose another exact algorithm, a branch-and-bound-based algorithm. A Lagrangian relaxation of the degree constraints is used in this algorithm and a subgradient algorithm is used to solve the dual.

Gavish and Srikanth [35] propose a branch-and-bound algorithm to solve the MTSP to optimality. Lower bounds are obtained by solving a Lagrangian degree constraint relaxation. Results show that the integer gap obtained by Lagrangian relaxation decreases as the size of the problem increases.

Gromicho, Paixão, and Bronco [36] develop another exact algorithm based on quasi-assignment relaxation by relaxing the subtour elimination constraints. They address a problem where the number of travelers is fixed. They also apply another bounding constraint to strengthen the lower bound via different r-arborescence and r-anti-arborescence relaxations.

2.2.3 Exact Algorithms to Solve MDVRP

The MDVRP is NP-hard [37], and a number of methods have been developed to solve it to optimality. There is a considerable amount of literature on the MDVRP by Montoya-Torres, Franco, Isaza, Jiménez, and Herazo-Padilla [38]. Here we explain some of the work referred to in this paper. Some research uses the branch-and-bound algorithm to solve the MDVRP. Laporte,

Nobert, and Arpin [39] use a branch-and-bound algorithm to solve MDVRP to optimality. The maximum size of the problem involves 50 customers. Another exact method is developed for the MDVRP, whereby the problem is transformed into an equivalent constrained assignment problem, and then a branch-and-bound algorithm is used to solve the transformed problem. The maximum size of the problem involves 80 customers and three depots [40]. Kek, Cheu, and Meng [41] address two problems of the fixed-destination and non-fixed-destination MDVRP called distance-constrained vehicle routing problems. An integer programming formulation is presented, and a branch-and-bound algorithm is performed to solve the problems.

The branch-and-cut algorithm can solve the MDVRP as well. Benavent and Martínez [42] present an ILP for the multiple depot multiple traveling salesman problem. Their mathematical model uses some valid inequalities to obtain a strong formulation. Some facet-defining inequalities are introduced, and a branch-and-cut algorithm is developed. This algorithm is based on the polyhedron concept and uses the facet including cuts.

Kulkarni and Bhave [43]; Laporte, Nobert, and Taillefer [44]; and Carpaneto, Dell'Amico, Fischetti, and Toth [45] present some exact methods. The mathematical model was presented by Kulkarni and Bhave [43] and then revised by Laporte et al. [44].

Some exact algorithms are based on the set partitioning problem (SPP) formulation. We refer to the work of Baldacci and Mingozzi [46] who propose an exact algorithm to solve different classes of the VRP, including the multiple depot vehicle routing problem, heterogeneous vehicle routing problem, and split delivery vehicle routing problem. Their algorithm is based on a SPP formulations. To solve the problem as an ILP, they use a bounding procedure to reduce the number of variables. In this procedure, the mathematical model is relaxed by using LP-relaxation and Lagrangian relaxation. Seth, Klabjan, and Ferreira [47] present a set of iterated tour partitioning

vehicle routing algorithms to solve a novel mobile depot problem where customers are partitioned into groups, and partially loading the vehicles is an option. Moreover, Contardo, Cordeau, and Gendron [48] develop a cut-and-column generation algorithm to solve the capacitated location routing problem. This algorithm is based on SPP formulations, and valid inequalities used in flow formulations are added to the problem as well. First, all subsets are determined using a two indexed formulation, and then each subset is solved using a column generation. Contardo and Martinelli [49] address an MDVRP with capacity and route-length constraints. An exact algorithm is developed based on a SPP formulation and vehicle flow formulation. A set of valid inequalities are added to problems to strengthen both problems. Other invalid inequalities are added to eliminate cycles.

Other work proposes an MILP to model MDVRP considering time windows. Isaza, Franco, and Herazo-Padilla [50] present an ILP for solving a heterogeneous fleet MDVRP with time windows. Dondo, Méndez, and Cerdá [51] propose an MILP framework for the multiple depot heterogeneous vehicle routing problem with time windows. Their procedure provides an optimal schedule and optimal size of fleet for each route. For larger-size problems, some elimination rules are implemented. Dondo, Méndez, and Cerdá [52] propose an MILP to solve a variant of the multiple vehicle constrained pickup and delivery with time windows problem. But for larger-size problems, a local search approach is implemented based on two evolutionary steps. By allowing exchanges among trips and then reordering the nodes on each route of the vehicle, a neighborhood structure around the starting solution is generated. If the solution is improved, then these steps are repeated until no better solution is found. Li, Li, and Pardalos [53] solve a multiple depot vehicle routing problem with time windows and without a fixed destination. They formulate the problem as an integer model, and they also develop a hybrid genetic algorithm to solve the problem.

Cornillier, Boctor, and Renaud [54] propose a multiple depot petrol station replenishment problem with time windows (MPSRPTW) in which the goal is to optimize the delivery of petroleum products to a set of petrol distribution stations to maximize net revenue allowing to deliver all demands.

2.3 Heuristics/Metaheuristics

In this section, we explain some of the variety of heuristics and metaheuristics used to solve the TSP, VRP/MTSP, and MDVRP. We address the most frequently used algorithms to solve these problems.

2.3.1 Heuristic Algorithms to Solve TSP

Heuristic algorithms for the TSP can be categorized into different groups: constructive, improvement, and hybrid. In constructive algorithms, a route is constructed by adding nodes to a route. If there is a constraint that should be maintained, then adding a node at a time can help to check the feasibility of the route until a route is constructed [55].

In improvement algorithms, a given route is improved by transposing two nodes based on the amount of savings. There are two different cases in improvement algorithms. In the first case, whole possible exchanges are considered for two nodes, and the best one is chosen. The 2-OPT and 3-OPT algorithms are instances of these algorithms. In the second case, two nodes are transposed until an improvement is done. If an improvement is gained, then the transposition of other nodes will be stopped, and the exchange will be done. Greedy 2-OPT and 3-OPT algorithms are examples of these algorithms. The 2-OPT and 3-OPT algorithms come from the family of ejection chain algorithms. More specifically, they are derived from the Lin-Kernighan, and stem and cycle methods [56].

In hybrid algorithms, an initial solution is constructed by a construction algorithm, and an improvement is implemented based on an algorithm. Genetic algorithms, simulated annealing, and particle swarm optimization algorithms are examples of those algorithms [57, 58, 59, 60].

2.3.2 Heuristic Algorithms to Solve MTSP/VRP

Most real-world problems are solved using heuristic methods. The most commonly used heuristics are tabu search, ant colony optimization, genetic algorithm, and simulated annealing. [61, 62, 63, 64]

A significant body of the MTSP/VRP literature employs tabu search. Taillard [61] and Rochat and Taillard [65] use TS to provide a benchmark for VRPs. Toth and Vigo [66] propose a granular TS algorithm, whereby a priori is used to remove the edges from the graph that has the least probability of being in the optimal solution in order to reduce computation time. This idea is conducted with TS first, but it can be applied to other algorithms as well. Toth and Vigo [66] recommend that incident edges to the depot as well as edges whose length is not more than a given granularity threshold should be kept.

Cordeau, Laporte, and Mercier [67] develop the unified tabu search algorithm (UTSA) to solve the periodic vehicle routing problem (PVRP) and the multiple depot vehicle routing problem. This algorithm is also extended to other problems such as the site-dependent VRP and to the time windows version of these problems. It has some features of the tabu route, which consider infeasible solutions through the use of a generalized objective function with self-adjusting coefficients, and another feature known as continuous diversification. Li, Golden, and Wasil [68] develop another heuristic, whereby they combine a record-to-record principle with a variable-length neighbor list whose principle is similar to the granular tabu search.

Some studies use simulated annealing to solve VRPs [64, 69]. Another method to solve VRPs is ant colony optimization. In this study, a two-optimal heuristic with artificial ants is used to tackle the problem, but the result is inferior to the tabu search results [62, 70].

Genetic algorithms are used widely among the metaheuristics in the literature. GAs are used to solve VRPs with time windows [71, 72]. The GA is also used to solve the VRP with backhauls [73], a multiple depot routing problem [74], and a school bus routing problem [75].

Kallel and Boujelbene [10] propose some dynamic performance measures added to the HVRP and solve it using a new scheme based on a clustering genetic algorithm. Since information related to the planning of the routes is not known by the planner when the routing starts, this problem is dynamic. The problem addressed in this paper is a combination of HVRP and dynamic VRP including the priority of customers. The GA they propose uses a k-means clustering method to have a clustering genetic algorithm.

There is also a hybrid approach to VRP, which is a combination of neural networks and GAs [76]. Jeon, Leep, and Shim [63] propose an HGA, where they consider the improvement of generations for an initial solution as well as three different heuristic processes and a float mutation rate for escaping from the local solution to address a VRP with heterogeneous vehicles, double trips, and multiple depots.

Alba and Dorronsoro [77] propose a cellular genetic algorithm (CGA) to solve the vehicle routing problem. Here, the population is structured as a two-dimensional toroidal grid, and the neighborhood defined on it contains five individuals: the considered one, plus the north, east, west, and south individuals. They also use a CGA to solve the capacitated VRP. In their basic CGA, they create the population in a two-dimensional toroidal grid, and the neighborhood defined on it has five individuals as well. The first parent is chosen using a binary tournament on the

neighborhood, while the other parent is the current individual. Each current individual is considered iteratively. Crossover and mutation are applied to the individuals, and a local search technique is performed for each individual. In their algorithm, they use an edge recombination operator and develop a new combined mutation, which is a combination of insertion, swap, and inversion. In their local search, they use 2-OPT and ƛ-interchange methods [78].

Wang and Lu [79] propose an HGA to solve capacitated vehicle routing problems. This algorithm consists of three stages. In the first stage, they use the nearest addition method into a sweep algorithm. Considering these two methods simultaneously provides a structured initial population where the axial and radius relationships among points is taken into account. In the second stage, response surface methodology is used to improve the crossover probability and mutation probability. To diversify the solutions, an improved sweep algorithm is performed on the HGA. In this way, the permutations of genes in a chromosome are stirred. The elitism conservation strategy is also considered to save the best chromosomes of each generation.

Marinakis and Marinaki [80] develop an HGA for the vehicle routing problem. In their HGA, they allow individuals to evolve during their lifetime. This concept is done by using the stored knowledge of a parent to teach the offspring to improve their fitness. Using a particle swarm optimizer, an individual will be improved until it reaches the requirements of a parent. In this way, the knowledge of each parent can be transferred to its offspring and the entire offspring of that generation. Therefore, the possibility of exploring a more effective solution is improved.

Potvin and Bengio [81] use a GA to solve the VRPTW. In the reproduction phase, a stochastic universal selection is used to select parents for the crossover. In the recombination phase, two types of crossover operators are used: sequence-based and route-based. In the mutation phase, three different types of mutation operators are available in a generous, one-level exchange

and two-level exchange and a local search. In the one-level exchange, a route is selected, and then for each customer in that route, the feasible insertion place that minimizes the detour over all other routes is found, and if there is such a place, then the insertion will be done. In the second mutation approach, a two-level exchange is done. The third mutation method is a local search method that starts with an initial feasible solution. By considering a sequence of consecutive customers and by moving this sequence at another feasible location within the same route or within another route, this procedure will be repeated until no improvement is possible.

Vidal, Crainic, Gendreau, and Lahrichi [82] develop an HGA for MDVRPs and PVRPs. They address three VRPs: multiple depot, periodic, and multiple depot periodic with capacitated vehicles and constrained route duration. The HGA is a combination of the exploration breadth population-based evolutionary search, aggressive improvement capabilities of neighborhood-based metaheuristics, and advanced population diversity management schemes. There is a set of three chromosomes: pattern, depot, and giant tour. The parent selection is performed based on a binary tournament, and a new periodic crossover with insertions is proposed. An education operator is applied with a certain probability to improve the quality of the offspring solution. Education goes beyond the classical GA concepts of random mutation and enhancement through hill-climbing techniques, because it includes several local-search procedures based on neighborhoods for the VRP. A repair phase eventually completes the education operator when the educated offspring is infeasible. For diversification, in each population, part of the subpopulation will be eliminated, and some new solutions will be added to the population using the same initialization method.

Vidal, Crainic, Gendreau, and Prins [83] develop an HGA with adaptive diversity management for a large class of vehicle routing problems with time windows. Some new move

evaluation techniques are proposed, and they consider some penalties for infeasible solutions with respect to time window and duration constraints, and moves from any classical neighborhood based on arc or node exchanges are evaluated. Also, some geometric and structural problem decompositions are developed to address efficiently large problems.

Berger and Barkaoui [84] develop an HGA that combines evolutionary search and local search. This memetic algorithm uses two populations. In each generation, the new offspring will replace the worst solutions of each population in order to keep the population size constant. Elitism exists to solve the best elements of each generation. In crossover, just one offspring is created from two parents. A partial solution is generated by obtaining a number of routes from the first parent and then completing it by inserting some of the vertices from the second parent using a proximity criterion (closeness to the centroid of a route) or by creating new routes. The insertion mechanism includes a random choice of the cost function parameters. Improvement is achieved by performing a large-scale neighborhood search using three different insertion mechanisms. The first insertion mechanism considers a combination of best reinsertion cost and a number of feasible insertions in order to rank vertices. The highest-ranked vertex is inserted based on a cheapest insertion cost criterion. A reject function is used as the second mechanism, whereby the cost of an insertion opportunity for a given vertex is compared to the best insertion cost achievable in the neighborhood of that vertex. The third mechanism is operated on two routes at a time by performing moves or swaps involving up to two vertices, and then the first improving move is implemented. Finally, an attempt to improve each route is made by removing in turn each vertex and reinserting.

Several heuristics are used to solve VRPTW based on route construction, route improvement, or the integration of both. Route construction heuristics consist of savings, a time-oriented nearest-neighbor insertion, and a time-oriented sweep heuristic [85]. Among these

algorithms, the time-oriented nearest-neighbor insertion heuristic is found to be successful. Other route construction procedures are the parallel insertion method [86], the greedy randomized adaptive procedure [87], and the generalized assignment heuristic [88]. A number of route-improvement heuristics include the branch exchange procedure [89, 90] and cyclic transfer algorithms [91].

Thangiah [92] develop another heuristic search strategy based on the genetic algorithm. This approach is a "cluster-first route-second" strategy. A set of customer clusters that are routed is found using the cheapest insertion procedure, and the routes are improved using a post-optimization procedure, which results in good feasible solutions in the genetic algorithm.

Alvarenga, Mateus, and De Tomi [93] propose a two-phase procedure (i.e., genetic and set partitioning) for the VRPTW. In the first phase, the objective is to produce high-quality routes for the set partitioning problem. The literature shows that several heuristics produce local minimal solutions. In their paper, they use a fast genetic algorithm to produce minimal solutions. Then they use the initial solutions produced in the SPP. In their SPP, since it is not possible for the SPP model to handle any mixed-integer program (MIP) algorithm, some decomposition method, such as the Wolf-Dantzig decomposition method, is applied.

Other evolutionary algorithms in the literature as well solve the VRP. Prins [94] proposes a new heuristic algorithm that combines the two main features of an evolutionary search, crossover and mutation operations. This mimetic algorithm includes vertex and edge reinsertions, vertex swaps, combined vertex and edge swaps, and edge swaps. A local search is used to improve a candidate solution. After finding the first improving solution, the search is ended. Solutions are presented as a sequence of customers. First, it is assumed that there is a single vehicle, and all the vertices except the depot will appear in the solution without considering the tour delimiters and

without going through the depot. Then by solving a shortest-path problem on an auxiliary graph, an optimal partition of this cycle is determined, which is similar to “route-first cluster-second” algorithms. This procedure is performed after mutation as well. The offspring are generated by selecting two positions on the first and second parent, and switching the partitions that lie on the sides of the two cutting points.

2.3.3 Heuristic Algorithms to Solve MDVRP

Since the MDVRP is an NP-hard problem, many metaheuristics in the literature address the problem. The tabu search algorithm is one of the first heuristics used to handle this problem. Renaud, Boctor, and Laporte [95] apply a TS to an MDVRP problem with vehicle capacities and a maximum duration of route constraints. The initial solution is constructed based on an improved petal heuristic developed by Renaud, Boctor, and Laporte [96]. Cordeau, Gendreau, and Laporte [97] develop another TS algorithm in which an initial solution is randomly generated. Their model can be applied to the PVRP. Other TS heuristics in the literature include the Tuzun and Burke algorithm [98] in which the objective is to minimize the total cost of routing, and the Cordeau, Laporte, and Mercier [99] algorithm in which the objective is to minimize the number of vehicles.

Parthanadee and Logendran [100] use three different TS algorithms to tackle a MDVRP in which every depot is in charge of supplying the demand of its customers, and they interact with other depots in the case of having over supplies.

In addition to the TS, other heuristic algorithms solve the MDVRP as well. Some heuristic methods are based on construction techniques and apply some improvement methods to the solutions thereafter. We can refer to the work of Tillman [101] who uses the savings method proposed by Clark and Wright [102]. Here there is a multi-phase heuristic where for each depot, the radial routes are found optimally, and then a GA is applied to merge the radial routes into a

network structure. Skok, Skrlec, and Krajcar [103] use a GA to solve a non-fixed-destination MDVRP. They investigate that the quality of the solution might be deteriorated due to decomposition of the problem to independent subsets, and therefore, GA-based algorithms [104, 105] can help to find the solution of the non-fixed-destination MDVRP.

Few works in the literature address solutions for the fixed-destination MDVRP using a GA. The major work in this area is the Thangiah and Salhi method [106], who use indirect encoding based on the permutation of customers. In this algorithm, customers do not represent the routes; rather, the routes are built using an intelligent route scheduler from the permutations, which uses the problem constraints as a guide. The objective here is to minimize the total number of vehicles and the total distance. In their algorithm, they have an adaptive method based on geometric shapes. Customers are assigned to different depots based on the clustering algorithm, and then a solution is found in each cluster.

2.4 Summary

This chapter provides an overview on different types of TSPs and VRPs that exist in the literature. Each of these well-studied problems can be extended to new variant problems by changing the objectives and constraints. Extensive methods to solve both problems can be categorized into two main groups: exact algorithms and heuristics/metaheuristics. As we address new classes of the TSP, MTSP, VRP, and MDVRP, an overview on the exact methods and heuristic/metaheuristic algorithms to solve each one is provided.

2.5 References

[1] Bektas, T. (2006). The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, *34*(3), 209-219.

[2] Bodin, L., Golden, B., Assad, A., & Ball, M. (1981). The state of the art in the routing and scheduling of vehicles and crews. Monograph UMTA-MD-11-0004-81-2Final Rpt.

[3] Baldacci, R., Battarra, M., & Vigo, D. (2008). Routing a heterogeneous fleet of vehicles. In *The vehicle routing problem: Latest advances and new challenges* (pp. 3-27). Springer U.S.

[4] Toth, P., & Vigo, D. (2002). Models, relaxations and exact approaches for the capacitated vehicle routing problem. *Discrete Applied Mathematics*, *123*(1), 487-512.

[5] Lim, A., & Wang, F. (2005). Multi-depot vehicle routing problem: A one-stage approach. *IEEE Transactions on Automation Science and Engineering*, *2*(4), 397-402.

[6] Kallehauge, B., Larsen, J., Madsen, O. B., & Solomon, M. M. (2005). Vehicle routing problem with time windows. In *Column generation* (pp. 67-98). Springer U.S.

[7] Gendreau, M., Laporte, G., & Séguin, R. (1996). Stochastic vehicle routing. *European Journal of Operational Research*, *88*(1), 3-12.

[8] Min, H. (1989). The multiple vehicle routing problem with simultaneous delivery and pick-up points. *Transportation Research Part A: General*, *23*(5), 377-386.

[9] Zhang, W. X., & Korf, R. E. (1996). A study of complexity transitions on the asymmetric traveling salesman problem. *Artificial Intelligence*, *81*(1–2), 223–239.

[10] Kallel, S. A., & Boujelbene, Y. (2013). Heterogeneous vehicle routing problem with profits dynamic solving by clustering genetic algorithm. *International Journal of Computer Science Issues*, *10*(4), 1-7.

[11] Laporte, G. (1992). The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, *59*(2), 231-247.

[12] Dantzig, G., Fulkerson, R., & Johnson, S. (1954). Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, *2*(4), 393-410.

[13] Miller, C. E., Tucker, A. W., & Zemlin, R. A. (1960). Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, *7*(4), 326-329.

[14] Eastman, W. L. (1958). *Linear programming with pattern constraints* (Doctoral thesis). Harvard University.

[15] Little, J. D., Murty, K. G., Sweeney, D. W., & Karel, C. (1963). An algorithm for the traveling salesman problem. *Operations Research*, *11*(6), 972-989.

[16] Tarjan, R. E. (1977). Finding optimum branchings. *Networks*, *7*(1), 25-35.

[17] Fischetti, M., & Toth, P. (1992). An additive bounding procedure for the asymmetric travelling salesman problem. *Mathematical Programming*, *53*(1-3), 173-197.

[18] Christofides, N. (1970). The shortest Hamiltonian chain of a graph. *SIAM Journal on Applied Mathematics*, *19*(4), 689-696.

[19] Held, M., & Karp, R. M. (1971). The traveling-salesman problem and minimum spanning trees: Part II. *Mathematical programming*, *1*(1), 6-25.

[20] Martin, G. T. (1966). Solving the traveling salesman problem by integer linear programming. *Operations Research* (p. B71).

[21] Miliotis, P. (1976). Integer programming approaches to the travelling salesman problem. *Mathematical Programming*, *10*(1), 367-378.

[22] Fischer, A., Fischer, F., Jäger, G., Keilwagen, J., Molitor, P., & Grosse, I. (2014). Exact algorithms and heuristics for the quadratic traveling salesman problem with an application in bioinformatics. *Discrete Applied Mathematics*, *166*, 97-114.

[23] Grötschel and M. W. Padberg. (1985). Polyhedral theory. In E. Lawler, J. Lenstra, A. Rinnooy Kan and D. Shmoys (Eds.), *The traveling salesman problem: A guided tour of combinatorial optimization* (pp. 251-305). New York: Wiley.

[24] Grötschel, M., & Padberg, M. W. (1979). On the symmetric travelling salesman problem I: Inequalities. *Mathematical Programming*, *16*(1), 265-280.

[25] Grötschel, M., & Pulleyblank, W. R. (1986). Clique tree inequalities and the symmetric travelling salesman problem. *Mathematics of Operations Research*, *11*(4), 537-569.

[26] Cornuéjols, G., Fonlupt, J., & Naddef, D. (1985). The traveling salesman problem on a graph and some related integer polyhedra. *Mathematical Programming*, *33*(1), 1-27.

[27] Naddef, D., & Rinaldi, G. (1992). The crown inequalities for the symmetric traveling salesman polytope. *Mathematics of Operations Research*, *17*(2), 308-326.

[28] Crowder, H., & Padberg, M. W. (1980). Solving large-scale symmetric travelling salesman problems to optimality. *Management Science*, *26*(5), 495-509.

[29] Padberg, M., & Rinaldi, G. (1990). Facet identification for the symmetric traveling salesman polytope. *Mathematical Programming*, *47*(1-3), 219-257.

[30] Grötschel, M., & Holland, O. (1991). Solution of large-scale symmetric travelling salesman problems. *Mathematical Programming*, *51*(1-3), 141-202.

[31] Applegate, D., Bixby, R., Chvátal, V. & Cook, W. (1998). On the solution of travelling salesman problems. *Documenta mathematica*, *3*, 645-656.

[32] Applegate, D., Bixby, R., Chvátal, V., & Cook, W. (1995). *Finding cuts in the TSP (A preliminary report).* Technical Report, Center for Discrete Mathematics & Theoretical Computer Science.

[33] Laporte, G., & Nobert, Y. (1980). A cutting planes algorithm for the m-salesmen problem. *Journal of the Operational Research Society*, *31*(11), 1017-1023.

[34] Ali, A. I., & Kennington, J. L. (1986). The asymmetric M-travelling salesmen problem: A duality based branch-and-bound algorithm. *Discrete Applied Mathematics*, *13*(2-3), 259-276.

[35] Gavish, B., & Srikanth, K. (1986). An optimal solution method for large-scale multiple traveling salesmen problems. *Operations Research*, *34*(5), 698-717.

[36] Gromicho, J., Paixão, J., & Bronco, I. (1992). Exact solution of multiple traveling salesman problems. In *Combinatorial Optimization* (pp. 291-292). Berlin Heidelberg: Springer.

[37] Ombuki-Berman, B., & Hanshar, F. T. (2009). Using genetic algorithms for multi-depot vehicle routing. In *Bio-inspired algorithms for the vehicle routing problem* (pp. 77-99). Berlin Heidelberg: Springer.

[38] Montoya-Torres, J. R., Franco, J. L., Isaza, S. N., Jiménez, H. F., & Herazo-Padilla, N. (2015). A literature review on the vehicle routing problem with multiple depots. *Computers & Industrial Engineering*, *79*, 115-129.

[39] Laporte, G., Nobert, Y., & Arpin, D. (1984). Optimal solutions to capacitated multidepot vehicle routing problems. *Congressus Numerantium*, *44*, 283.

[40] Laporte, G., Nobert, Y., & Taillefer, S. (1988). Solving a family of multi-depot vehicle routing and location-routing problems. *Transportation Science*, *22*(3), 161-172.

[41] Kek, A. G., Cheu, R. L., & Meng, Q. (2008). Distance-constrained capacitated vehicle routing problems with flexible assignment of start and end depots. *Mathematical and Computer Modelling*, *47*(1), 140-152.

[42] Benavent, E., & Martínez, A. (2013). Multi-depot multiple tsp: polyhedral study and computational results. *Annals of Operations Research*, *207*(1), 7-25.

[43] Kulkarni, R. V., & Bhave, P. R. (1985). Integer programming formulations of vehicle routing problems. *European Journal of Operational Research*, *20*(1), 58-67.

[44] Laporte, G., Nobert, Y., & Taillefer, S. (1988). Solving a family of multi-depot vehicle routing and location-routing problems. *Transportation Science*, *22*(3), 161-172.

[45] Carpaneto, G., Dell'Amico, M., Fischetti, M., & Toth, P. (1989). A branch and bound algorithm for the multiple depot vehicle scheduling problem. *Networks*, *19*(5), 531-548.

[46] Baldacci, R., & Mingozzi, A. (2009). A unified exact method for solving different classes of vehicle routing problems. *Mathematical Programming*, *120*(2), 347-380.

[47] Seth, A., Klabjan, D., & Ferreira, P. M. (2013). Analyses of advanced iterated tour partitioning heuristics for generalized vehicle routing problems. *Networks*, *61*(4), 290-308.

[48] Contardo, C., Cordeau, J. F., & Gendron, B. (2013). An exact algorithm based on cut-and-column generation for the capacitated location-routing problem. *INFORMS Journal on Computing*, *26*(1), 88-102.

[49] Contardo, C., & Martinelli, R. (2014). A new exact algorithm for the multi-depot vehicle routing problem under capacity and route length constraints. *Discrete Optimization*, *12*, 129-146.

[50] Isaza, S. N., Franco, J. L., & Herazo-Padilla, N. (2012, July). Desarrollo y Codificación de un Modelo Matemático para la Optimización de un Problema de Ruteo de Vehículos con Múltiples Depósitos. In *Tenth LACCEI Latin American and Caribbean Conference* (LACCEI'2012), Megaprojects: Building Infrastructure by fostering engineering collaboration, efficient and effective integration and innovative planning (pp. 23-27).

[51] Dondo, R., Méndez, C. A., & Cerdá, J. (2003). An optimal approach to the multiple-depot heterogeneous vehicle routing problem with time window and capacity constraints. *Latin American Applied Research*, *33*(2), 129-134.

[52] Dondo, R., Méndez, C. A., & Cerdá, J. (2008). Optimal management of logistic activities in multi-site environments. *Computers & Chemical Engineering*, *32*(11), 2547-2569.

[53] Li, J., Li, Y., & Pardalos, P. M. (2016). Multi-depot vehicle routing problem with time windows under shared depot resources. *Journal of Combinatorial Optimization*, *31*(2), 515-532.

[54] Cornillier, F., Boctor, F., & Renaud, J. (2012). Heuristics for the multi-depot petrol station replenishment problem with time windows. *European Journal of Operational Research*, *220*(2), 361-369.

[55] Kim, B. I., Shim, J. I., & Zhang, M. (1998). Comparison of TSP algorithms: Project for models in facilities planning and materials handling. URL: http://www.slideshare.net/kaalnath/comparison-of-tsp-algorithms

[56] Lin, S., & Kernighan, B. (1973). An effective heuristic algorithm for the traveling salesman problem. *Operations Research 21*, 498−516.

[57] Schmitt, L. J., & Amini, M. M. (1998). Performance characteristics of alternative genetic algorithmic approaches to the traveling salesman problem using path representation: An empirical study. *European Journal of Operational Research*, *108*(3), 551–570.

[58] Wang, Y. (2014). The hybrid genetic algorithm with two local optimization strategies for traveling salesman problem. *Computers & Industrial Engineering*, *70*, 124-133.

[59] Munakata, T., & Nakamura, Y. (2001). Temperature control for simulated annealing. *Physical Review E*, *64*(4), 046127.

[60] Clerc, M. (2004). Discrete particle swarm optimization, illustrated by the traveling salesman problem. In *New optimization techniques in engineering* (pp. 219-239). Springer Berlin Heidelberg

[61] Taillard, É. (1993). Parallel iterative search methods for vehicle routing problems. *Networks*, *23*(8), 661-673.

[62] Bullnheimer, B., Hartl, R. F., & Strauss, C. (1999). An improved ant system algorithm for the vehicle routing problem. *Annals of Operations Research*, *89*, 319-328.

[63] Jeon, G., Leep, H. R., & Shim, J. Y. (2007). A vehicle routing problem solved by using a hybrid genetic algorithm. *Computers & Industrial Engineering*, *53*(4), 680-692.

[64] Osman, I. H. (1993). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, *41*(4), 421-451.

[65] Rochat, Y., & Taillard, É. D. (1995). Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, *1*(1), 147-167.

[66] Toth, P., & Vigo, D. (2003). The granular tabu search and its application to the vehicle-routing problem. *Informs Journal on Computing*, *15*(4), 333-346.

[67] Cordeau, J. F., Laporte, G., & Mercier, A. (2001). A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational Research Society*, *52*(8), 928-936.

[68] Li, F., Golden, B., & Wasil, E. (2005). Very large-scale vehicle routing: new test problems, algorithms, and results. *Computers & Operations Research*, *32*(5), 1165-1179.

[69] Hiquebran, D. T., Alfa, A. S., Shapiro, J. A., & Gittoes, D. H. (1993). A revised simulated annealing and cluster-first route-second algorithm applied to the vehicle routing problem. *Engineering Optimization*, *22*(2), 77-107.

[70] Gambardella, L. M., Taillard, É., & Agazzi, G. (1999). MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. In D. Corne, M. Dorigo, and F. Glover (Eds.), *New ideas in optimization* (pp. 63-76), London: McGraw-Hill .

[71] Thangiah, S. R., Nygard, K. E., & Juell, P. L. (1991, February). Gideon: A genetic algorithm system for vehicle routing with time windows. In *Proceedings of the Seventh IEEE Conference on Artificial Intelligence Applications*, *1*, 322-328).

[72] Blanton Jr, J. L., & Wainwright, R. L. (1993, June). Multiple vehicle routing with time and capacity constraints using genetic algorithms. In *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 452-459). Morgan Kaufmann Publishers Inc.

[73] Potvin, J. Y., Duhamel, C., & Guertin, F. (1996). A genetic algorithm for vehicle routing with backhauling. *Applied Intelligence*, *6*(4), 345-355.

[74] Salhi, S., Thangiah, S. R., & Rahman, F. (1998). A genetic clustering method for the multi-depot vehicle routing problem. In *Artificial Neural Nets and Genetic Algorithms* (pp. 234-237). Vienna: Spriner.

[75] Thangiah, S. R., & Nygard, K. E. (1992, March). School bus routing using genetic algorithms. In *Aerospace sensing* (pp. 387-398). International Society for Optics and Photonics.

[76] Potvin, J. Y., Dubé, D., & Robillard, C. (1996). A hybrid approach to vehicle routing using neural networks and genetic algorithms. *Applied Intelligence*, *6*(3), 241-252.

[77] Alba, E., & Dorronsoro, B. (2004, April). Solving the vehicle routing problem by using cellular genetic algorithms. In *European Conference on Evolutionary Computation in Combinatorial Optimization* (pp. 11-20). Berlin Heidelberg: Springer.

[78] Osman, I. H. (1993). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, *41*(4), 421-451.

[79] Wang, C. H., & Lu, J. Z. (2009). A hybrid genetic algorithm that optimizes capacitated vehicle routing problems. *Expert Systems with Applications*, *36*(2), 2921-2936.

[80] Marinakis, Y., & Marinaki, M. (2010). A hybrid genetic-particle swarm optimization algorithm for the vehicle routing problem. *Expert Systems with Applications*, *37*(2), 1446-1455.

[81] Potvin, J. Y., & Bengio, S. (1996). The vehicle routing problem with time windows, Part II: Genetic search. *INFORMS Journal on Computing*, *8*(2), 165-172.

[82] Vidal, T., Crainic, T. G., Gendreau, M., Lahrichi, N., & Rei, W. (2012). A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, *60*(3), 611-624.

[83] Vidal, T., Crainic, T. G., Gendreau, M., & Prins, C. (2013). A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & Operations Research*, *40*(1), 475-489.

[84] Berger, J., & Barkaoui, M. (2000). An improved hybrid genetic algorithm for the vehicle routing problem with time windows. In *International ICSC Symposium on Computational Intelligence*, part of the International ICSC Congress on Intelligent Systems and Applications (ISA'2000), University of Wollongong, Wollongong, Australia.

[85] Bräysy, O., & Gendreau, M. (2005). Vehicle routing problem with time windows, Part I: Route construction and local search algorithms. *Transportation Science*, *39*(1), 104-118.

[86] Savelsbergh, M. W. P. (1990). A parallel insertion heuristic for vehicle routing with side constraints. *Statistica Neerlandica*, *44*(3), 139-148.

[87] Pitsoulis, L. S., & Resende, M. G. (2002). Greedy randomized adaptive search procedures. In *Handbook of applied optimization* (pp. 168-182). New York: Oxford University Press.

[88] Fisher, M. L., & Jaikumar, R. (1981). A generalized assignment heuristic for vehicle routing. *Networks*, *11*(2), 109-124.

[89] Stewart, W. R. (1987). Accelerated branch exchange heuristics for symmetric traveling salesman problems. *Networks*, *17*(4), 423-437.

[90] Golden, B., Bodin, L., Doyle, T., & Stewart Jr, W. (1980). Approximate traveling salesman algorithms. *Operations Research*, *28*(3-part-ii), 694-711.

[91] Thompson, P. M., & Psaraftis, H. N. (1993). Cyclic transfer algorithm for multivehicle routing and scheduling problems. *Operations Research*, *41*(5), 935-946.

[92] Thangiah, S. (1995). Vehicle routing with time windows using genetic algorithms. In *Application handbook of genetic algorithms: New frontiers* (Volume II, pages 253−277). Boca Raton, FL: CRC Press.

[93] Alvarenga, G. B., Mateus, G. R., & De Tomi, G. (2007). A genetic and set partitioning two-phase approach for the vehicle routing problem with time windows. *Computers & Operations Research*, *34*(6), 1561-1584.

[94] Prins, C. (2002). Efficient heuristics for the heterogeneous fleet multitrip VRP with application to a large-scale real case. *Journal of Mathematical Modelling and Algorithms*, *1*(2), 135-150.

[95] Renaud, J., Boctor, F. F., & Laporte, G. (1996). An improved petal heuristic for the vehicle routeing problem. *Journal of the Operational Research Society*, *47*(2), 329-336.

[96] Renaud, J., Laporte, G., & Boctor, F. F. (1996). A tabu search heuristic for the multi-depot vehicle routing problem. *Computers & Operations Research*, *23*(3), 229-235.

[97] Cordeau, J. F., Gendreau, M., & Laporte, G. (1997). A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks*, *30*(2), 105-119.

[98] Tuzun, D., & Burke, L. I. (1999). A two-phase tabu search approach to the location routing problem. *European Journal of Operational Research*, *116*(1), 87-99.

[99] Cordeau, J. F., Laporte, G., & Mercier, A. (2001). A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational Research Society*, *52*(8), 928-936.

[100] Parthanadee, P., & Logendran, R. (2006). Periodic product distribution from multi-depots under limited supplies. *IIE Transactions*, *38*(11), 1009-1026.

[101] Tillman, F. A. (1969). The multiple terminal delivery problem with probabilistic demands. *Transportation Science*, *3*(3), 192-204.

[102] Clarke, G. U., & Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, *12*(4), 568-581.

[103] Skok, M., Skrlec, D., & Krajcar, S. (2000, June). The non-fixed destination multiple depot capacitated vehicle routing problem and genetic algorithms. In *Information Technology Interfaces*, 2000. ITI 2000. Proceedings of the 22nd IEEE International Conference on Computational Cybernetics and Simulation (pp. 403-408).

[104] Filipec, M., Skrlec, D., & Krajcar, S. (1997, October). Darwin meets computers: New approach to multiple depot capacitated vehicle routing problem. In *Systems, man, and cybernetics*, *1*, 421-426. IEEE International Conference on Computational Cybernetics and Simulation, October 12-15.

[105] Filipec, M., Škrlec, D., & Krajcar, S. (2000). Genetic algorithm approach for multiple depot capacitated vehicle routing problem solving with heuristic improvements. *International Journal of Modelling and Simulation*, *20*(4), 320-328.

[106] Thangiah, S. R., & Salhi, S. (2001). Genetic clustering: An adaptive heuristic for the multi depot vehicle routing problem. *Applied Artificial Intelligence*, *15*(4), 361–383.

CHAPTER 3

HYBRID GENETIC ALGORITHM WITH LOCAL OPTIMIZATION STRATEGY FOR TRAVELING SALESMAN PROBLEM WITH PARTIAL COVERAGE

Abstract

This chapter introduces a new variant of the traveling salesman problem, or traveling salesman problem with partial coverage. Here, the goal is to maximize profit (served demand), subject to a limited budget and travel time. A partial demand of nodes that are not on the route can be maintained by those nodes that are on the route. Furthermore, each node has a limited capacity to serve other nodes, and there is a predetermined distance for each node to travel in order to obtain service. A genetic algorithm with local optimization has been designed to provide good quality solutions in a reasonable amount of time to the TSPWPC. In this approach, instead of mutation, first, the nodes are organized based on a clustering algorithm. Consequently, the travel cost and travel time are reduced and more budget is released. Then, within the released budget, some nodes that are not being served are inserted to the clusters. These nodes are added based on the closeness of their polar angle to the average polar angle of the clusters. To measure the efficiency of the proposed GA, another genetic algorithm with an insertion mutation is developed. In this mutation, cities are added based on the closeness of their polar angles to the average polar angle of nodes on the route. Both GAs are coded in C. Moreover to measure the efficiency of the GA approaches, the problem has been solved as an MILP. Due to problem complexity, dynamic cuts are added to the problem in order to eliminate the subtours. Because execution time of the MILP is sensitive to the amount of budget, this is set such that the problem can be executed in a reasonable period of time. A variety of instances with different distributions and sizes are generated by GIS software to perform GA approaches and the MILP. The results are presented and analyzed.

3.1 Introduction

The purpose of this chapter is to introduce a new variant of the traveling salesman problem, called the traveling salesman problem with partial coverage, and designing a hybrid genetic algorithm to solve it. The TSPWPC is defined on an undirected graph $G = (V \cup W \cup Z, E)$, where $V \cup W \cup Z$ the vertex is set, and $E = \{(v_i, v_j): v_i, v_j \in V \cup W \cup Z, \text{i} < j\}$ is the edge set. Set V is the set of nodes served directly, W is the set of nodes served indirectly by nodes of set V, and set Z is the set of nodes not being served. A cost c_{ij} and a travel time t_{ij} are associated with each edge (v_i, v_j). Demand d_k and capacity cap_k are associated with each vertex v_k. Profit p_k can be attained from each node, which depends on the nodes visited previously. Profit is expressed in terms of served demand at each node. In this problem, it is assumed that a partial demand of each node can be maintained by other cities being served, providing that the distance between those nodes is not greater than a predetermined distance, and the capacity of serving the city is large enough. Because the budget and time to serve the nodes are both limited, all nodes cannot be fitted into set $V \cup W$. The aim of the TSPWPC is to make a subtour of maximal profit that can serve the vertices of $V \cup W$ subject to budget and time constraints.

Application of the TSPWPC involves planning a tour for a doctor in an undeveloped region to serve the maximum number of patients. The amount of time spent and the cost of the tour are limited. The service capacity of each node is determined. A portion of patients in other nodes can be served by nodes on the tour, providing that the distance between the nodes is not greater than a predetermined distance. Another application of the TSPWPC is in health economics, where the problem is to find a subtour for the registrar of special diseases traveling to different hospitals to collect patient documents, which is an increasing function. The aim here is to maximize the total number of served hospitals within a limited budget [1]. Another application of the TSPWPC is

disk defragmentation, whereby the objective is to have more assigned storage clusters to a corresponding file within a limited period of time [2]. In disaster management problems, one objective can be to maximize the number of injured people served by trained medical staff who must travel to different locations affected by the disaster, assuming that injured people could be transferred to locations where the medical travelers could go, subject to limited time and budget constraints [3].

In the literature, there are many studies on the TSP with profits as well as studies on visiting a subset of vertices. Beasley and Nascimento [4] solve a single vehicle routing allocation problem (SVRAP) in which the objective is to minimize routing costs, allocation costs, and isolated customer costs. The SVRAP is different from the TSPWPC, since it does not require that all customers are visited by a vehicle, and customers not visited can be covered by nodes on the route. Current and Schilling [5] address the covering salesman problem with the objective of minimizing the cost of the tour, where it is not required to visit all cities, but the remaining cities should lie within the covering standard distance of a city on the salesman route. Arkin and Hassin [6] introduce a geometric version of the CSP in which a compact set constructs the neighborhood, and whole vertices of the neighborhood are covered by a set intersection. This model is aimed to find a minimum-length tour through intersection of all neighborhoods starting from and returning to the depot. Gendreau, Laporte, and Semet [7] propose a generalization of the CSP called the covering tour problem (CTP), whereby there is an undirected graph, and its set of vertices is partitioned into three groups: the set of vertices can be visited, the set of vertices must be visited, and the set of vertices must be covered. The CTP is aimed to find a minimum-length Hamiltonian cycle over a subset of vertices of a can-be-visited set in which the tour must consist of the entire must-be-visited set vertices, and every vertex of the must-be-covered set has to be covered by at

least one vertex visited by the tour. Hachicha, Hodgson, Laporte, and Semet [8] propose a multiple vehicle covering tour problem (MVCTP) to generalize the CTP, which designs *m* Hamiltonian cycles over a subset of vertices covering whole vertices subject to side constraints. Labbé, Laporte, Rodríguez Martín, and González [9] solve the median cycle problem to determine a cycle of vertices, with the goal of minimizing the routing cost and the cost of assisting vertices that are not in the cycle. Archetti, Hertz, and Speranza [10] solve a team-orienting problem with the objective of defining a subset of customers to maximize profit while the time spent on visiting nodes is limited. In a survey by Feillet, Dejax, and Gendreau [11] on the TSP, these problems are studied based on such objectives as minimizing the tour cost, maximizing profit, or a combination of both. Our problem can be fitted into the maximizing profit group.

The TSP is NP-complete [12]. To solve this problem, exact algorithms such as integer programming methods [13] can find the solution in a reasonable period of time when the size of the problem is small [14]. However, when the problem size increases, intelligent algorithms/metaheuristics can find good solutions in a reasonable amount of time. Examples of such algorithms are the nearest neighborhood search [15], simulated annealing [16], particle swarm optimization [17], neural network [18], tabu search [19], and ant colony [20] algorithms.

Schmitt and Amini [21] propose a genetic algorithm to solve the TSP. This GA is an optimization technique based on natural evolution and the survival of the fittest concept. Here, it is not necessary to explore all possible solutions in the feasible region to obtain a good quality solution. In each iteration, the GA provides a solution that is not worse than the solutions of previous generations. Since the current solutions in each generation are the parents of the next generation, the possibility of having a better solution in the next generation increases.

For a generalized traveling salesman problem, Bontoux, Artigues, and Fillet [22] utilize clustering to partition a problem into smaller clusters and to find the best tour for each cluster using a dynamic program. In other algorithms, the best subtour over each cluster is followed by finding the best tour that merges the subtours. [23].

Wang [24] introduces a hybrid genetic algorithm with two local optimization strategies. The first local optimization is used to generate the shorter Hamiltonian paths using the four vertices and three lines of inequality, and the second local optimization is performed to reverse the local Hamiltonian paths to generate shorter Hamiltonian circuits.

This chapter introduces a new variant of the traveling salesman problem, called the traveling salesman problem with partial coverage, and proposes a hybrid genetic algorithm with a local optimization strategy that takes advantage of clustering methods. The work here contributes to the literature in terms of proposing a new model for a number of different applications as discussed earlier. The model is different than the regular problem in the literature since there is a limited capacity for each node to serve other nodes, and for each node there is a predefined distance to travel in order to receive service. An assignment problem must be solved to measure the profit that can be gained, along with finding a Hamiltonian subtour over the selected nodes. Due to the constraints of budget and time, the Hamiltonian cycle is found over part of the nodes. Therefore, the complexity of this variant problem is more than the TSP. In addition, the paper contributes to the literature by developing a new GA with two different methodologies: one uses polar angles of cities to improve the solutions, and the other is based on a clustering method to solve the TSPWPC. Providing a well-clustered instance, the solution can be improved dramatically, and more budget can be saved in terms of money and time. After this improvement, the nodes are added to routes based on their polar angles.

The remainder of the paper is organized as follows: the problem is defined in section 3.2 by using a mathematical model. In section 3.3, the mixed-integer linear program is explained. In section 3.4, a genetic algorithm to solve the proposed model is presented. Computational results and analysis are found in section 3.5, followed by conclusions and future work in section 3.6.

3.2 Mathematical Model

The TSPWPC can be considered a form of selective TSP. Here, a given set of cities with their demands exist. Furthermore, there is a limited budget and time for travel. The objective is to find the best subtour with maximum total profit, i.e., served demand, subject to limited budget and tour duration. A portion of the demand of other cities can also be served by cities on the route. The distance between the serving city and the served city cannot be more than a predetermined distance. In addition, there is a limited capacity at each serving city, i.e., each city can serve the demand of other cities up to its capacity. The total profit depends on which cities are chosen to be on the route and which cities are assigned to cities on the route.

This problem is modeled as an MILP. The notation, decision variables and mathematical model are presented as follows:

Sets

I, J: set of nodes

Parameters

$i, j \in I$: nodes indices

C_{ij}: $cost\ of\ traveling\ from\ node\ i\ to\ node\ j$

L_{ij}: $distance\ between\ node\ i\ and\ node\ j$

DIS_j: $maximum\ distance\ that\ people\ will\ travel\ to\ come\ to\ city\ j$

CAP_i: $capacity\ of\ node\ i$

E_j: $number\ of\ people\ who\ travel\ to\ city\ j$

D_i: $demand\ of\ node\ i$

tp_i: $service\ time\ at\ node\ i$

T_{ij}: $travel\ time\ between\ city\ i\ and\ city\ j$

Variables

$$Z_i = \begin{cases} 1 & if\ node\ i\ is\ on\ route \\ 0 & o.w. \end{cases}$$

$$W_i = \begin{cases} 1 & if\ node\ i\ is\ on\ route\ but\ served\ indirectly \\ 0 & o.w. \end{cases}$$

$$V_i = \begin{cases} 1 & if\ node\ i\ is\ neither\ on\ route\ nor\ served\ indirectly \\ 0 & o.w. \end{cases}$$

$$X_{ij} = \begin{cases} 1 & if\ node\ j\ immediately\ proceeds\ node\ i \\ 0 & o.w. \end{cases}$$

$$Y_{ij} = \begin{cases} 1 & if\ node\ j\ is\ assigned\ to\ node\ i \\ 0 & o.w. \end{cases}$$

Model

$$\max \sum_{i\in I} D_i Z_i + \sum_{j\in I}\sum_{i\in I} \mathrm{E}_j Y_{ij} \tag{3.1}$$

Subject to

$$Z_i + W_i + V_i = 1 \qquad \forall\, i \in I \tag{3.2}$$

$$\sum_{j\in I} X_{0j} = 1 \tag{3.3}$$

$$\sum_{i\in I} X_{i0} = 1 \tag{3.4}$$

$$\sum_{i\in I} X_{ij} = \sum_{i\in I} X_{ji} = Z_j \quad i \neq j \qquad \forall j \in I \tag{3.5}$$

$$\sum_{i\in I} Y_{ij} = W_j \qquad \forall j \in I \tag{3.6}$$

$$Z_i \geq Y_{ij} \qquad \forall i, j \in I \tag{3.7}$$

$$u_{\,i} - u_j + (N-1)x_{ij} \leq N - 2 \qquad i \neq j \tag{3.8}$$

$$\sum_{j\in I} E_j Y_{ij} + D_i Z_i \le CAP_i Z_i \qquad \forall i \in I \tag{3.9}$$

$$L_{ij} Y_{ij} \le DIS_j \qquad \forall\, i,j \in I \tag{3.10}$$

$$\sum_{j\in I}\sum_{i\in I} T_{ij} X_{ij} + \sum_{i\in I} tp_i Z_i \le T_{tour} \tag{3.11}$$

$$\sum_{j\in I}\sum_{i\in I} C_{ij} X_{ij} \le BUDGET_{tour} \tag{3.12}$$

$$1 \le u_{\,i} \le N - 1 \tag{3.13}$$

Constraint (3.2) ensures that each node takes only one of three possible states: the node is on the route, the node is not on the route but is served partially by one of the nodes on the route, or the node is not on the route and is not served indirectly. Constraints (3.3) and (3.4) represent one leaving the origin once and returning back to the origin from one node. Similarly, constraint (3.5) ensures the same condition for all other nodes that are on the route. Constraint (3.6) ensures that each node can be served indirectly by only one of the other nodes that is on the route. Constraint (3.7) implies that if node *i* is served by node *j*, then node *j* must be on the route (otherwise it cannot serve node *i*). Constraint (3.8) is the Miller-Tucker-Zemlin subtour elimination constraint [25]. Constraint (3.9) restricts the number of customers served directly and indirectly in each node by the capacity of that node. Constraint (3.10) limits the distance between the serving node and the served node. Constraints (3.11) and (3.12) imply that the time and cost of traveling cannot exceed the budget. This mathematical program includes $5\,[I] + 2[I]^2$ variables and $8\,[I] + 3[I]^2 - [I]$ constraints.

The TSPWPC is a more complex problem than the TSP and is investigated as follows:

Proposition 1. The complexity of the TSPWPC problem is $\sum_{x=1}^{n}\binom{n}{x}\frac{(x-1)!}{2}\times 2^{n-x}$.

Proof:

Because this is a selective problem subject to a budget, there can be up to *n* nodes on the route. Therefore, there can be $\sum_{x=1}^{n}\binom{n}{x}$ different sets of cities on the route. For the selected nodes,

there are $\frac{(x-1)!}{2}$ Hamiltonian cycles for an asymmetric distance matrix. In the case of having *x* nodes on a route, an assignment problem must be solved to assign *n-x* cities to the cities on the route. For each node, there are two possibilities: either assigned to one of the nodes on the route, or not. Then, there are 2^{n-x} cases of assigning nodes. Hence, in total, there are $\sum_{x=1}^{n}\binom{n}{x}\frac{(x-1)!}{2} \times 2^{n-x}$ ways to be evaluated.

Proposition 2. The TSPWPC is NP-hard.

Proof:

The TSP is NP-hard [8]. In an asymmetric TSP, there are $\frac{(n-1)!}{2}$ cycles to be evaluated to find the best Hamiltonian cycle. Complexity of the TSPWPC is $\sum_{x=1}^{n}\binom{n}{x}\frac{(x-1)!}{2} \times 2^{n-x}$, which is greater than $\frac{(n-1)!}{2}$ cycles. Therefore, the TSPWPC is NP-hard.

In the next section, we will provide an exact solution method to solve the TSPWPC problem.

3.3 Solving TSPWPC Using Miller-Tucker-Zemlin Dynamic Cuts

The TSPWPC MILP is solved using the General Algebraic Modeling System (GAMS) [26] using the CPLEX solver [27]. It is known that subtour elimination constraints inspired by the Miller-Tucker-Zemlin formulation is weaker than the general subtour elimination formulation.

Since TSP is a special case of this problem we propose a dynamic procedure to solve TSPWC inspired from work in the literature. In the dynamic cut algorithm, we propose to add the MTZ subtour elimination constraints as cuts in an iterative manner. In other words, we first solve the TSPWPC model without the subtour elimination constraint, as shown in equation (3.8). If the relaxed problem solution is feasible for the TSPWPC, then the optimal TSPWPC is found. If not, the subtour elimination constraints corresponding to the unsatisfied subtour elimination constraints

are added. In other words, if the solution is a connected graph, then it is a tour, so the problem is solved. If the solution is a disconnected graph, then each component forms a violated subtour elimination constraint. These constraints are added to the formulation as dynamic cuts, and the integer program is re-solved. The same process is repeated until one connected Hamiltonian cycle (an optimal traveling salesman tour) is found.

The procedure can be summarized as shown in Table 3.1.

Table 3.1. Dynamic Cut Procedure to Solve TSPWPC

Step	Pseudo Code of Adding Dynamic Cuts
0	Let TSPWPC without subtour elimination constraints be TSPWPC no tour.
1	Solve TSPWPC no tour MILP.
2	If all subtours in equation (8) are satisfied, then optimal is found; go to step 4.
3	Add all violated subtour elimination constraints to TSPWP no tour; go to step 1.
4	End

3.4 Genetic Algorithm with Local Optimization Strategy

The genetic algorithm [28] is an intelligent algorithm that can be applied to combinatorial optimization problems such as the TSP. This algorithm requires that the solution of the problem is encoded by chromosomes. Binary coding [29], path representation, and ordinal representation [30] are some sample types of encoding that can be used. Binary coding is not appropriate for the TSP since it magnifies its complexity [31]. The path representation can map the city's number to the genes of the chromosomes. The GA operators such as crossover and mutation can easily be applied to the chromosomes.

By setting the number of generations, population size of each generation, selection, crossover, and mutation rate parameters, a genetic algorithm finds a good solution in a reasonable

amount of time. In each generation, the chromosomes are evaluated based on their fitness function, and the best chromosomes are selected in order for the crossover and mutation to create the next generation. Obviously, the larger the population and the higher number of generations result in a more accurate approximate optimal solution.

The framework of the proposed GA is presented in Table 3.2. First, the population is initialized. In this problem, up to *n* potential cities can be on the route. Initially, all routes (paths) are generated randomly. Since this problem is a constrained TSP, one needs to check the feasibility of each path. If the generated path is infeasible, some cities are dropped from the path to achieve a feasible route.

To solve the TSPWPC efficiently using the proposed GA, the population size, selection probability, and mutation probability parameters must be assigned so that the convergence to optimality may be accelerated. The population size and number of generations have a great effect on the problem runtime. As the population size increases, the problem runtime will converge to be exponential [32]. It is also obvious that as the number of iterations increases, a better solution can be achieved, but the computational time will increase as well. So these parameters must be optimized to achieve an ideal balance between solution time and solution quality.

To calculate the fitness of each chromosome, an assignment, or optimization, problem must be solved. Chromosome selection is done based on a roulette wheel selection [33]. Then the crossover is executed, which can result in an infeasible solution; therefore, the chromosomes may be modified to obtain feasible solutions.

Finally, instead of mutation, a local optimization will be applied to the existing path to find the best tour. The local optimization is based on clustering methods, and the objective here is to

release more budget and consequently to increase the fitness by adding more cities to the route within the released budget.

Table 3.2. Genetic Algorithm with Local Optimization

Step	Pseudo Code of Proposed Genetic Algorithm
1	Set number of generations *N* and population size.
2	Set selection probability p_s, crossover probability p_c, and mutation probability p_m.
3	Generate initial population.
	3.1 Check feasibility of each chromosome.
	3.2 Modify infeasible chromosome to make it feasible.
4	While (number of iterations is less than *N*)
5	Evaluate each chromosome according to its fitness function.
	5.1 Solve assignment problem to find fitness for each path.
6	Select chromosomes according to their fitness using roulette wheel selection until pool is full.
7	Execute crossover operation
	7.1 Check feasibility of each chromosome
	7.2 Modify infeasible chromosome to make it feasible
8	Execute second level of optimization to improve path
	8.1 Find best order of cities
	8.2 Add more cities regarding their polar angles subject to released budget
9	End

Furthermore, a second GA with insertion mutation is proposed. As in each solution, there is a list of cities not served. A city which is neither traveled to nor served is inserted into the current route based on closeness of its polar angle to the average polar angle of the nodes on the route.

3.4.1 Population Initialization

A path representation is used to encode the tour. A chromosome represents a path. For example, the following chromosome is a tour of five cities: $c^1 = (5,4,0,1,2)$. This chromosome represents the following path: depot-5-4-1-2-depot. All cities that are not on the tour are assigned a value of 0 (in this case a total of four cities are not on the tour). The length of each chromosome is constant, but the number of non-zero genes may change (i.e., it is not necessary to have entire cities on the route). Each solution is presented by an array of genes. Let a new array showing the sequence of on-route nodes be defined, represented by on-route[q]. For example, the on-route array of c^1 is 5-4-1-2. Let $lengthq$ represent the length of the on-route array, and QS represent the index of nodes in on-route[q].

In initializing the chromosomes, since each chromosome represents a feasible path, the constraints of time and budget are considered as follows:

$$\sum_{q=1}^{lengthq}\left(\mathrm{t}_{on-route[q],on-route[q+1]}\right)+\mathrm{t}_{depot,on-route[1]}+\mathrm{t}_{depot,on-route[lengthq]}+\sum_{q=1}^{lengthq} tp_{on-route(q)} \leq T_{tour} \tag{3.14}$$

$$\sum_{q=1}^{lengthq} \mathrm{c}_{on-route[q],on-route[q+1]}+\mathrm{c}_{depot,on-route[1]}+\mathrm{c}_{depot,on-route[lengthq]} \leq B \tag{3.15}$$

Constraint (3.14) ensures that the tour time is less than tour time limit. Constraint (3.15) ensures that the cost of traveling is less than the budget.

3.4.2 Fitness Evaluation and Selection

In the genetic algorithm, each chromosome is evaluated using a fitness function, that is, a function of the problem objective, which is the maximum number of customers is been served, using the current solution:

$$f = \exp(obj) \tag{3.16}$$

where f is the fitness of a chromosome, and obj is the maximum number of served customers on the current tour (solution). To find the value of *obj*, the following MILP is solved:

Parameters

$$\bar{Z}_\iota = \begin{cases} 1 & if\ node\ i\ is\ on\ route \\ 0 & o.w. \end{cases}$$

Model

$$\max \sum_{i\in I} D_i \bar{Z}_\iota + \sum_{j\in I}\sum_{i\in I} \mathrm{E}_j Y_{ij} \tag{3.17}$$

Subject to

$$\bar{Z}_\iota + W_i + V_i = 1 \qquad \forall\, i \in I \tag{3.18}$$

$$\sum_{i=1}^{n} Y_{ij} = W_j \qquad \forall j \in I \tag{3.19}$$

$$\bar{Z}_\iota \geq Y_{ij} \qquad \forall i,j \in I \tag{3.20}$$

$$\sum_{j\in I} E_j Y_{ij} + D_i \bar{Z}_\iota \leq CAP_i \bar{Z}_\iota \qquad \forall i \in I \tag{3.21}$$

$$L_{ij} Y_{ij} \leq DIS_j \qquad \forall\, i,j \in I \tag{3.22}$$

Constraint (3.18) ensures that each node takes only one of three possible states: the node is on the route, the node is not on the route but is being served partially by one of the nodes on the route, or the node is not on the route and is not served indirectly. (Given a chromosome, we already know which nodes are on the route, i.e., Z_i is known.) Constraint (3.19) ensures that each node can

be served indirectly by only one of the other nodes which is on the route. Constraint (3.20) implies that if node *i* is served by node *j*, then node *j* must be on the route (otherwise it cannot serve node *i*). Constraint (3.21) restricts the number of customers served directly and indirectly in each node by the capacity of that node. Constraint (3.22) limits the distance between the serving node and the served node. Note that $\overline{Z}_\iota$ are known for each path and are fixed.

After computing the fitness of each chromosome, chromosomes should be selected to execute crossover and mutation operators. There are several methods of selection: roulette wheel, sigma scaling, Boltzmann, rank, tournament, and steady-state. Each selection has its merits and demerits [33]. In this algorithm, roulette wheel selection is applied to select the parents to cross over. The fitness function is designed such that the fittest chromosome has less than 85% of the roulette wheel, thus guaranteeing that other chromosomes have a chance to be selected as well [34]. In summary, the better the chromosome fitness, the more likely that chromosome will be selected for crossover.

3.4.3 Crossover

Executing the crossover operation will provide new offspring. There are different patterns of crossover operators: partially mapped, order, and position-based [35]. In this chapter, a modification of the partially mapped single-point crossover (PMX) is used [36]. Here, a random number less than *n* (length of chromosome, number of cities) is generated to position the crossover point for parents. Then, by exchanging two parts of the parents' chromosomes, two new offspring are produced. If the offspring is infeasible (e.g., city appears more than once in chromosome), then the offspring must be modified, in order to maintain the feasibility of the solution. In an ordinary PMX, the gene which has the replicated number in the not-exchanged part is replaced by the exchanged city. In this implementation, the repeated cities are replaced by those that are not on the

parents' route, in order to provide some randomness. To clarify this procedure, the following instance is presented. In this instance, there are five cities and the parents c_p^1 and c_p^2 are selected for crossover at step 1. At step 2, the crossover point, which is position 3, is randomly generated, and two new offspring c_{off}^1 and c_{off}^2 are generated. At step 4, the feasibility of the offspring is maintained by replacing the replicated city with a randomly selected city that does not appear in any of the parents.

Step 1:

$$c_p^1 = (5,4,0,1,2)$$

$$c_p^2 = (5,0,2,3,4)$$

Step 2:

$$c_p^1 = (5,4,0,\|1,2\|)$$

$$c_p^2 = (5,0,2,\|3,4\|)$$

Step 3:

$$c_{off}^1 = (5,4,0,\|3,4\|)$$

$$c_{off}^2 = (5,0,2,\|1,2\|)$$

Step 4:

$$c_{off}^1 = (5,4,0,\|3,1\|)$$

$$c_{off}^2 = (5,0,2,\|1,4\|)$$

After creating a feasible offspring without repeating cities, the solution represented by the offspring should be checked with respect to time limit and budget constraints. In the case of infeasibility, some cities are dropped from the current route randomly to achieve feasibility:

$$\sum_{q=1}^{lengthq}\left(\mathrm{t}_{on-route[q],on-route[q+1]}\right)+\mathrm{t}_{depot,on-route[1]}+\mathrm{t}_{depot,on-route[lengthq]}+\sum_{q=1}^{lengthq} tp_{on-route(q)} \leq T_{tour} \tag{3.23}$$

$$\sum_{q=1}^{lengthq} \mathrm{c}_{on-route[q],on-route[q+1]}+\mathrm{c}_{depot,on-route[1]}+\mathrm{c}_{depot,on-route[lengthq]} \leq B \tag{3.24}$$

3.4.4 Mutation-Local Optimization

To provide diversity to the population, the mutation operation is executed, in order to avoid trapping in a local optima. There are different mutation operators: exchange, inversion, insertion, and displacement [37]. In this chapter, as a mutation procedure, a second-level optimization strategy presented in Table 3.3 is applied.

Table 3.3. Second Level of Optimization

Steps	Pseudo Code of Second Level of Optimization
1	Fit cities into appropriate number of clusters using one of clustering methods
2	Detect cities which are in same cluster
3	Find best tour for whole clusters
4	In each cluster, sort cities according to their polar angles
5	Add more cities to tour
	5.1 Determine released budget
	5.2 If released budget is zero, then go to step 6
	5.3 For each cluster, find average polar angle
	5.4 Among detected cities, add city that has minimum difference from average polar angle of each cluster
	5.5 Check feasibility and maintain it; go to step 5
6	End

In this local optimization strategy, since the initial route is generated regardless of how cities are placed, it may be a good idea to employ clustering, and then ensure that the new tours will include cities on the same cluster, and those cities will be visited first before moving to other cities. To clarify the problem, consider the following four-step subtour of cities involving two clusters, as shown in Figures 3.1 to 3.4. As shown, neglecting the clusters can have an effect on the tour cost. Hence, by following this four-step procedure, the solution can be improved in terms of budget.

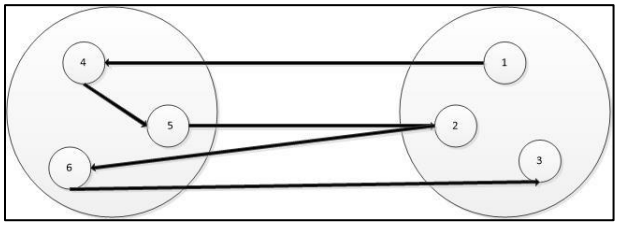


Figure 3.1. Tour before local optimization

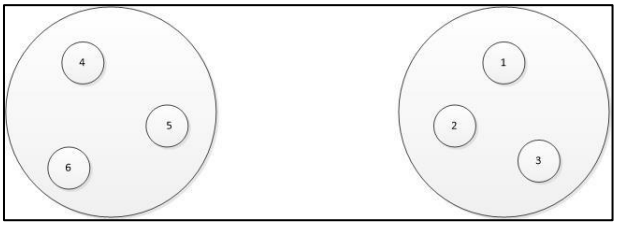


Figure 3.2. Cluster detection

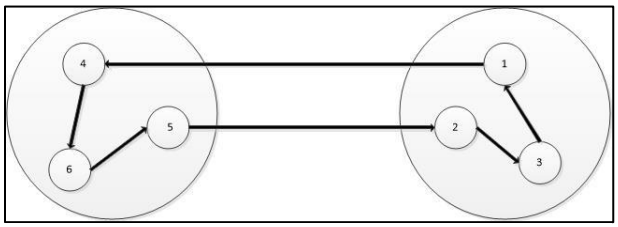


Figure 3.4. Sorting cities in each cluster among linked cities

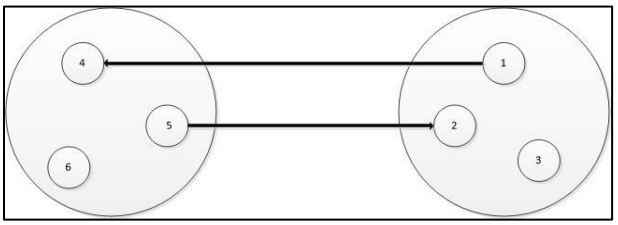


Figure 3.3. Finding best tour over clusters and finding shortcuts

To compare the GA with clustering against the GA without clustering, another GA is designed with an insertion mutation with a polar angle. To insert cities into the route, the average polar angle of on-the-route cities is calculated. Thereafter, out of the not-served cities, a city with the closest polar angle to the average one is inserted to the route. Feasibility of the solution is checked to evaluate the effect of insertion.

In this chapter, to differentiate between these two genetic algorithms, the GA with clustering based on the clustering method is denoted by GA-cluster, and the GA with an insertion mutation based on the polar angle method is denoted by GA-polar.

3.4.5 Elitism

The best chromosomes of each generation will carry over to the next generation to ensure that the best solutions of each generation are not worse than the previous generation

3.5 Computational Results and Analysis

This chapter presents two different genetic algorithm approaches. In the first approach, GA-polar, the polar angle of nodes is used to perform the insertion mutation. Here, the nodes not served are sorted based on their polar angles. Then, the nodes are added to the route based on the closeness of their polar angles to the average polar angle of on-route nodes in order. In the second approach, GA-cluster, a local optimization based on clustering methods is applied, and then cities are added based on the closeness of their polar angles to the average polar angle of the clusters. This procedure was explained in section 3.4.

3.5.1 Experimental Design

The web-based geographic midpoint calculator, Geomidpoint.com, is used to generate the problem data in order to have different distributions of potential customer locations to estimate the impact of different clustering methods on the performance of the proposed algorithms. By providing the latitude and longitude of the depot, cities are randomly generated around the depot within a certain distance, another parameter that should be provided to the software program. Then, each city is randomly generated in terms of the radius of the given distance around the depot. Moreover, the number of nodes to be generated is another parameter which is provided by the user to the software program. Figure 3.5 shows randomly dispersed customer data.

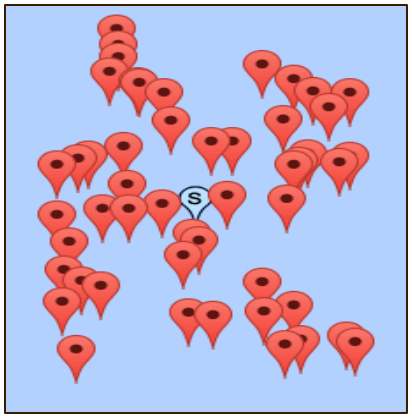


Figure 3.5. K-mean distribution/200 cities—Cluster 1

The distance between cities is calculated using the Euclidean distance measurement. In addition, demand and capacity are generated according to different distributions, in order to evaluate their impacts on the effectiveness of the proposed genetic algorithms. Both proposed GAs are coded in C programming language in order to test their performance. Moreover, each instance is solved as a mixed-integer linear program in GAMS using the CPLEX solver to provide a benchmark from which to compare the results of both GAs to the optima.

For each experiment (i.e., problem instance), the problem is solved using three different algorithms: GA-polar, GA-cluster, and the exact method by MILP (especially for small-size problems). For checking the quality of the GA-cluster and GA-polar, the best solutions obtained by these metaheuristics are compared with the exact solution found by the MILP using the optimality gap, which is calculated in terms of the relative error (RE) as follows:

$$RE = \frac{optimal\ profit - best\ profit}{optimal\ profit} \tag{3.25}$$

For larger problems, with a size less than 200 nodes, the program is executed five times using the GA-polar and the GA-cluster. The average and best solutions are recorded for each instance. The best profit obtained from these runs for each instance is shown in the best profit column.

To illustrate the performance of the GA-cluster, the best profit of some instances are depicted using GA-polar and GA-cluster approaches. In most instances, the efficiency of the GA-cluster is better than that of the GA-polar cluster. The GA-cluster, which is based on the clustering method, decreases the number of unnecessary commutes among the clusters. In well-clustered problems, the travel cost between clusters is very high, and the execution of local optimization saves considerably in terms of budget. Consequently, this results in visiting more nodes within the budget and increasing the profit. The efficiency of the GA-cluster is still better than that of the GA-polar when the nodes are randomly distributed, but the performance is not as high as in the well-clustered instances.

3.5.2 Selection of Control Parameters

Different parameters in the genetic algorithm should be tuned. The main parameters are as follows:

- Size of population (sp) implies how many chromosomes are in the population in any generation.
- Selection probability (p_s) indicates how often crossover is performed.
- Mutation probability (p_m) presents how often parts of the chromosome are mutated.
- Elitism probability (p_e) defines the rate of saving the best solutions of the previous generation in the new generation.

The probabilities are set such that the sum of p_s, p_m, and p_e are equal to one. The number of computational cycles (N_{max}) is set such that the final solution is no longer changed, and the CPU time is practical. A different set of parameters is considered, and the GAs are run for a given data set for five times, and the best combination resulting in the highest profit is selected. The set of GA control parameters are provided in Table 3.4.

Table 3.4. GA Control Parameters Set

Control Parameters	**Values**					
Size of Population (Sp)	100,50,30	100,50,30	100,50,30	100,50,30	100,50,30	100,50,30
Selection Probability (p_s)	0.7	0.65	0.6	0.55	0.50	0.50
Mutation Probability (p_m)	0.2	0.25	0.3	0.35	0.4	0.45
Elitism Probability (p_e)	0.1	0.1	0.1	0.1	0.1	0.05

For different size of the TSPWPC, the proposed GA parameters are optimized, and the optimized parameters are shown in Table 3.5. Note that for a larger problem size, the number of GA iterations increases, and the population size decreases. The selection and crossover parameters also change slightly. Furthermore, a larger problem size requires higher mutation probability.

Table 3.5. Parameters of GA with Second Level of Optimization

Scale of TSP	N_{max}	sp	p_s	p_m	p_e
$n \leq 50$	30	100	0.6	0.3	0.1
$50 < n \leq 80$	50	100	0.6	0.3	0.1
$80 < n \leq 200$	100	30	0.5	0.4	0.1

Note: N_{max}: number of computational cycles, sp: size of population, p_s: selection probability, p_m: mutation probability, p_e: elitism probability

3.5.3 Numerical Results

The following sections show an investigation of the following: (a) impact of budget amount on optimal solution; (b) impact of clustering algorithms on solution quality; (c) GA evaluation for well-fitted clustered instances; (d) impact of problem size on GA-cluster and GA-polar performance, and (e) evaluation of GAs for large-scale experiments.

3.5.3.1 Effect of Budget Amount on Optimal Solution for 32-Node Instance

For a 32-node instance, we investigate the impact of the budget amount on the optimal solution and CPU time needed by CPLEX to determine this solution. In one of our first executions,

we noted that the amount of budget affects the execution time. Our goal is to study the effect of different budget amounts on the execution time for the exact solution. First, we calculate the minimum budget required to visit all nodes as a traveling salesman problem, which also gives the maximum amount of profit. We then limit the total budget for the sensitivity analysis from 10% to 90% of this amount ($6,790). The total profit corresponding to each case is provided in Table 3.6. Furthermore, the changes in the profit while solving each problem via CPLEX using branch and bound is presented in Figure 3.6.

Table 3.6. Profit and Execution Time for Exact Method under Different Budget Amounts

Budget ($)	**Percent of Minimum Cost**	**Profit**	**Number of Directly Served Nodes**	**Number of Indirectly Served Nodes**	**Number of Not-Visited Nodes**	**CPLEX Time**
679	10	75	5	5	22	0.44
1,358	20	115	8	7	17	2.02
2,037	30	155	13	5	14	6.69
2,716	40	185	15	7	10	12.78
3,395	50	215	18	7	7	2.39
4,070	60	245	21	7	4	1.30
4,753	70	265	24	5	3	1.64
5,432	80	285	28	1	3	0.45
6,111	90	305	31	1	0	0.83
6,790	100	320	32	0	0	0.19

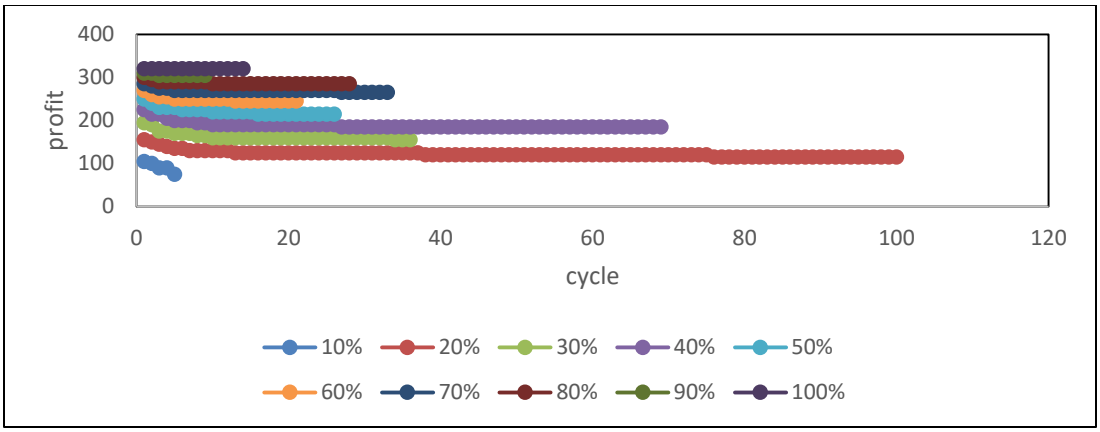


Figure 3.6. Convergence trend of different budget percentages

As can be seen in Figure. 3.7, the most difficult instance of the problem is when the budget is around 40% of the required budget for visiting all nodes. All other instances seem to have a shorter execution time, which indicates that they are easier instances to solve.

The effect of budget amount on the final solution is also studied. As shown in Figure 3.8., it can be expected that increasing the amount of the budget results in gaining more profit by virtue of having a larger feasible region. In doing so, the number of directly served nodes increases. Consequently, more nodes are served indirectly by nodes on route as well. For amounts that are more than the minimum tour cost to visit all nodes, the budget increase does not have any more impact on the final result, since the maximum profit is reached at that point.

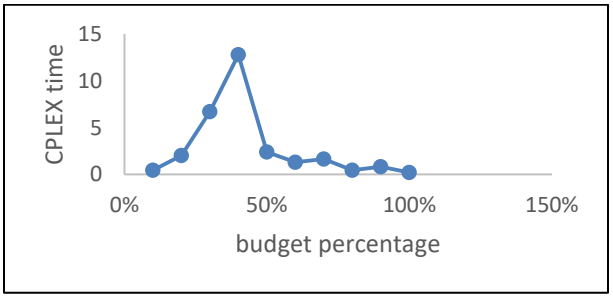


Figure 3.7. Impact of budget on CPU time required for finding optimal solution obtaining final solution percentage on CPLEX time

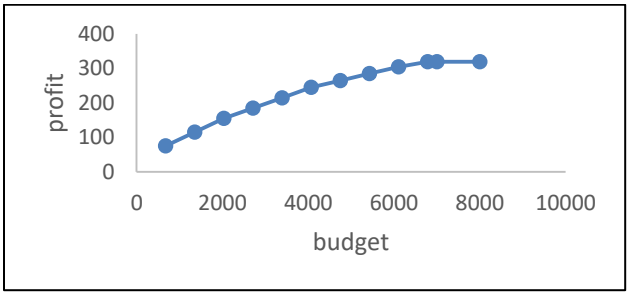


Figure 3.8. Impact of budget amount on optimal profit

3.5.3.2 Does Clustering Method Have Any Impact on Performance of Proposed Solution Method?

Clustering problems arise in many different applications, such as data mining, pattern recognition, and pattern classification. Cluster analysis is the process of classifying objects into subsets that have meaning in the context of a particular problem. In this chapter, the impact of three different clustering methods on the solution quality of the metaheuristic solution is analyzed: hierarchical clustering, k-means clustering, and k-medoids clustering.

In a k-means clustering algorithm, the objective is to minimize the squared mean between the empirical mean of a cluster and points in that cluster [38]. While in a k-medoids clustering algorithm, each cluster is represented by one of its points, and the objective is to minimize the distance between the medoid and the rest of the objects in the corresponding cluster [39]. In the hierarchical clustering algorithm, a hierarchical iterative framework is utilized to form clusters, starting from the individual elements by progressively merging the elements (i.e., clusters). To visualize the hierarchical clustering, a dendrogram, which has a tree structure, can be used [40].

The effect of different clustering algorithms is evaluated for two different size of random problem instances: The first instance includes 32 nodes and second instance includes 75 nodes.

3.5.3.2.1 Impact of Clustering Method on Solution Quality in 32-Node Instance

To evaluate the effect of different clustering methods on the GA-cluster approach, k-means, k-medoids, and hierarchical clustering methods are applied to a random 32-node instance (Figures 3.9 to 3.11). The clustering algorithms are implemented in R software [41]. In the k-means clustering method, the number of clusters is optimized based on a threshold denoted by ρ:

$$\rho = (between\ ss\ )/(total\ ss) \tag{3.26}$$

where *between ss* is the sum of squares (SS) of distances between centers of clusters to the global sample mean multiplied by the number of elements in that cluster, and *total ss* is the total sum of squares of each data point to the global sample mean. A larger ρ represents a better clustering.

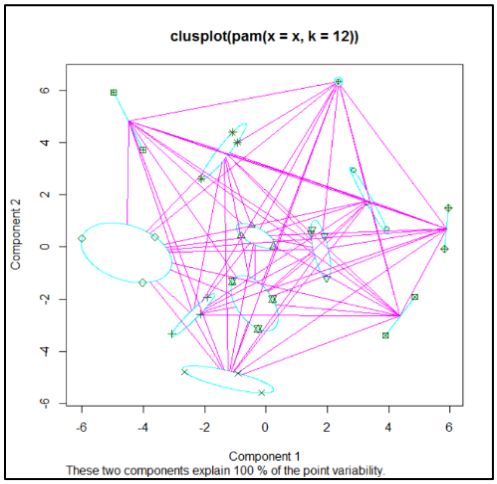


Figure 3.9. K-medoids clustering (k = 12)

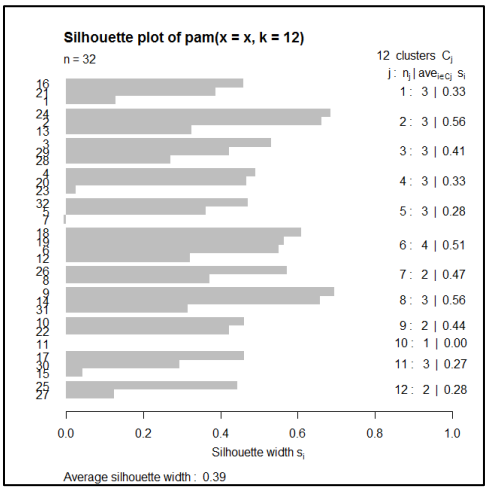


Figure 3.10. Silhouette plot-k-medoids clustering (k = 12)

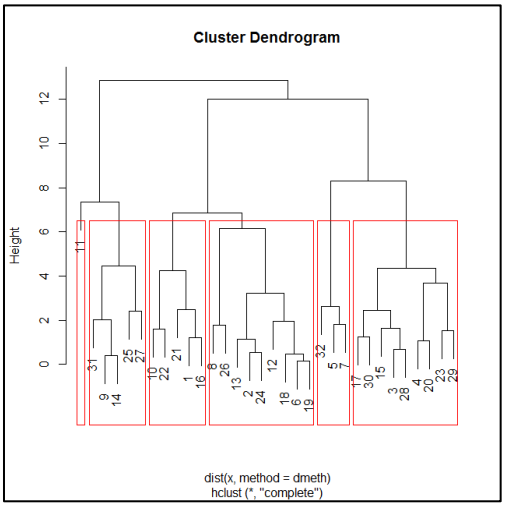


Figure 3.11. Hierarchical clustering (k = 6)

The k-means clustering method results in six clusters, whereas the k-medoids clustering method finds 12 clusters while maximizing the average silhouette width, and the hierarchical clustering method determines six major clusters. The GA-cluster method is executed using these

three different clustering methods, and the results are reported in Table 3.7. In all GA-cluster runs, the number of computational cycles is fixed to 30. When the final solution quality is compared, the k-means clustering provides a better solution in terms of profit and convergence rate.

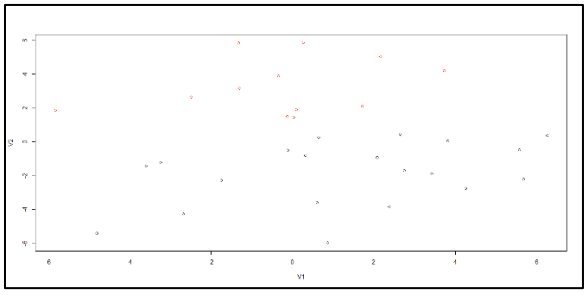
Table 3.7. Impact of Clustering Methods on Solution Quality for 32-Node Problem (Optimal Solution = 85/CPLEX Time = 22.354 Seconds)

Clustering Method	Average Profit	Best Profit	Convergence Iteration Number	RE
	GA-Cluster	GA-Cluster		
K-Means	73.4	75	4	0.11
K-Medoids	58.4	60	4	0.29
Hierarchical	60.5	75	11	0.11

In k-means clustering, the number of clusters is set to 12, whereby the highest amount of ρ is achieved. In k-medoid clustering, the number of clusters is set to 12 based on the silhouette threshold, and in hierarchical clustering, the six major clusters are chosen based on the dendrogram (Figures 3.9 to 3.11). As shown in Table 3.7, k-means clustering provides a better rate of convergence and smaller relative error.

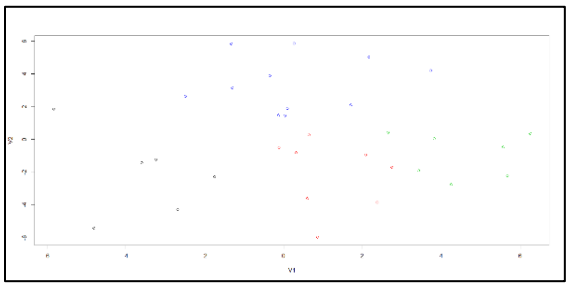
3.5.3.2.2 Impact of Number of Clusters in k-Means Clustering Method on Solution Quality for GA-Cluster Method in 32-Node Instance

To determine the number of clusters for the k-means clustering algorithm, a 32-node instance is used. Clustered instances are shown in Figures 3.12(a) to 3.12(e).

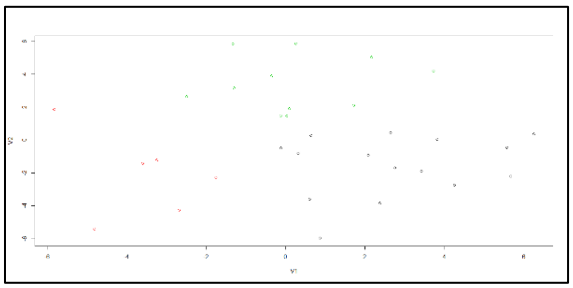


(a) k-means clustering (k = 2)

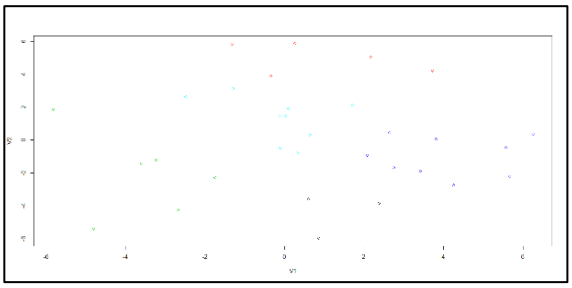
Figure 3.12. Different k-means clustered instances



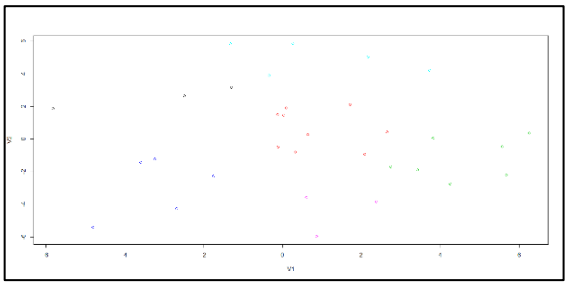
(b) k-means clustering (k = 4)



(c) k-means clustering (k = 3)



(d) k-means clustering (k = 5)



(e) k-means clustering (k = 6)

Figure 3.12. (continued)

Generally, when the number of clusters increases, the ρ percentage increases, and the average and best profit improve (Table 3.8). The best number of clusters is six, based on the average profit, the best profit, and the number of iterations/populations it takes to converge. In this instance, the average result of a two-cluster case is close to the average result of the best scenario, and its best profit is the same as the best scenario. However, the six-cluster case converges to the optimal solution much faster. In the two-means clustering case, it takes 14 iterations, whereas in the six-means clustering case, it only takes four iterations. Figure 3.13 shows the detailed convergence behavior of all analyzed cases.

Table 3.8. Quality of GA-Cluster Solution Based on Different Number of Clusters for 32-Node Instance (Optimal Solution = 85/CPLEX Time = 22.354 Seconds)

Number of Clusters	ρ (%)	Average Profit	Best Profit	Convergence Iteration Number	Re
		GA-Cluster	GA-Cluster		
2	37.9	68.3	75	14	0.11
3	63.9	58.4	60	4	0.29
4	72.0	59.4	60	2	0.29
5	78.7	59.4	75	14	0.11
6	83.7	73.4	75	4	0.11

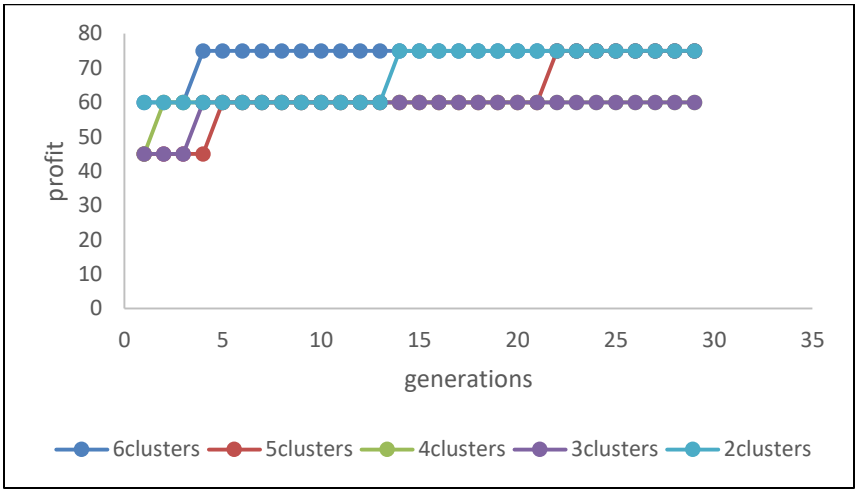


Figure 3.13. Convergence trend of different number of clusters (RA-32)

3.5.3.2.3 Impact of Clustering Method on Solution Quality in 75-Node Instance

The GA cluster method is applied to a 75-node randomly generated problem to analyze the impact of different clustering approaches on the solution quality. Figures 3.14 to 3.16 show the analysis for k-means, k-medoids, and hierarchical clustering methods, respectively. In the k-means clustering method, the number of clusters, based on the ρ threshold, is seven. In the k-medoids clustering method, the number of clusters maximizing the average silhouette width equals six, and in the hierarchical clustering method, seven major clusters are formed. The quality of solutions

obtained for the GA-cluster method is presented in Table 3.9. The maximum number of computational cycles is fixed to 50. However, in all cases, the GA-cluster either found the optimal or could not improve its current solution before 25 iterations. Furthermore, the GA-cluster found the best results (profit) using the k-means clustering approach. After 25 iterations (generations), the GA-cluster found the optimal solution (i.e., Re is 0) using this approach. It seems that the k-medoids algorithm is the second best clustering approach when the GA-cluster is utilized.

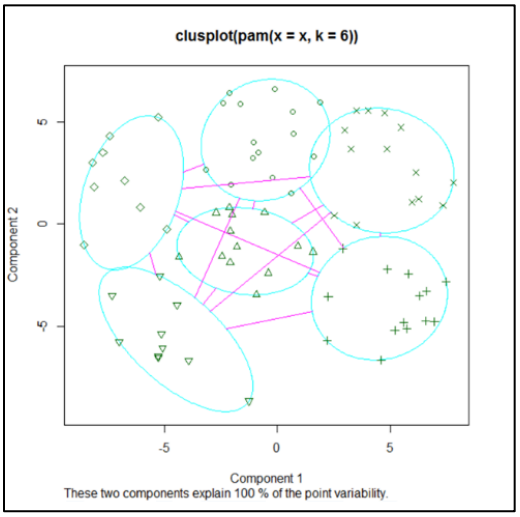


Figure 3.14. K- medoid clustering (k = 6)

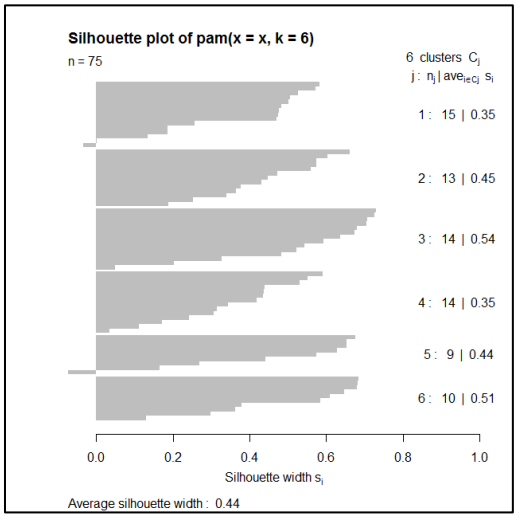


Figure 3.15. Silhouette plot-k-medoid clustering (k = 6)

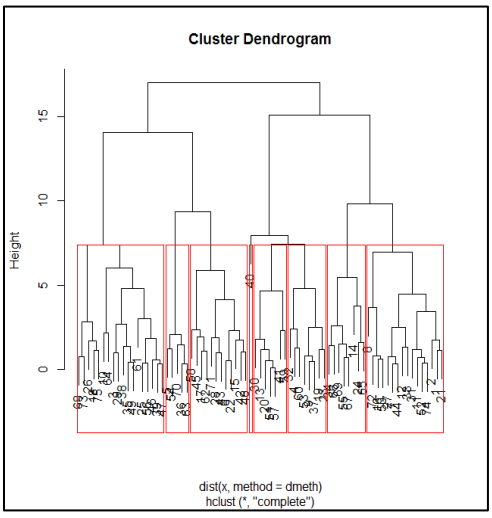


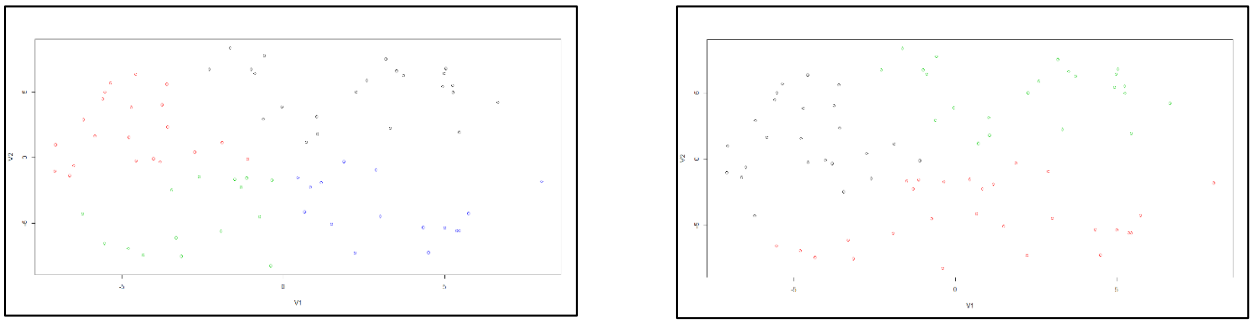
Figure 3.16. Hierarchical clustering (k = 7)

Table 3.9. Results of Different Clustering Methods in 75-Node Instance
(CPLEX Time = 34.934 Seconds)

Clustering Method	Average Profit	Best Profit	Cycle Number Convergence	Optimal Profit	RE (%)
	GA-Cluster	GA-Cluster			
K-means	77	90	25	90	0
K-medoids	71	75	8	90	0.17
Hierarchical	63	75	15	90	0.17

3.5.3.2.4 Impact of Number of Clusters in k-Means Clustering Method on Solution Quality for GA-Cluster Algorithm in 75-Node Instance

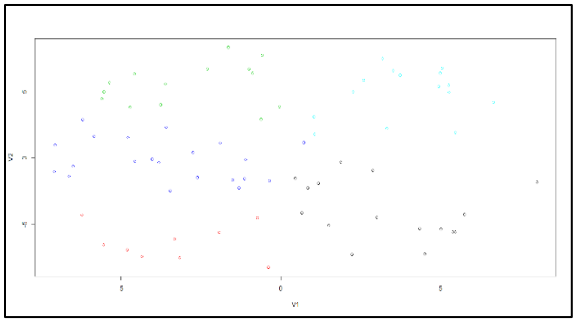
To determine the impact of number of clusters generated by the k-means algorithm on the solution quality of the GA-cluster, a 75-node instance is solved, as shown in Figures 3.17(a) to 3.17(g), and 3.18. Table 3.10 shows that as the number of clusters increase, the ρ percentage and the average profit increase, which is similar to the 32-node instance, and the best profit obtained by the GA-cluster is the same value (75 with an Re of 17%) until ρ becomes 88.6% and k is eight. When there are seven clusters, the optimal solution can be found. However, with more clusters, the solution deteriorates, which may be due to having more interaction among the clusters and reducing diversity in the mutation. Results are shown in Table 3.10 and Figure 3.18.



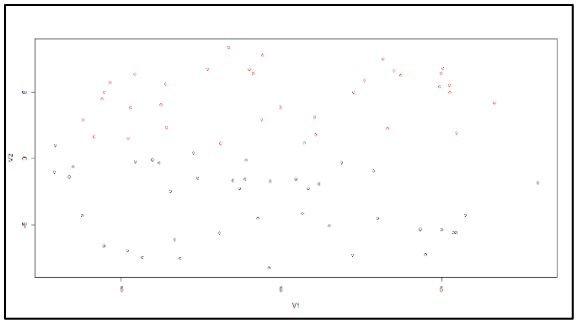
(a) k-means clustering (k = 2)

(b) k-means clustering (k = 3)

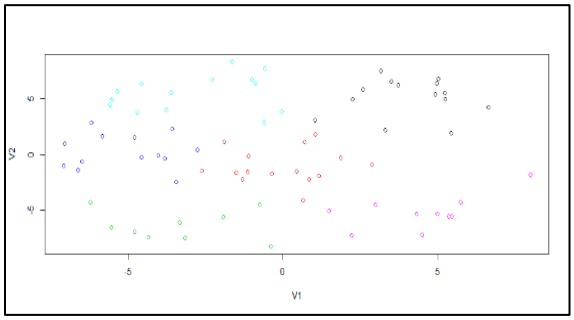
Figure 3.17. Different k-means clustered instances



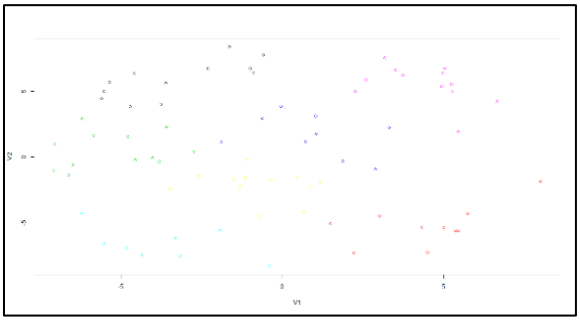
(c) k-means clustering (k = 4)



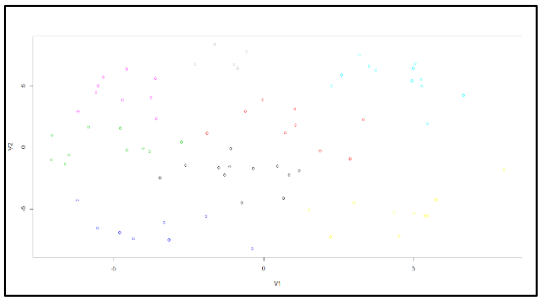
(d) k-means clustering (k = 5)



(e) k-means clustering (k = 6)



(f) k-means clustering (k = 7)



(g) k-means clustering (k = 8)

Figure 3.17. (continued)

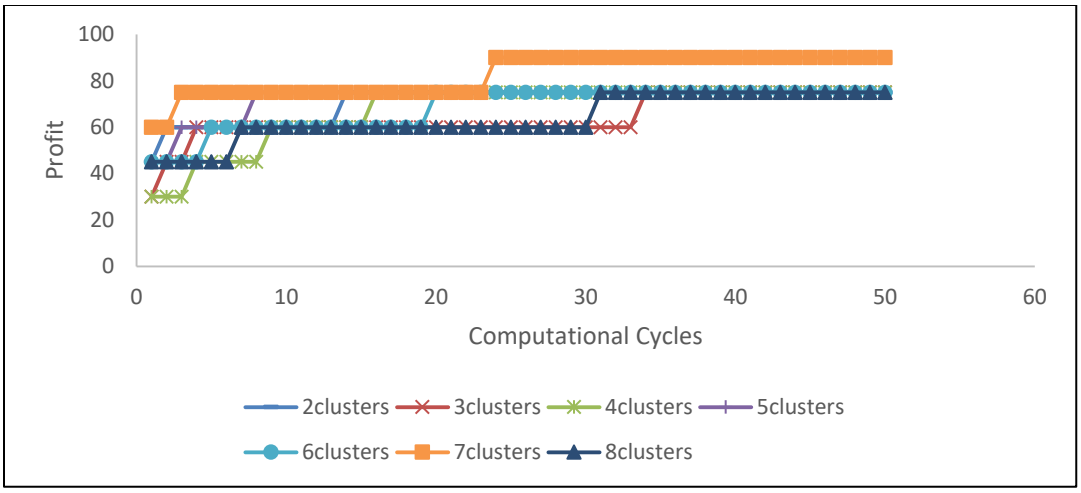


Figure 3.18. Convergence trend of different numbers of clusters (RA-75)

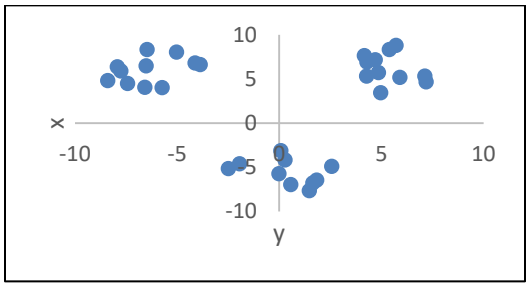
Table 3.10. Results of GA-Cluster for Different-Size Clusters of K-Means in 75-Node Instance (CPLEX Time=34.934 Seconds)

Number of Clusters	ρ (%)	Average Profit	Best Profit	Convergence Iteration Number	Optimal Profit	Re
		GA-Cluster	GA-Cluster			
2	42.7	70.8	75	14	90	0.17
3	64.9	63.9	75	34	90	0.17
4	74.6	67.2	75	16	90	0.17
5	81.0	72.3	75	8	90	0.17
6	86.3	68.1	75	20	90	0.17
7	88.6	77	90	25	90	0
8	90.4	64.2	75	31	90	0.17

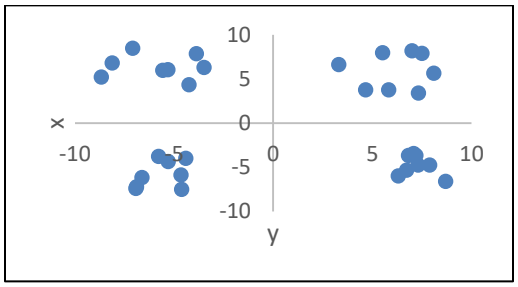
3.5.3.3 GA Evaluation for Well-Fitted Clustered Instances

To evaluate the performance of the proposed GA methods, eight different distributions of nodes are generated where all clusters are well separated, as shown in Figure 3.19. Furthermore, the demand distribution of each instance is varied to analyze the sensitivity of algorithms to the demand distribution. Both GA-polar and GA-cluster approaches are run five times to check the impact of demand distribution variations. The average profit and best profit of these runs are

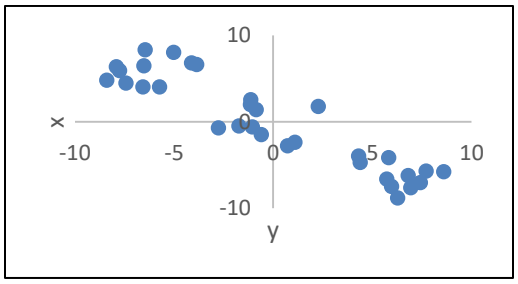
presented in Table 3.11. To calculate the relative error from the optimal solution, mixed-integer linear program is solved using GAMS/ CPLEX solver. It is also found that both GAs are sensitive to the demand distribution: when demand increases in some clusters, more visits to cities in that cluster are made, resulting in higher profit.



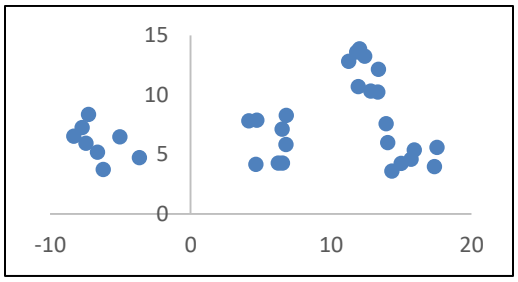
(a) Km1-3-u/d distribution graph



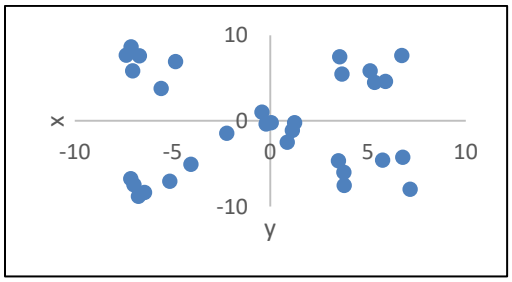
(b) Km1-4-u/d distribution graph



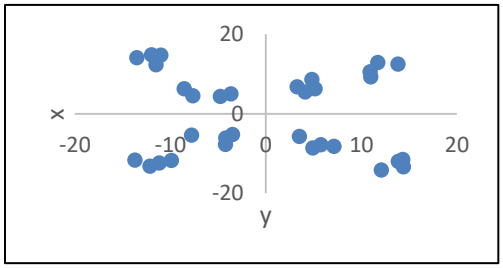
(c) Km2-3-u/d distribution graph



(d) Km2-4-u/d distribution graph



(1) Km-5-u/d distribution graph



(f) Km-8-u/d distribution graph

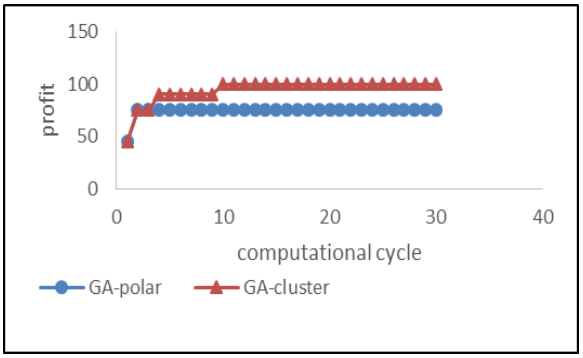
Figure 3.19. Different distributed instances based on k-means clustering algorithm

Table 3.11. Results of GA Approaches for Well-Separated Clustered Instances

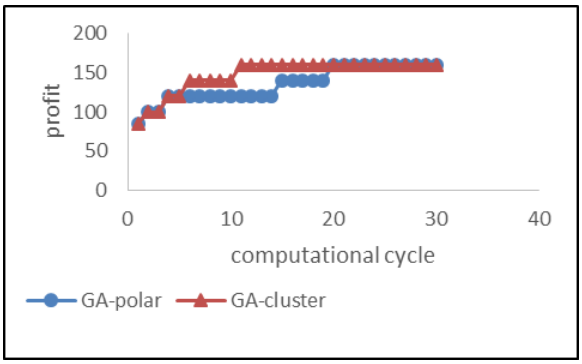
Name	Average Profit		Best Profit		Optimal Profit	RE (%)		CPLEX Time
	GA-Polar	GA-Cluster	GA-Polar	GA-Cluster		GA-Polar	GA-Cluster	
Km1-4-u	78	94.65	90	105	135	0.33	0.22	124.683
Km-5-u	76.67	121	90	150	175	0.48	0.16	712.504
Km-8-u	30	53	30	60	60	0.50	0	64.654
Km2-4-u	74	94.5	75	100	100	0.25	0	894.375
Km1-3-u	43.5	45	45	45	45	0	0	18.060
Km2-3-u	76	100	90	120	120	0.25	0	619.116
Km1-4-d	140	146.67	150	150	190	0.21	0.21	497.037
Km-5-d	206.67	227.5	225	250	260	0.10	0.03	152.409
Km-8-d	85.53	100	100	100	100	0	0	77.996
Km2-4-d	135.5	147.5	160	160	160	0	0	67.01
Km1-3-d	73.3	73.5	75	75	75	0	0	19.548
Km2-3-d	137.5	150.83	150	175	175	0.14	0	45.752

For each instance of the problem, demand is uniformly and manually distributed. For example, km1-4-u represents an instance generated by k-means clustering algorithm with four clusters and a uniform demand distribution over the nodes, whereas in km1-4-d, demand distribution is higher for some clusters, as shown previously in Figure 3.17(b). The average profit and best profit for each instance are calculated for both GAs. Table 3.10 shows that the GA-cluster has a considerably better performance based on the relative error.

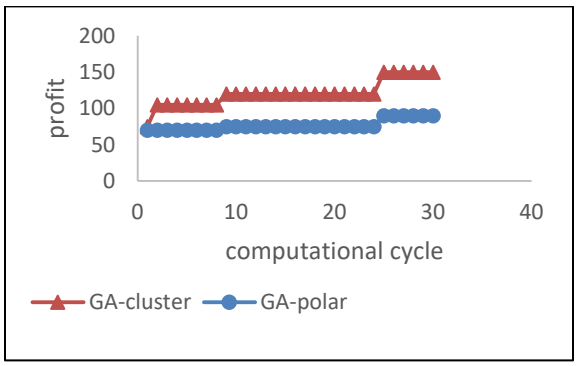
Profit changes as a function of the number of iterations for both GA approaches, as depicted in Figure 3.20. It is clear that the profit of the GA-cluster converges faster and is higher than the GA-polar approach. In addition, in the case where the clusters can be identified optimally, the GA-cluster approach finds the optimal solution more quickly.



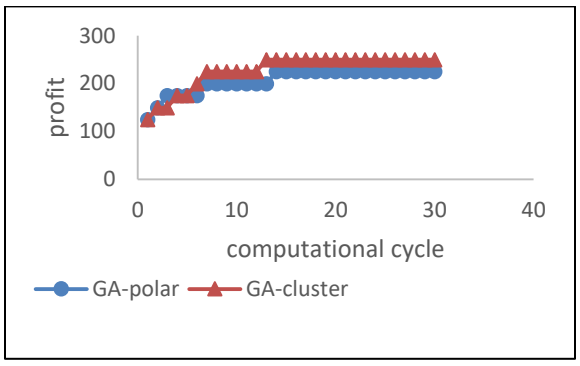
(a) Convergence trend of km2-4-u



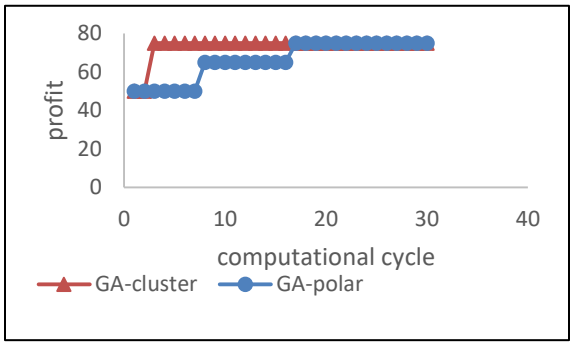
(b) Convergence trend of km2-4-d



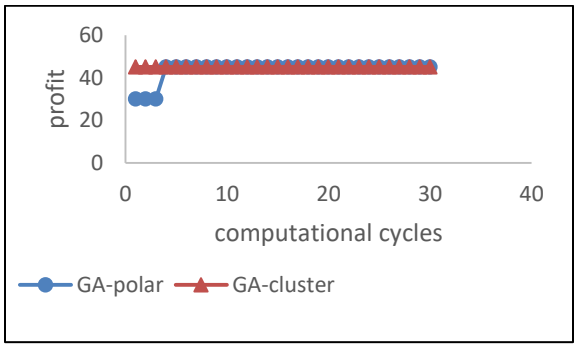
(c) Convergence trend of km-5-u



(d) Convergence trend of km-5-d

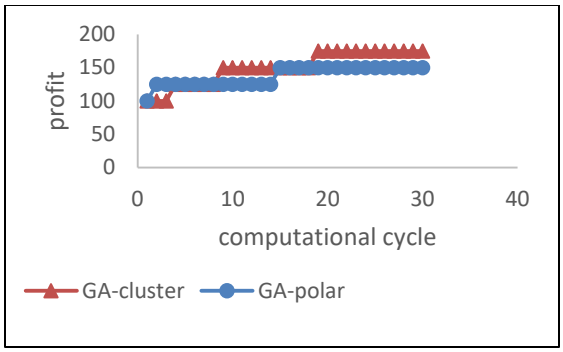


(e) Convergence trend of km1-3-u

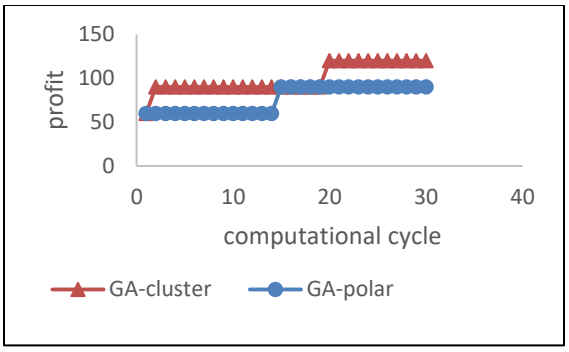


(f) Convergence trend of km1-3-d

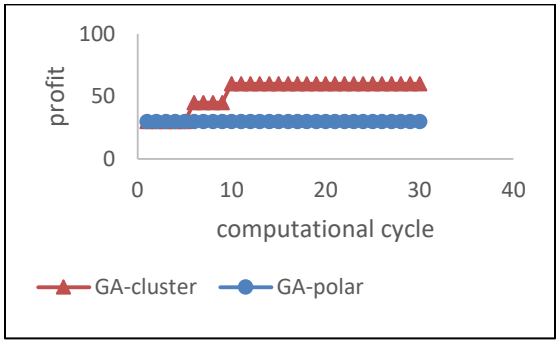
Figure 3.20. Convergence trend of well-clustered instances



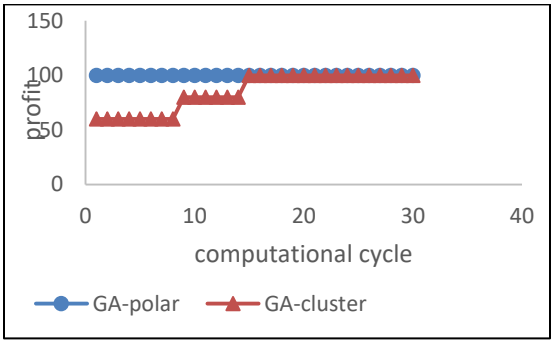
(g) Convergence trend of km2-3-u



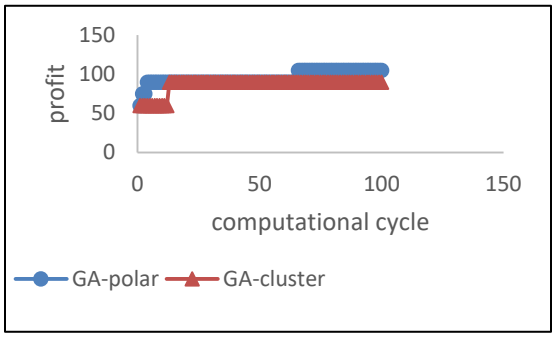
(h) Convergence trend of km2-3-d



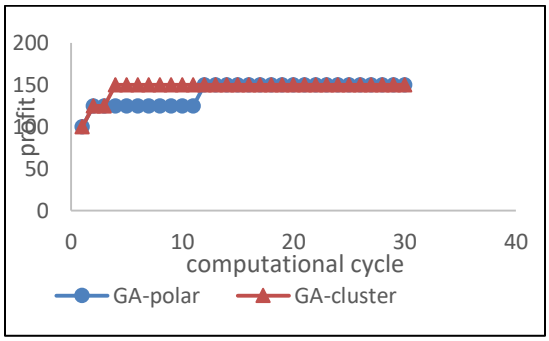
(i) Convergence trend of km-8-u



(j) Convergence trend of km-8-d



(k) Convergence trend of km1-4-u



(l) Convergence trend of km1-4-d

Figure 3.20. (continued)

We also analyze how absolute error (i.e., difference between the optimal solution and the GA solution). Changes as a function of iteration count (computational cycles) are shown in Figures 3.21 to 3.23, which indicate that the GA-cluster decreases the absolute error more quickly than the GA-polar, and in most of the cases, the error reduces to zero (i.e., optimal solution is found).

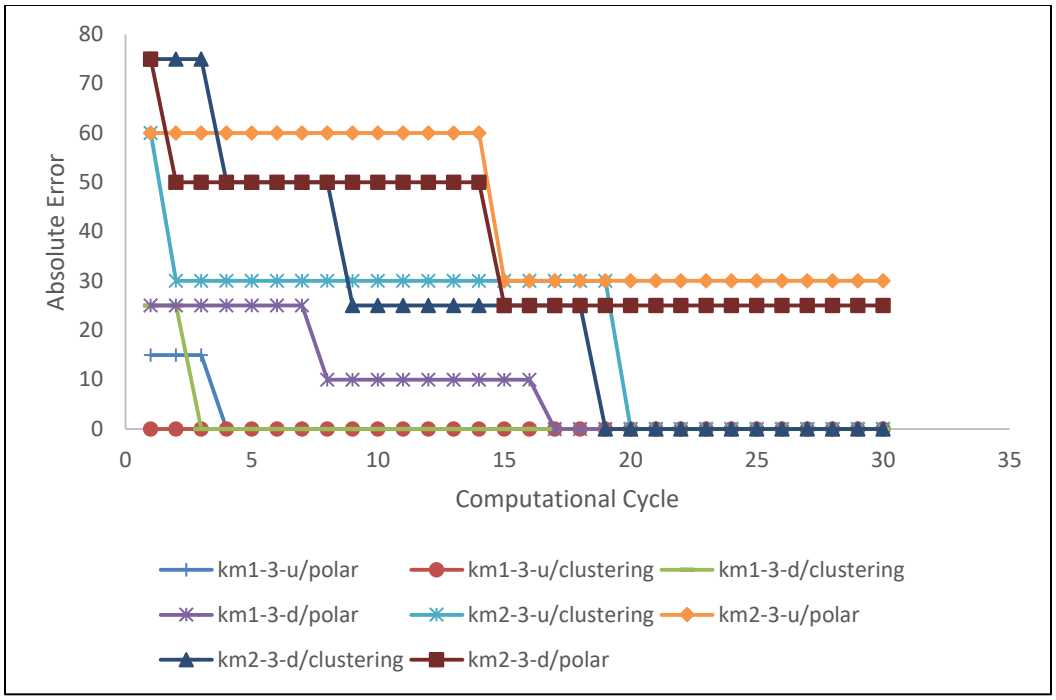


Figure 3.21. Change of absolute error of km1-3-u/d and km2-3-u/d using GA-polar and GA-clustering approaches

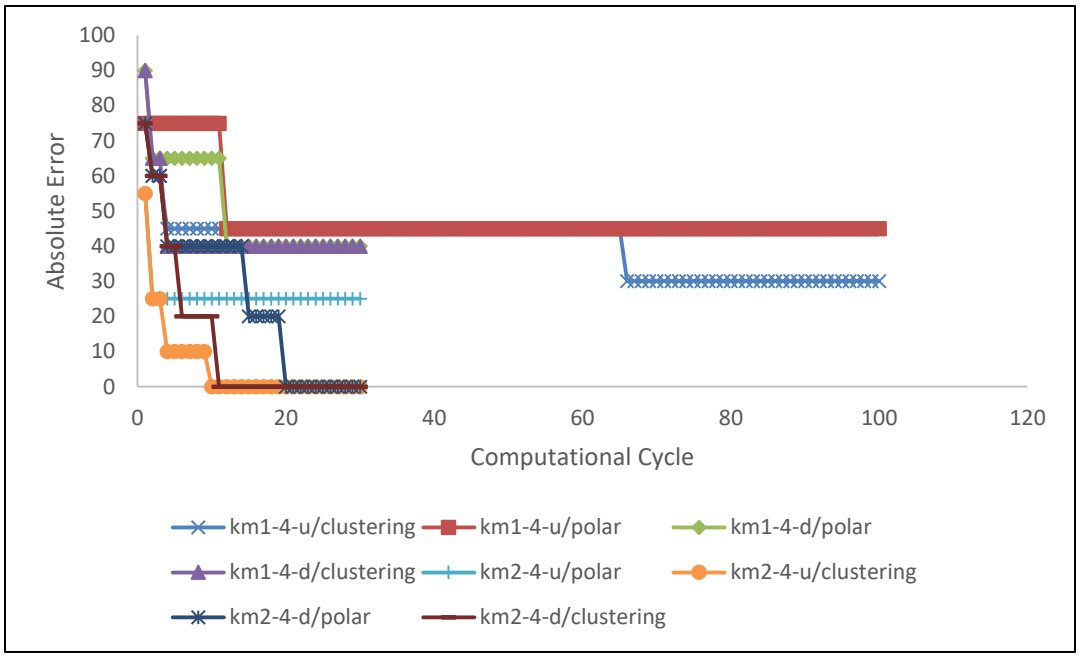


Figure 3.22. Change of absolute error of km1-4-u/d and km2-4-u/d using GA-polar and GA-clustering approaches

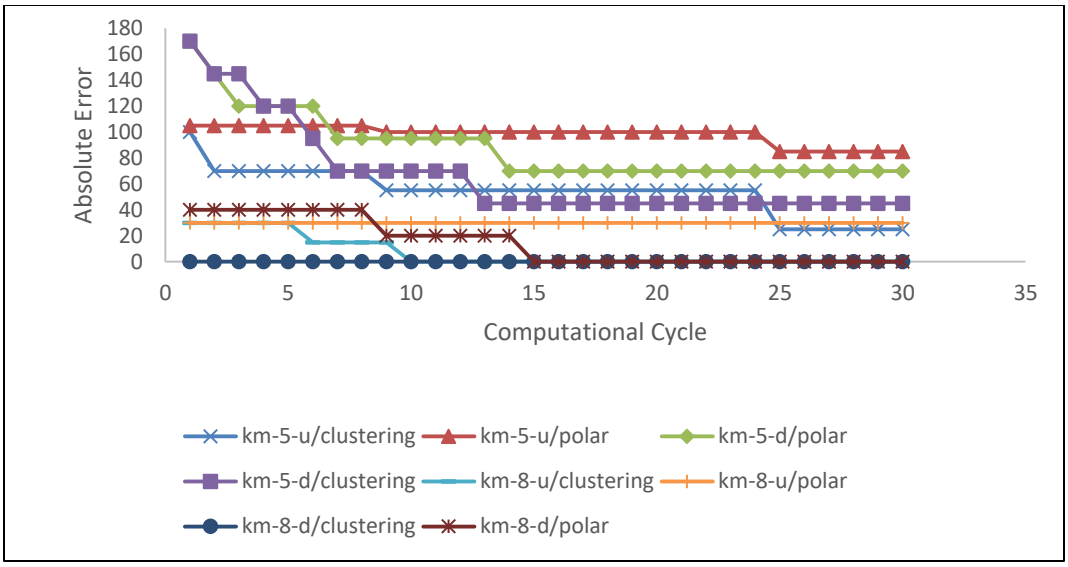


Figure 3.23. Change of absolute error of km-5-u/d and km-8-u/d using GA-polar and GA-cluster approaches

3.5.3.4 Impact of Problem Size on GA-Cluster and GA-Polar Performance

To analyze the efficiency of GA-cluster and GA-polar, eleven different-size randomly dispersed instances are generated. The results are summarized in Table 3.12, which provides the average and best profit for both genetic algorithms. In order to calculate the optimality gap, each instance of the problem, which is a mixed-integer linear program, is solved to optimality using CPLEX.

In the GA-cluster approach, first, clusters are defined using the k-means clustering algorithm executed in R (see Figures 3.24 and 3.25). For larger-size problems, GAs are run for a greater number of iterations. In all of the experiments, the GA-cluster is giving better results, although results are quite close to those of the GA-polar in some instances.

Table 3.12. Results of GA Approaches for Large-Size Instances

Instance	Average Profit		Best Profit		Optimal Profit	RE		CPLEX Time
	GA-Polar	GA-Cluster	GA-Polar	GA-Cluster		GA-Polar	GA-Cluster	
RA-50	68	82	75	105	115	0.35	0.087	985.71
RA-75	45	82.5	45	90	90	0.50	0	34.934
RA-80	58.5	50.5	60	60	60	0	0	112.11
RA-90	46.15	55	56.8	65	65	0	0	325.09
RA-100	56	60.9	60	75	80	0.25	0.0625	192.802
RA-120	60	47.1	60	60	70	0.14	0.14	419.68
RA-130	48.9	71.25	60	90	150	0.6	0.4	7,981.828
RA-140	59.1	74.4	75	90	100	0.25	0.1	11,630.729
RA-150	58.2	59.1	60	60	75	0.2	0.2	157,056.325
RA-170	70.05	80.1	75	90	105	0.28	0.14	168,712.504

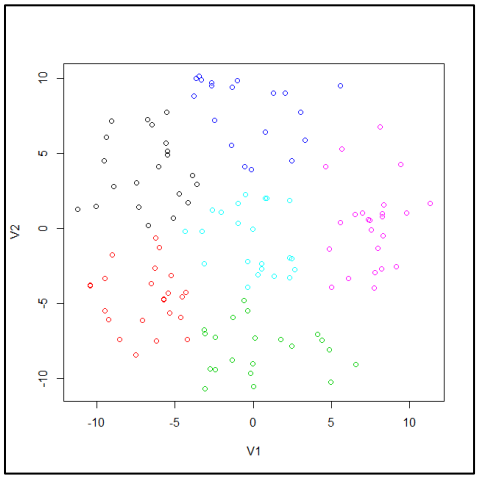


Figure 3.24. K-means clustering for RA-130

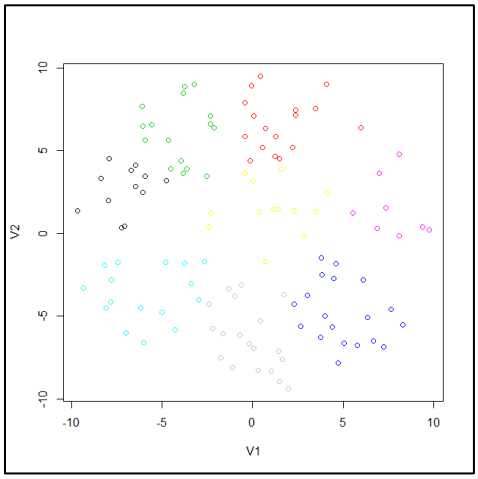
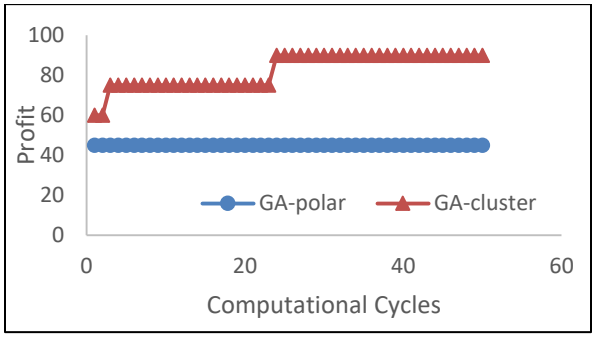


Figure 3.25. K-means clustering for RA-120

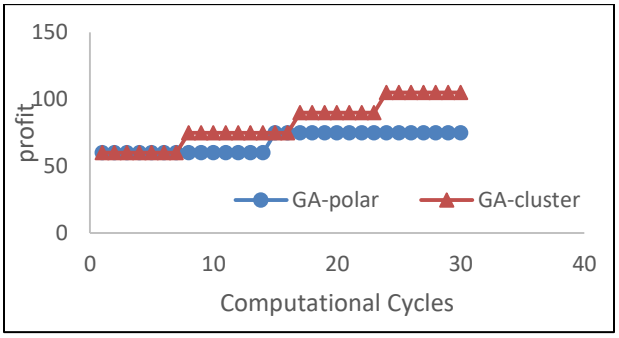
As shown, for most cases, the GA-cluster performs better because its relative error is less than the relative error of the GA-polar. It should be mentioned that the profit is in terms of served demand, which is a discrete amount. For most cases, the difference between solutions of CPLEX

and GA-cluster in terms of visited nodes is just one node (i.e., RA-140, RA-150, and RA-170). Hence, the performance of the GA-cluster is fairly good since it is short of only one visited node.

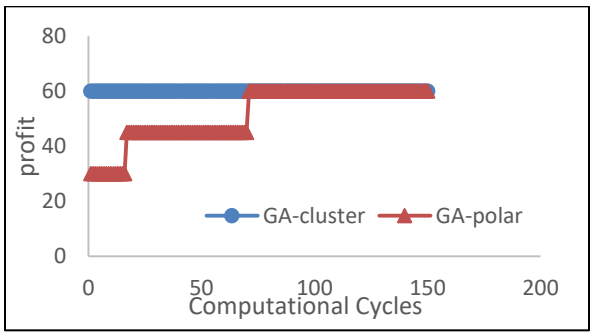
Figures 3.26(a) to 3.26(j) show that GA-cluster is better than GA-polar in finding a better solution closer to the optimal solution. In RA-80 and RA-120 instances, the convergence of the GA-cluster is considerably high. However, in RA-80, RA-90, and RA-150 instances, both of the proposed genetic algorithms converge to the same solution. The GA-cluster solution is usually better than the GA-polar solution over the same number of generations.



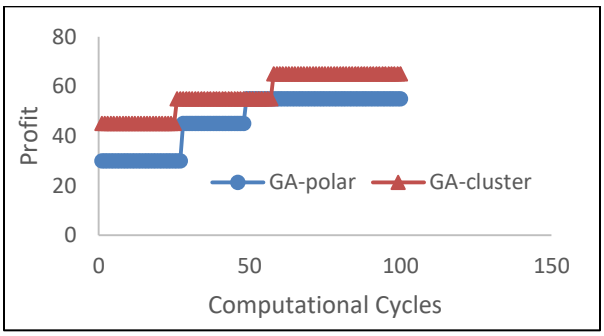
(a) RA-50



(b) RA-75

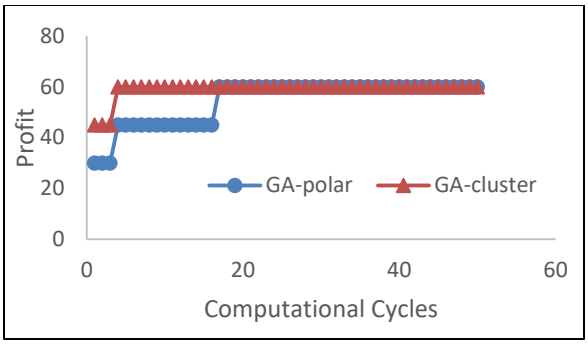


(c) RA-80

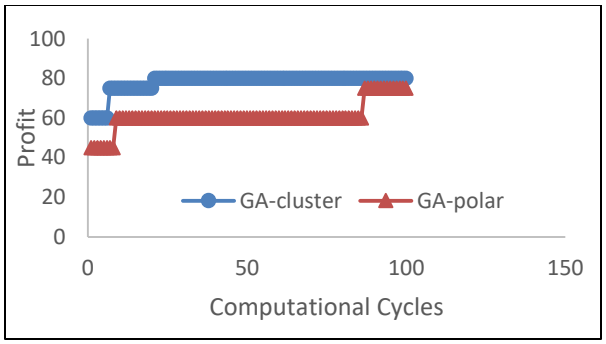


(d) RA-90

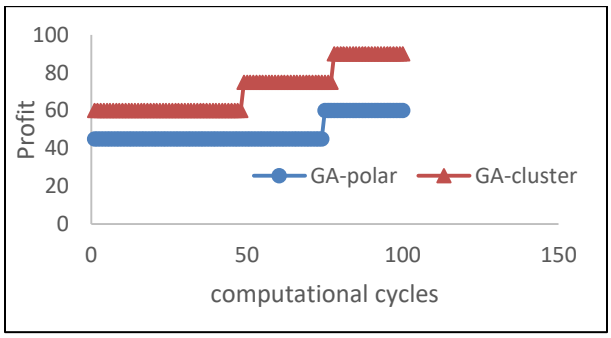
Figure 3.26. Profit changes trend over different generations



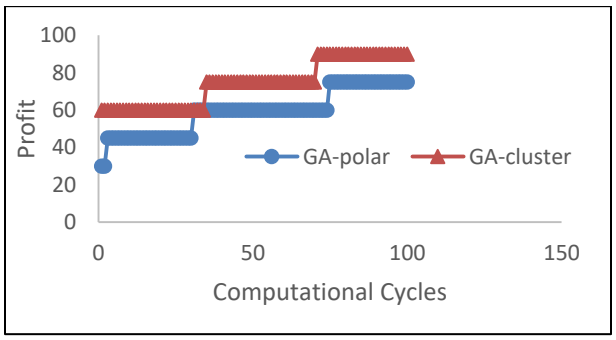
(e) RA-100



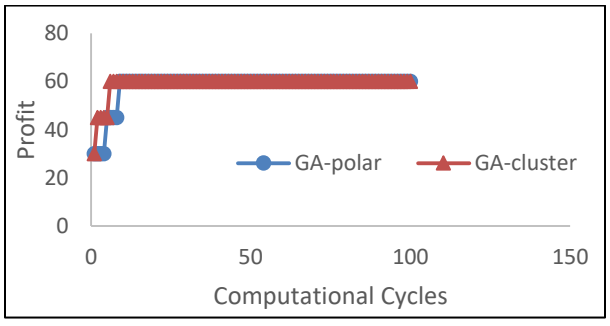
(f) RA-120



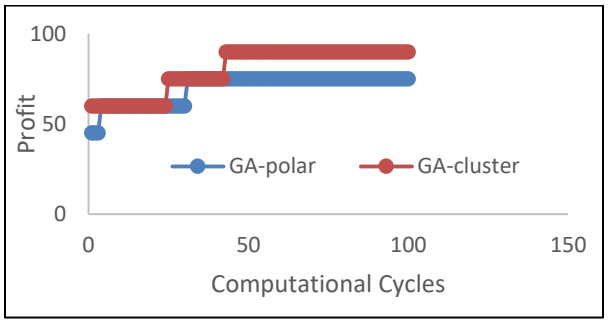
(g) RA-130



(h) RA-140



(i) RA-150



(j) RA-170

Figure 3.26. (continued)

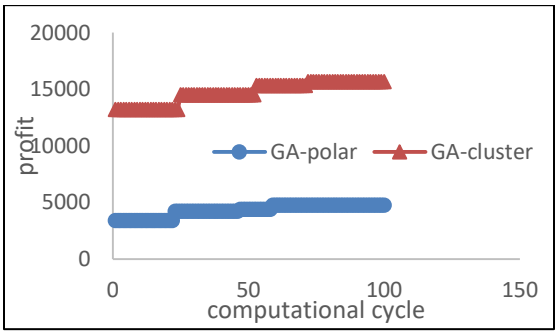
3.5.3.5 Evaluation of GAs for Large-Scale Experiments

We also run the proposed GA approaches for large-scale problems when we were not able to find an exact solution using the CPLEX MILP solver. It is found that the GA-cluster outperforms the GA-polar for larger problems, especially when the clusters are generated by the k-means clustering method. Furthermore, for randomly generated instances, although the GA-cluster still

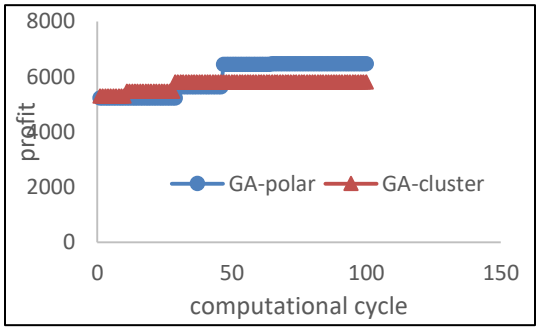
outperforms the GA-polar, the difference is not as significant as k-means clustered instances, as shown in Table 3.13 and Figures 3.27(a) to 3.27(d).

Table 3.13. Results of GA Approaches for Large-Scale Instances

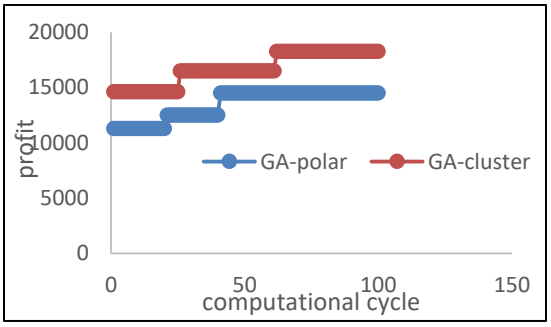
Cities Distribution	Demand and Capacity Distribution	Size	Average Profit		Best Profit	
			GA-Polar	GA-Cluster	GA-Polar	GA-Cluster
K-means	Uniform	200	4460	14784	4786	15672
Uniform	Uniform	200	5745	5516.67	6470	5810
K-means	Random	200	2612	3195	3440	4530
Uniform	Random	200	5499.3	6009.67	5865	6344
K-means	Uniform	500	13488	16742	14530	18280
Uniform	Uniform	500	22504	22734	22756	22830



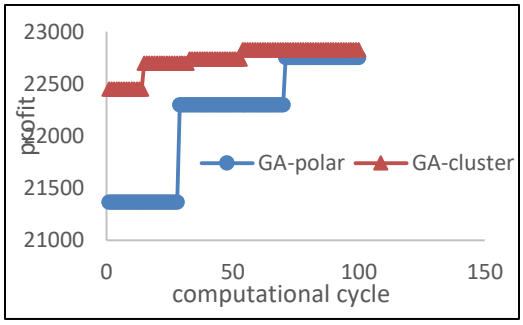
(a) K-Means/200 nodes



(b) Uniform/200 nodes



(c) K-means/500 nodes



(d) Uniform/500 nodes

Figure 3.27. Trend in profit changes over different generations for large-scale instances

The GA-cluster is giving significantly better results than the GA-polar, where the instances are generated by the k-means algorithm. When the instances are generated uniformly, the GA-cluster is still better than the GA-polar in all instances except for the uniform/200-node instance, as shown in Figure 3.27(b). This may be due to the decreasing diversity of the pool, since clusters do not have that much effect on local optimization.

The GA-cluster approach is always providing a better solution, given that the data has some clusters. Although, the solution of the GA-polar approach is as good as the solution of the GA-cluster approach, in some cases, the GA-cluster converges faster (i.e., in a fewer number of iterations and more efficiently).

Different clustering methods have different effects on the quality of the final result. In both the 32-node and 75-node instances, each clustering method results in a different solution in terms of profit and the rate of convergence (see sections 3.5.2.1 and 3.5.2.3). Among the clustering methods, the k-means clustering method is working better with the proposed GAs. Furthermore, the number of clusters in the k-means clustering method has a major effect on the GA-cluster performance. The number of clusters optimized with respect to ρ provides better results.

Another observation is that the MILP execution time of an instance is variable due to the budget and tour duration amounts. If these are set to very large or very small, then the execution time would be quite a bit lower, compared to the case where the budget and time are constrained at a medium range. In the first case, most of the cities can be on the route, whereas in the second case, only very few cities would be on the route.

3.6 Conclusion

A new traveling salesman problem with partial coverage, the TSPWPC, is introduced in this chapter. Due to the complexity of the problem, two genetic algorithms are designed. The

genetic algorithm with local optimization based on clustering provides good solutions for the TSPWPC in a reasonable amount of time. If the nodes are clustered, then preprocessing with clustering optimization helps the GA-cluster method to find good solutions: clustering optimization can be done with respect to the number of clusters and also which clustering method is utilized. It is observed that when the number of clusters is optimized, the convergence rate increases, and a closer solution to optimal is found. Furthermore, among the different clustering algorithms, the k-means clustering provides a better separation of nodes; hence, the result of the GA-cluster improves faster.

Since the complexity of the problem is o $(\sum_{x=1}^{n}\binom{n}{x}\frac{(x-1)!}{2}\times 2^{n-x})$, the MILP cannot find the solution for large-scale problems in a reasonable amount of time.

Parameters for the GAs play an important role in their performance. Generally, larger parameters provide better solutions. For instance, with a larger population size and iteration number, a better solution can be obtained. However, for large-scale problems, a large population size implies a greater computational time. For this reason, the population size and number of iterations must be optimized with respect to an allowable computation time.

It is also observed that the budget and time constraint affect the solution time. Hence, to measure the efficiency of the GA approaches, we chose problem instances, which were more difficult.

In the future, this problem can be extended to different cases where there are multiple depots, multiple capacitated vehicles, and multiple travelers. Time windows and uncertainty could be considered for each problem as well. Some of these cases are addressed in the following chapters.

3.7 References

[1] Lupsa, L., Chiorean, I., Neamtiu, L., & Lupsa, R. (2010). Some special traveling salesman problems with applications in health economics. INTECH Open Access Publisher.

[2] Laporte, G., & Palekar, U. (2002). Some applications of the clustered travelling salesman problem. *Journal of the Operational Research Society*, *53*(9), 972-976.

[3] Arulselvan, A. (2009). Network model for disaster management (Doctoral dissertation). University of Florida, Gainesville.

[4] Beasley, J. E., & Nascimento, E. M. (1996). The vehicle routing-allocation problem: A unifying framework. *Top*, *4*(1), 65-86.

[5] Current, J. R., & Schilling, D. A. (1989). The covering salesman problem. *Transportation Science*, *23*(3), 208-213.

[6] Arkin, E. M., & Hassin, R. (1994). Approximation algorithms for the geometric covering salesman problem. *Discrete Applied Mathematics*, *55*(3), 197-218.

[7] Gendreau, M., Laporte, G., & Semet, F. (1997). The covering tour problem. *Operations Research*, *45*(4), 568-576.

[8] Hachicha, M., Hodgson, M. J., Laporte, G., & Semet, F. (2000). Heuristics for the multi-vehicle covering tour problem. *Computers & Operations Research*, *27*(1), 29-42.

[9] Labbé, M., Laporte, G., Rodríguez Martín, I., & González, J. J. S. (2005). Locating median cycles in networks. *European Journal of Operational Research*, *160*(2), 457-470.

[10] Archetti, C., Hertz, A., & Speranza, M. G. (2007). Metaheuristics for the team orienteering problem. *Journal of Heuristics*, *13*(1), 49-76.

[11] Feillet, D., Dejax, P., & Gendreau, M. (2005). Traveling salesman problems with profits. *Transportation Science*, *39*(2), 188-205.

[12] Zhang, W. X., & Korf, R. E. (1996). A study of complexity transitions on the asymmetric traveling salesman problem. *Artificial Intelligence*, *81*(1–2), 223–239.

[13] Climer, S., & Zhang, W. X. (2006). Cut-and-solve: An iterative search strategy for combinatorial optimization problems. *Artificial Intelligence*, *170*(8–9), 714–738

[14] Johnson, D. S., & McGeoch, L. A. (2004). *The traveling salesman problem and its variations*, *combinatorial optimization*. London: Springer Press, pp. 445–487.

[15] Burke, E. K., Cowling, P. I., & Keuthen, R. (2001). Effective local and guided variable neighbourhood search methods for the asymmetric travelling salesman problem. In *Applications of evolutionary computing* (pp. 203-212). Berlin Heidelberg: Springer.

[16] Munakata, T., & Nakamura, Y. (2001). Temperature control for simulated annealing. *Physical Review E*, 64, 046127.

[17] M. Clerc, Discrete particle swarm optimization illustrated by the traveling salesman problem, http://www.mauriceclerc.net, 2000.

[18] Budinich, M. (1996). A self-organizing neural network for the traveling salesman problem that is competitive with simulated annealing. *Neural Computation*, *8*(2), 416-424.

[19] Knox, J. (1994). Tabu search performance on the symmetric traveling salesman problem. *Computers & Operations Research*, *21*(8), 867-876.

[20] Dorigo, M., & Gambardella, L. M. (1997). Ant colonies for the travelling salesman problem. *BioSystems*, *43*(2), 73-81.

[21] Schmitt, L. J., & Amini, M. M. (1998). Performance characteristics of alternative genetic algorithmic approaches to the traveling salesman problem using path representation: An empirical study. *European Journal of Operational Research*, *108*(3), 551–570.

[22] Bontoux, B., Artigues, C., & Feillet, D. (2010). A memetic algorithm with a large neighborhood crossover operator for the generalized traveling salesman problem. *Computers & Operations Research*, *37*(11), 1844–1852.

[23] Ding, C., Cheng, Y., & He, M. (2007). Two-level genetic algorithm for clustered traveling salesman problem with application in large-scale TSPs. *Tsinghua Science and Technology*, *12*(4), 459–465.

[24] Wang, Y. (2014). The hybrid genetic algorithm with two local optimization strategies for traveling salesman problem. *Computers & Industrial Engineering*, *70*, 124-133.

[25] Miller, C. E., Tucker, A. W., & Zemlin, R. A. (1960). Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, *7*(4), 326-329.

[26] Brooke, A., Kendrick, D., Meeraus, A., Raman, R., & America, U. (1998). The general algebraic modeling system. *GAMS Development Corporation*.

[27] CPLEX, I. I. (2009). V12. 1: User's Manual for CPLEX. *International Business Machines Corporation*, *46*(53), 157.

[28] Goldberg, D. E., & Holland, J. H. (1988). Genetic algorithms and machine learning. *Machine learning*, *3*(2), 95-99.

[29] Gen, M., & Cheng, R. (1997). *Genetic algorithms & engineering design*. New York: John Wiley & Sons, pp. 82–89.

[30] Michalewicz, Z. (1994). *Genetic algorithms + data structures = evolution programs*. Berlin: Springer-Verlag, pp. 209–237.

[31] Oliveto, P. S., & Witt, C. (2012, July). On the analysis of the simple genetic algorithm. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation* (pp. 1341-1348). ACM.

[32] Oei, C. K., Goldberg, D. E., & Chang, S. J. (1991). Tournament selection, niching, and the preservation of diversity (IlliGAL Report No. 91011). *Urbana, IL: University of Illinois at Urbana-Champaign*.

[33] Srinivas, M., & Patnaik, L. M. (1994). Genetic algorithms: A survey. *IEEE Journal on Computer*, *27*(6), 11–26.

[34] Wang, Y. (2014). The hybrid genetic algorithm with two local optimization strategies for traveling salesman problem. *Computers & Industrial Engineering*, *70*, 124-133.

[35] Yu, X., & Gen, M. (2010). *Introduction to evolutional algorithms*. London: Springer-Verlag, pp. 288–298, pp. 285–287.

[36] Goldberg, D. E., & Lingle, R. (1985, July). Alleles, loci, and the traveling salesman problem. In *Proceedings of an International Conference on Genetic Algorithms and Their Applications* (Vol. 154, pp. 154-159). Lawrence Erlbaum, Hillsdale, NJ.

[37] Liu, Y. H. (2010). Different initial solution generators in genetic algorithms for solving the probabilistic traveling salesman problem. *Applied Mathematics and Computation*, *216*(1), 125–137.

[38] Jain, A. K. (2010). Data clustering: 50 years beyond K-means. *Pattern recognition letters*, *31*(8), 651-666.

[39] Reynolds, A. P., Richards, G., & Rayward-Smith, V. J. (2004, August). The application of k-medoids and pam to the clustering of rules. In *International Conference on Intelligent Data Engineering and Automated Learning* (pp. 173-178). Berlin Heidelberg: Springer.

[40] Jain, A. K., & Dubes, R. C. (1988). *Algorithms for clustering data*. Prentice-Hall, Inc.

[41] RStudio Team (2015). RStudio: Integrated Development for R. *RStudio, Inc., Boston, MA, URL http://www.rstudio.com/.*

CHAPTER 4

SELECTIVE CAPACITATED VEHICLE ROUTING PROBLEM WITH BUDGET AND TIME CONSTRAINTS

Abstract

This chapter introduces a new variant of the vehicle routing problem, the capacitated vehicle routing problem with partial coverage. In this problem, the goal is to maximize the profit (served demand), subject to limited budget, travel time, and capacity. A partial demand of the nodes that are not on the route can be maintained by nodes on the route. Furthermore, each node has a limited capacity to serve other nodes, and there is a predetermined distance for each node to travel in order to obtain service. Two new genetic algorithms, one based on the assignment method (GA-AS) and the other based on the sweep method (GA-SW), are designed to solve the problem in a reasonable amount of time. In the GA-AS, the solutions are constructed based on an assignment algorithm. In the GA-SW, the solutions are structured based on the sweep method. Both GAs are coded in Scala. Moreover, to measure the efficiency of GA approaches, the problem is solved as an MILP. A variety of instances with different distributions and sizes are generated by a web-based software to perform GA approaches and MILP. For most instances, GA-SW provides a closer solution to optima. The results are presented and analyzed.

4.1 Introduction

In a vehicle routing problem, there is a fleet of supply vehicles that must depart from a single depot to serve customers, and each customer can be served by only one vehicle. Vehicle routing problems can be classified based on different objectives and constraints [1]: heterogeneous vehicle routing problem [2], capacitated vehicle routing problem [3], multiple depot vehicle routing problem [4], vehicle routing problem with time windows [5], stochastic vehicle routing

problem [6], and vehicle routing problem with pickup and delivery [7]. In a resource-constrained environment, a variation of the VRP problem may have an objective of maximizing service that is provided to a subset of cities. In disaster management problems, an objective can be to maximize the number of injured people served by trained medical staff who must travel to different locations affected by the disaster, assuming that injured people could be transferred to locations where the medical travelers could go, subject to limited time and budget constraints [8].

In this chapter, we introduce a new class of VRP, the CVRPWPC, which is defined on an undirected graph $G = (V^K \cup W^k \cup Z, E)$, where $V^K \cup W^k \cup Z$ is the vertex set, and $E = \{(v_i^k, v_j^k): v_i^k, v_j^k \in V^K \cup W^k \cup Z^k, i < j\}$ is the edge set. The set of V^K is the set of nodes served directly by vehicle *k*. The set of W^k is the set of nodes served indirectly by the nodes of set V^K. Set Z is the set of nodes not being served. A cost of c_{ij}^k and a travel time of t_{ij}^k are associated with each edge (v_i^k, v_j^k). A demand d_i and a capacity cap_i are associated with each vertex v_i. There is a limited budget and a limited time for each vehicle to serve the nodes. Therefore, each vehicle can serve a subtour of cities subject to its budget, time, and capacity. A prize is associated with each vertex, which is the total demand of the nodes that are served directly and indirectly by vehicle *k*. The aim of CVRPWPC is to determine a set of subtours with maximal profit that can serve the nodes of $V^K \cup W^k$ subject to budget and time constraints.

A variety of exact algorithms can solve the VRP [9,10]. However, using exact algorithms is not practical for large-size problems, since the number of potential solutions grows exponentially as the problem size increases. Therefore, heuristics and meta heuristics are used to solve most of the real-world VRP problems in a reasonable amount of time such as tabu search, ant colony optimization, genetic algorithm, and simulated annealing [11,12].

Taillard [11] and Rochat and Taillard [13] use tabu search to provide a benchmark for VRPs. Toth and Vigo propose a granular tabu search algorithm for VRP. In their algorithm, a reduced graph is constructed to find a good feasible solution [14]. Cordeau, Gendreau, and Laporte [15] develop the unified tabu search algorithm to solve periodic and multiple depot VRPs. Li, Golden and Wasil [16] combine a record-to-record principle with a variable-length neighbor list, which is similar to a granular tabu search.

Genetic algorithms are also widely used metaheuristics in the literature for combinatorial optimization problems including VRPs. They are used to solve VRPs with time windows [17,18], VRPs with backhauls [19], a multiple depot VRP [20], and a school bus routing problem [21].

Kallel and Boujelbene [12] propose a hybrid GA that takes into account the k-means clustering method to solve a heterogeneous vehicle routing problem, in which there is an unlimited number of heterogeneous vehicles that should serve customers. Neural networks and GAs can be used as a hybrid approach to solve the VRP [22]. Jeon, Leep, and Shim [23] propose a hybrid genetic algorithm for a VRP with heterogeneous vehicles, double trips, and multiple depots in which three different heuristics and a float mutation rate are used to escape from the local optima. Alba and Dorronsoro [24] propose a cellular GA to solve the VRP.

Wang and Lu [25] propose a hybrid GA to solve capacitated vehicle routing problems using a sweep algorithm and nearest-insertion method. Marinakis and Marinaki [26] develop a hybrid genetic algorithm that uses an education factor and swarm optimization for the vehicle routing problem. In this algorithm, solutions are allowed to evolve during their own lifetimes, and parents can teach their offspring how to improve their fitness.

Potvin and Bengio [17] use a GA to solve the VRPTW in which there are different crossover and mutation operations. In the reproduction phase, a stochastic universal selection

(SUS) is used to select the parents for crossover. Vidal, Crainic, Gendreau, Lahrichi, and Rei [27] develop a hybrid GA for multiple depot and periodic vehicle routing problems. In their paper, they address three VRPs: multiple depot, periodic, and multiple depot periodicwith capacitated vehicles and constrained route duration. Berger and Barkaoui [28] also develop a hybrid GA that combines an evolutionary search and a local search.

Potvin and Bengio [29] propose a genetic algorithm to solve the VRPTW using a strategy based on tabu search and a branch exchange. Thangiah [30] and Thangiah and Nygard [31] develop a heuristic search strategy based on a GA that uses a "cluster first, route second" strategy resulting in good feasible solutions in the genetic algorithm.

In this chapter, we define CVRPWPC as an MILP and solve medium-size problems to optimality using CPLEX, a mixed-integer programming solver [32]. To solve large-scale problems, genetic algorithm-based metaheuristics are proposed. In these genetic algorithms, two different methods are used to construct the structured solutions in the initialization phase: one inspired from the "sweep method," and the other one inspired from the "assignment method" [33, 34].

This chapter contributes to the literature in several ways: First, we propose a new capacitated vehicle routing problem in which not all customers have to be visited. Customers may travel to other nodes that are visited by a fleet of vehicles to get served. In addition, genetic algorithms using assignment and sweep methods are proposed. These methods provide a pool of structured solutions that can accelerate the convergence rate. Finally, an insertion mutation operator based on the closeness of the polar angle of nodes is developed. The problem is solved as a mixed-integer linear program for medium-size problems, in order to have a benchmark to evaluate the performance of proposed metaheuristics.

The remainder of this chapter is organized as follows: The problem is defined in detail in section 4.2 by using a mathematical model. In section 4.3, the proposed genetic algorithm has been explained. In section 4.4, computational results and analysis are presented. Finally, a conclusion is presented in section 4.5.

4.2 Mathematical Model

CVRPWPC is a selective vehicle routing problem [35] in which a set of cities with their demands, and a fleet of vehicles exist. Due to limited budget and time limit, the fleet of vehicles may not visit all nodes, thereby satisfying a partial demand. Each vehicle can load each node according to its capacity, and each node can serve a portion of the other nodes' demand if their distance is less than a predefined distance limit. Because each node can serve the other nodes, profit that can be gained from visiting a node depends on the profit of the visited nodes. The objective here is to find the best subtour with maximal total profit, i.e., served demand, for each vehicle subject to limited budget, tour duration, and limited capacity.

The mathematical model is presented as follows:

Sets

$I: set\ of\ nodes$

$K: set\ of\ vehicles$

Parameters

$i,j \in I$: nodes indices

$k \in K$: vehicle index

$tp_i^k: time\ of\ getting\ service\ at\ node\ j$

$T_{tour}^k: maximum\ time\ that\ vehicle\ k\ can\ be\ on\ route$

$t_{ij}^k: time\ of\ traveling\ between\ node\ i\ and\ node\ j$

D_j: $demand\ at\ node\ j$

c_{ij}^k: $cost\ of\ traveling\ from\ node\ i\ to\ node\ j\ by\ traveler\ k$

L_{ij}: $distance\ between\ node\ i\ and\ node\ j$

DIS_j: $maximum\ distance\ which\ people\ will\ travel\ to\ come\ to\ city\ j$

CAP_i: $capacity\ of\ node\ i$

E_j: $number\ of\ people\ who\ travel\ to\ city\ j$

$capv_k$: $capacity\ of\ vehicle\ k$

B_k: $budget\ assigned\ to\ vehicle\ k$

Variables

$$z_i^k = \begin{cases} 1 & if\ node\ i\ gets\ served\ by\ vehicle\ k \\ 0 & if\ node\ i\ does\ not\ get\ served\ by\ vehicle\ k \end{cases}$$

$$w_i^k = \begin{cases} 1 & if\ node\ i\ gets\ served\ by\ vehicle\ k\ indirectly \\ 0 & if\ node\ i\ does\ not\ get\ served\ by\ vehicle\ k\ indirectly \end{cases}$$

$$v_i = \begin{cases} 1 & if\ node\ i\ \ is\ not\ on\ the\ route\ nor\ gets\ served\ \ indirectly \\ 0 & O.W. \end{cases}$$

$$x_{ij}^k = \begin{cases} 1 & if\ node\ i\ proceed\ to\ node\ j\ by\ vehicle\ k\ immediately \\ 0 & O.W. \end{cases}$$

$$y_{ij}^k = \begin{cases} 1 & if\ node\ j\ is\ assigned\ to\ node\ i \\ 0 & O.W. \end{cases}$$

u_{ik}: $integer\ variable,\ 1 \leq\ u_{ik} \leq n-1$

Objective

$$\text{Max} \sum_{k\in K} \sum_{i\in I} z_i^k d_i + \sum_{k\in K} \sum_{i\in I} \sum_{j\in I} y_{ij}^k e_j \qquad (4.1)$$

Constraints

$$d_i \sum_{k\in K} z_i^k + \sum_{k\in K} \sum_{j\in I} y_{ij}^k e_j \leq CAP_i \sum_{k\in K} z_i^k \ \ i \in I \qquad (4.2)$$

$$\sum_{j\in I, j\neq i} x_{ij}^k = z_i^k \quad i \in I \tag{4.3}$$

$$\sum_{i\in I, i\neq j} x_{ij}^k = z_j^k \quad j \in I \tag{4.4}$$

$$\sum_{j\in I} x_{ij}^k = 1 \quad i = 0, k \in K \tag{4.5}$$

$$\sum_{j\in I} x_{ji}^k = 1 \quad i = 0, k \in K \tag{4.6}$$

$$\sum_{k\in K} z_i^k \quad + \sum_{k\in K} w_i^k + V_i = 1 \quad \forall\, i \in I \tag{4.7}$$

$$\textstyle\sum_{i\in I} y_{ij}^k = w_j^k \quad \forall j \in I, \forall k \in K \tag{4.8}$$

$$\textstyle\sum_{j\in I} \sum_{i\in I} t_{ij}^k x_{ij}^k + \sum_{i\in I} tp_i^k z_i^k \leq\ T_{tour}^k \quad \forall k \in K \tag{4.9}$$

$$\textstyle\sum_{j\in I} \sum_{i\in I} c_{ij}^k x_{ij}^k \leq B_k \ \ \forall k \in K \tag{4.10}$$

$$\textstyle\sum_{i\in I} d_i z_i^k + \sum_{i\in I} \sum_{j\in I} y_{ij}^k e_j \leq capv_k \ \ \forall k \in K \tag{4.11}$$

$$z_i^k \geq y_{ij}^k \quad \forall i, j \in I\ , \forall k \in K \tag{4.12}$$

$$L_{ij} y_{ij}^k \leq DIS_j \quad \forall\, i, j \in I \tag{4.13}$$

$$u_{ik} - u_{ik} + n x_{ij}^k \leq n - 1 \quad \forall k \in K, \forall\, i, j \in I\ i \neq j \tag{4.14}$$

$u_{ik}: integer, 0 \leq u_{ik} \leq n - 2\ , z_i^k, x_{ij}^k, y_{ij}^k, w_i^k, v_i: binary\ variables$

The objective function maximizes demand on the directly served nodes on the route, and indirectly served demand by the nodes on the route, as shown in equation (4.1). Constraint (4.2) implies that the sum of demand at a node on a route and demand at other nodes served indirectly by that node is less than the capacity of the node. Constraints (4.3) and (4.4) ensure that if a node is on a route for vehicle *k*, it can only be reached from nodes on the same route immediately

neighboring that node. Constraints (4.5) and (4.6) each ensure that each vehicle departs from the depot and returns back to the depot only once. Constraint (4.7) ensures that each node is either on the route of vehicle *k*, the node is not on the route of vehicle $\underline{k}$ but is being served indirectly by one of the nodes on the route, or the node is not served at all. Constraint (4.8) implies that a node can only obtain the service indirectly from one of the other nodes that is directly served on the route. Constraints (4.9) and (4.10) ensure that the time and cost of traveling cannot exceed the allocated time and cost budget. Constraint (4.11) implies that each vehicle capacity should be greater than the demand of the nodes that it serves directly and indirectly. Constraint (4.12) implies that if node *j* is receiving service indirectly from node *i*, then node *i* must be on the route. Constraint (4.13) limits the distance between the serving node and the served node (i.e., distance between directly served and indirectly served nodes). Constraint (4.14) is the subtour elimination constraint for each route. This mathematical program includes $3\,[I][K] + [I]^2[K] + [I]^2 + [I]$ variables, whereby $[I][K]$ variables are integers and the rest are binary variables. Moreover, there are $4\,[I] + 5\,[K] + [I]^2 + 2[I]^2[K]$ constraints. The number of variables and constraints are determined by summing the cardinality of all variables and constraints.

Proposition 1. The complexity of CVRPWPC problem is $\binom{n}{x_1,\ldots,x_m}\frac{(x_1-1)!}{2}\times \ldots \times \frac{(x_m-1)!}{2}\times 2^{n-(x_1+\cdots+x_m)}$, such that $x_1+\cdots+x_m \le n$, where *m* is number of vehicles and *n* is number of nodes.

Proof:

The number of visited nodes by a typical vehicle denoted by *m* can be a variable denoted by x_m. The number of different ways that *n* nodes can be assigned to *m* vehicles is $\binom{n}{x_1,\ldots,x_m}$. Each vehicle can have $\frac{(x_m-1)!}{2}$ Hamiltonian cycles for an asymmetric distance matrix. For the

remaining nodes that are not served directly (i.e. $n-(x_1+\cdots+x_m)$), there are two possibilities: either they can be either assigned to one of the nodes on the route or not. Then, there are $2^{n-(x_1+\cdots+x_y)}$ cases of assigning nodes. In total, there are $\binom{n}{x_1,\ldots,x_m}\frac{(x_1-1)!}{2}\times\ldots\times\frac{(x_m-1)!}{2}\times 2^{n-(x_1+\cdots+x_m)}$ ways to be evaluated.

Proposition 2. CVRPWPC is NP-hard.

Proof:

The TSP is NP-hard [36]. In an asymmetric TSP, there are $\frac{(n-1)!}{2}$ cycles to be evaluated to find the best Hamiltonian cycle. Complexity of the CVRPWPC is $\binom{n}{x_1,\ldots,x_m}\frac{(x_1-1)!}{2}\times\ldots\times\frac{(x_m-1)!}{2}\times 2^{n-(x_1+\cdots+x_m)}$, which is greater than $\frac{(n-1)!}{2}$ cycles. Therefore, CVRPWPC is NP-hard.

Because the problem is NP-hard and it is difficult to solve large-scale problems, for larger instances of this problem, one may need to use heuristic methods to find good-quality solutions in a reasonable amount of time. Thus, we propose to solve the CVRPWPC using a genetic algorithm-based metaheuristic.

4.3 Genetic Algorithm to Solve CVRPWPC

Genetic algorithms are inspired from the evolution theory and have been very successful in solving combinatorial optimization problems such as the TSP and VRP. The proposed GA algorithm, GA-CVRPWPC, begins with an initial population that corresponds to several solutions, and improves the current solution iteratively using GA operators such as mutation and crossover. The GA-CVRPWPC implementation is summarized in Table 4.1, and details of the implementation are presented in sections that follow.

Table 4.1. GA Procedure

Step	Pseudo Code of Proposed Genetic Algorithm
1	Set number of generations *N* and the population size
2	Set selection probability p_s, crossover probability p_c, mutation probability p_m
3	Generate initial population
4	While (number of iterations is less than *N*)
5	Evaluate each chromosome according to its fitness function
	5.1 Solve the assignment problem to find fitness for each path
6	Select chromosomes according to their fitness using roulette wheel selection until pool is full
7	Execute crossover operation
	7.1 Check feasibility of each chromosome
	7.2 Modify unfeasible chromosome to make it feasible
8	Execute mutation
9	End

4.3.1 Initialization

To encode a tour, a path representation is used. For instance, a tour of 20 cities can be shown as follows

$$ch^1 = (|1,2,3,6,15|, |13,16,19|, |4,11,12,18|, |5,9,10,19|, 0,0,0,0)$$

This chromosome is representing four paths of depot-1-2-3-6-15-depot, depot-13-16-19-depot, depot-4-11-12-18-depot, and depot-5-9-10-19-depot, where each path is served by one vehicle. The 0 shows that the other nodes are not on the route.

Two methods are used to determine the paths. The first method is the sweep approach proposed by Gillett and Miller [33]. First, the polar angle of each node is calculated. Then a city is randomly generated to be assigned to a vehicle. The order of cities on the route is based on their polar angle. This process is done until the feasibility of solution is violated. (i.e., at least one of the budget, time limit, and capacity constraints is violated.)

The second method, which is used to generate initial solutions is inspired from the assignment problem [34]. The pseudo code is summarized in Table 4.2.

Table 4.2. Assignment Method of CVRPWPC

Step	Pseudo Code of Assignment Method
1	Determine cones
2	Define seed for each vehicle/cone
3	Calculate threshold for each node at each cone relative to seed of corresponding cone
4	While each path is feasible,
	Assign nodes to vehicles based on its threshold
9	End

Some locations will be considered as seed points of vehicles. These seed points are dummy points and are defined in order to calculate the assignment cost of nodes to each vehicle. The assignment of nodes to vehicles is based on the traveling cost to these seeds and the potential profit of visiting those nodes. This method can be applied to the planar case where whole cities are on a plane and distances between the cities can be measured by using the Euclidean distance. The plane should be partitioned into *k* cones. Each cone is constructed by merging the cities' cones. Each city cone is defined by drawing rays from the depot that bisect the angle between adjacent cities and corresponding city. The cone of each vehicle is formed by merging the adjacent cones of cities

until the cumulative demand of those cities in the cone is less than the vehicle capacity. To determine the location of each seed, we need to know the distance of the seed from depot and the direction vector of that seed. Each seed is located along the ray that bisects the angle of the corresponding vehicle cone (i.e., dividing the cone angle into two equal angles), which is the direction vector of that seed. The distance of each seed from the depot is fixed and is equal to the maximum distance of the cities belonging to vehicle cone.

Then, the cities are assigned to the vehicles based on the following threshold:

$$CT_{si} = cap_i/[c_{0i} + c_{si} - c_{0s}] \quad (4.15)$$

Assignment cost of node *i* to seed *s* equals $c_{0i} + c_{si} - c_{0s}$ [23], where *s* is the index of seed city, c_{is} is the cost of traveling between cities *i* and seed *s*, c_{0s} is the cost of traveling between the depot and the seed city, and cap_i is the capacity of city *i*.

The insertion is done until the feasibility of the solution is violated, i.e.,:

- Total cost of traveling between nodes on the route of each vehicle is greater than the assigned budget to that vehicle.
- Total capacity of the nodes on the route of each vehicle is greater than the capacity of each vehicle.
- Total time of traveling between nodes on the route of each vehicle is greater than the maximum time that the vehicle can be on the route.

In other words, assignment will be performed until any budget, capacity, or time constraints are violated. The order of nodes is based on their polar angles.

4.3.2 Fitness Evaluation and Selection

In genetic algorithms, each chromosome is evaluated based on a fitness function to select the chromosomes for crossover and mutation. The fitness is a function of the problem objective function value, the number of customers that have been served for a given chromosome:

$$f = \exp(obj) \tag{4.17}$$

where f is the fitness of the chromosome, and obj represents the maximum number of served customers on the tour. To find the fitness of each chromosome, an optimization problem is solved to determine the maximum number of served customers given assigned paths to different vehicles in that chromosome.

The maximum number of served nodes comes from assigning nodes that are not on the route to the nodes on route optimally. This is an optimization problem where a set of nodes should be assigned to another set of nodes with a certain serving capacity. Furthermore, the distance of nodes assigned to a node on route cannot be more than a predetermined distance.

To find the objective function value for this set of fixed paths, the following assignment problem is solved as mixed-integer linear program:

Parameters

$$\bar{Z}_i = \begin{cases} 1 & if\ node\ i\ is\ on\ any\ given\ route \\ 0 & o.w. \end{cases}$$

Variables

$$h_i = \begin{cases} 1 & if\ node\ i\ is\ on\ any\ route\ but\ served\ indirectly \\ 0 & o.w. \end{cases}$$

$$q_{ij} = \begin{cases} 1 & if\ node\ j\ is\ assigned\ to\ node\ i \\ 0 & o.w. \end{cases}$$

Model

$$\max \sum_{i\in I} D_i \bar{Z}_\iota + \sum_{j\in I}\sum_{i\in I} \mathrm{E}_j q_{ij} \tag{4.17}$$

Subject to

$$\bar{Z}_\iota + h_i + V_i = 1 \qquad \forall\, i \in I \tag{4.18}$$

$$\sum_{i\in I} q_{ij} = W_j \qquad \forall j \in I \tag{4.19}$$

$$\bar{Z}_\iota \geq q_{ij} \qquad \forall i, j \in I \tag{4.20}$$

$$\sum_{j\in I} E_j q_{ij} + D_i \bar{Z}_\iota \leq CAP_i \bar{Z}_\iota \qquad \forall i \in I \tag{4.21}$$

$$L_{ij} q_{ij} \leq DIS_j \qquad \forall\, i, j \in I \tag{4.22}$$

$$h_i, q_{ij}: binary\ variables$$

Given a chromosome, we already know which nodes are on the route, i.e., $\bar{Z}_\iota$ is known. Constraint (4.18) ensures that each node takes only one of three possible states: the node is on the route, the node is not on the route but is served partially by one of the nodes on the route, or the node is neither on the route nor served indirectly. Constraint (4.19) ensures that each node can be served indirectly by only one of the other nodes on the route. Constraint (4.20) implies that if node *i* is served by node *j*, then node *j* must be on the route (otherwise it cannot serve node *i*). Constraint (4.21) limits the number of customers served directly and indirectly at each node by the capacity of that node. Constraint (4.22) limits the distance between the serving node and the served node. In this problem, there are $2\,[I] + [I]^2$ binary variables and $3\,[I] + 2[I]^2$ constraints. Compared to the original problem, there are $[I]^2[K] + 3[I][K] - [I]$ less number of variables and $5[K] + 2[I]^2[K] + [I] - [I]^2$ less number of constraints.

After computing fitness, roulette wheel selection is implemented to select parents for the crossover operation.

4.3.3 Crossover

After selecting two parents, the parent with best fitness is chosen as the first parent. Then a two-points crossover is performed to generate the offspring. In this crossover, the chromosomes must be transformed to new chromosomes based on a different representation, in which each chromosome includes the vehicle number serving the city located in the corresponding sequence of genes [38]. For instance, in ch_1^p, the second gene is 1, which implies that the second city is served by vehicle 1, and because the seventh gene is 0, city 7 is not served (Figure 4.1). The key point is to number the vehicles according to their polar angles. Then, for all cities in the same area, the same vehicles would serve the same region. To perform the transformation, for each vehicle, the average coordinates of visited cities by this vehicle is calculated, and the polar angle of that point is calculated. Then for the first parent, numbering is performed sequentially. To number vehicles of the second parent, the measurement criterion is the difference between the polar angles of the visited cities by vehicles of the second parent and the first parent. Vehicles are numbered according to the closeness of the polar angles of visited cities.

Then, two points are randomly generated to determine the crossover points. The first offspring inherits genes of first parent located to the left side of first crossover point and to the right side of second crossover point, while getting genes of the second parent located between two crossover points. The second child is receiving genes opposite to the first offspring. After generating offspring, chromosomes must be recoded again in order to have the same representation as the original chromosomes. Note that, during initialization, cities are sorted based on their polar angles, which in turn facilitates transition to the previous path representation.

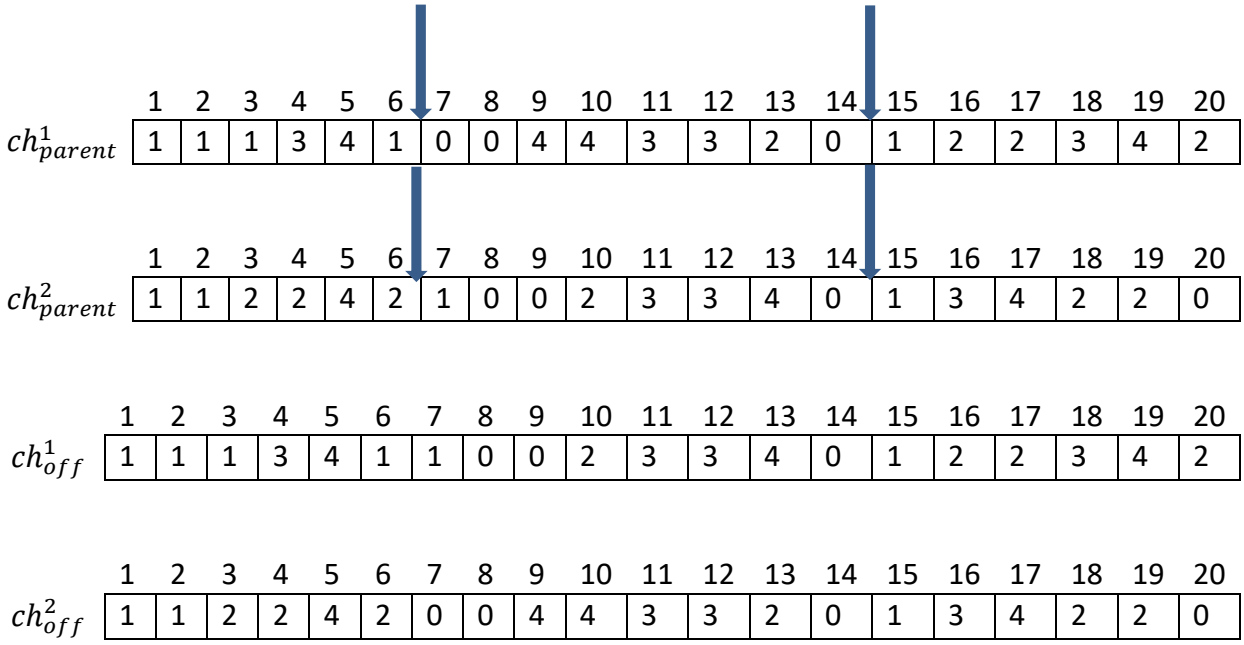


Figure 4.1. Crossover

After creating offspring, time limit and budget constraints should be checked with respect to feasibility. In the case of infeasibility, some cities are dropped randomly from the current route to achieve feasibility. The infeasibility of repeated nodes on the same route cannot happen in this new representation of chromosomes, because each node can only obtain service from a vehicle once or not at all.

4.3.4 Mutation

In order to create a diverse set of solutions, mutation is performed. The first mutation operator is a two-exchange mutation in which all possible swapping of two nodes on the current route is examined to determine the best move that reduces the cost of traveling.

The second type of mutation is an insertion mutation that results in an increased number of served cities. To implement the insertion mutation, a list of not-served cities is determined for any individual chromosome in the new pool. Then, for each vehicle, the average polar angle of cities on the route is measured, and among the cities not assigned, the city that has the minimum

difference between its polar angle and the average polar angle is assigned to that vehicle. This procedure is repeated until adding another city violates the feasibility of the route.

4.3.5 Elitism

The best chromosomes of each generation survive. This assures that the solution in each generation does not worsen.

4.4 Computational Results and Analysis

Two methods are used to create structured solutions in order to construct the initial population: sweep method and assignment method. The sweep method (GA-SW) assigns nodes to each vehicle randomly, where nodes are sorted based on the increasing amount of their polar angle. The assignment method (GA-AS) assigns a seed to each vehicle where nodes are assigned to each vehicle based on the closeness of a criterion to the vehicle seed. The sequence of nodes is based on the increasing amount of their polar angle. This initialization procedure is explained in section 4.3.1 in detail.

Both of the proposed genetic algorithms have been coded in Scala programming language [39] in order to test the performance of the algorithms. Moreover, each instance is solved as an MLIP in the GAMS [40] using the CPLEX [32] solver to provide a benchmark from which to compare the results of both GAs to the optima.

4.4.1 Selection of Control Parameters

Different parameters in GA should be tuned. The main parameters are as follows:

- Size of population (sp) implies how many chromosomes are in the population of any generation.
- Selection probability (p_s) indicates how often crossover is performed.
- Mutation probability (p_m) presents how often parts of a chromosome are mutated.

- Elitism probability (p_e) defines the rate of saving the best solutions of a previous generation in a new generation.

The probabilities are set such that the sum of p_s, p_m, and p_e are equal to one. The number of computational cycles (N_{max}) is set such that the final solution is no longer changed, and the CPU time is practical. A different set of parameters is considered, the GAs are run for a given data set for five times, and the best combination resulting in the highest profit is selected. The set of parameters is provided in Table 4.3.

Table 4.3. GA Control Parameters Set

Control Parameters	**Values**					
Size of Population (*sp*)	100,50,30	100,50,30	100,50,30	100,50,30	100,50,30	100,50,30
Selection Probability (p_s)	0.7	0.65	0.6	0.55	0.50	0.50
Mutation Probability (p_m)	0.2	0.25	0.3	0.35	0.4	0.45
Elitism Probability (p_e)	0.1	0.1	0.1	0.1	0.1	0.05

For different-size CVRPWPC problems, the GA parameters are optimized, as shown in Tables 4.4 and 4.5. For larger-size problems, the population size should decrease, and the number of generations should increase due to the increase in CPU time. The best selection and crossover probabilities changes slightly, but the mutation rate increases to provide a higher diversity when the number of cities increases. It is observed that for GA-AS, the mutation rate is slightly greater than for GA-SW. In GA-AS, the initial solutions are constructed based on determination of cones. Therefore, the diversity of the initial population of GA-AS is less than the diversity of the initial population of GA-SW. A higher mutation rate helps to improve the solutions in terms of diversity. As can be seen, the mutation rate is slightly greater for GA-AS (Table 4.5). Having more diverse solutions help to probe new search areas. Consequently, the chance of escaping from the local optima increases.

Table 4.4. Parameters of GA-SW

Number of Cities	N_{max}	sp	p_s	p_m	p_e
$n \leq 50$	30	100	0.6	0.3	0.1
$50 < n \leq 90$	40	100	0.55	0.35	0.1
$90 < n \leq 200$	60	30	0.55	0.40	0.05

Table 4.5. Parameters of GA-AS

Scale of TSP	N_{max}	sp	p_s	p_m	p_e
$n \leq 50$	30	100	0.55	0.35	0.1
$50 < n \leq 90$	40	100	0.55	0.35	0.1
$90 < n \leq 200$	60	30	0.50	0.45	0.05

4.4.2 Experimental Design

A web-based geographic information system software, GeoMidpoint (http://www.geomidpoint.com/random/), is used to generate the problem data in order to have different distributions of potential customer locations to estimate the performance of the proposed algorithms. In this software, by providing the latitude and longitude of the depot, cities are randomly generated around the depot within a certain distance. The distance between cities is calculated using the Euclidean distance

Each experiment is solved by using three different methods: GA-AS, GA-SW, and an MILP on a computer with 8 GB RAM, Core™ i7 CPU, and 465 GB hard disk space.

We want to evaluate the efficiency of proposed algorithms for randomly dispersed instances. Therefore, ten randomly dispersed instances with different sizes are generated. A predetermined demand and capacity are assigned to each node. Demand and capacity of each node

is constant and equal. The cost of traveling is assumed to be $0.15 per mile. The budget amount is set such that the computational time of solving the problem with CPLEX is large. In some cases, where CPLEX fails to solve the problem, the upper limit of the CPLEX solution is captured. Moreover, each instance is solved considering two different numbers of vehicles: two and three.

Because determining the cones is based on the depot location, in the GA-AS approach, we construct two different sets of instances and change the location of depot for each instance. In the first, there are two vehicles, and in the second, there are three vehicles. GA-AS and GA-SW are run for each instance.

For each experiment, the problem is solved using two genetic algorithms, GA-SW and GA-AS, and an MILP for medium-size problems where budget and capacity are set such that the solution can be found for that size of the problem in order to determine the efficiency of the proposed metaheuristics. The solutions found by GA-SW and GA-AS are compared to the solution found by MILP to obtain the relative error:

$$RE = \frac{optimal\ profit\ upper\ bound - best\ profit}{optimal\ profit\ upper\ bound} \tag{4.25}$$

As it can be seen in this formula, the RE is calculated regarding to the upper bound of the optimal profit. The reason for this is that in some cases, the problem cannot be solved to optimality by CPLEX. In this case, the upper bound of the optimal profit calculated by CPLEX is used instead. In the case of finding an optimal solution, the optimal profit is used instead of its upper bound.

For each instance, the genetic algorithms are run five times, and the average and best profit are recorded. In general, over the same number of generations, GA-SW is providing an optimal solution faster than GA-AS for random distributed nodes. Although GA-AS provides better solutions in the very first generations, after a number of generations, the solution becomes steady

and does not improve in the next computational cycles. This could be due to the low diversity of solutions of the initial population.

On the other hand, for cases where the cones are well separated, GA-AS is providing an optimal solution faster than GA-SW, over the same number of generations. Where the nodes are not distributed fairly around the depot, the solution of GA-AS is worse than the solution of GA-SW at each generation. Hence, GA-AS is not recommended in this case. The details of different experiments are discussed in following section.

4.4.3 Numerical Results

The following subsections will show the following: (a) effect of budget amount on the optimal solution for a 16-node instance and two vehicles, (b) evaluation of the GA-SW and GA-AS performance for randomly generated instances with a different number of vehicles, and (c) evaluation of the GA-SW and GA-AS performance on clustered instances with varying depot locations.

4.4.3.1 Effect of Budget Amount on Optimal Solution for 16-Node Instance

For a 16-node instance, we investigate the impact of the budget on the optimal solution and CPU time needed by CPLEX to determine this solution. First, we calculate the minimum budget amount to visit all nodes as a vehicle routing problem (3824), which also gives the maximum amount of profit (160). We then limit the total budget in 10% increments of the determined budget starting with 10%. The total profit corresponding to each case is listed in Table 4.6.

Table 4.6. Profit and CPU Time for Exact Method under Different Budget Amounts (Budget = 3824)

Budget ($)	Percent of Minimum Cost	Profit	Number of Served Nodes	Number of Indirectly Served Nodes	Number of Not Served Nodes	CPLEX CPU Time (sec)
382.4	10	0	0	0	16	0.732
764.8	20	45	3	3	13	1.157
1147.2	30	60	4	4	8	4.173
1529.6	40	105	7	7	2	2.703
1912	50	125	9	7	0	6.015
2294.4	60	135	11	5	0	8.628
2676.8	70	140	12	4	0	3.652
3059.2	80	145	13	3	0	4.455
3441.6	90	150	14	2	0	4.102
3824	100	160	16	0	0	2.811

As can be seen in Table. 4.6, the most difficult instance of the problem that requires the most CPU time is when the budget is around 60% of the required budget for visiting all nodes. All other instances seem to have a shorter execution time, which shows that they are easier instances to solve, such as those cases where we have 10% and 100% of the required budget. Increasing the amount of the budget (thus increasing the size of the feasible region) results in gaining more profit (i.e., served demand). Moreover, the number of unserved nodes decreases when the budget amount increases directly or indirectly served nodes. As can be seen, after serving more than half of the population directly (i.e., number of directly served nodes is equal to 9), the number of not-served nodes equals zero.

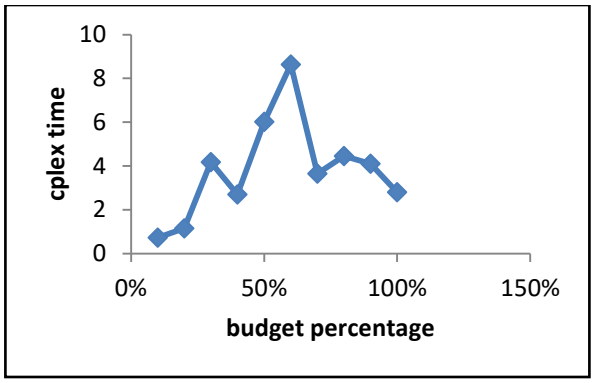


Figure 4.2. Impact of budget on CPU time required for finding optimal solution obtaining final solution percentage on CPLEX time

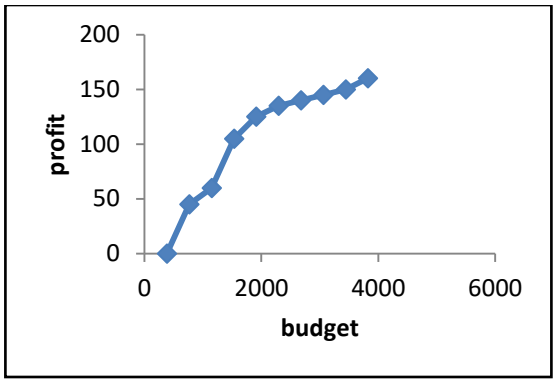


Figure 4.3. Impact of budget amount on optimal profit

As it can be expected, increasing the amount of budget results in gaining more profit by virtue of having a larger feasible region. As expected, more than a minimum cost budget to visit all nodes amount does not improve the objective function.

4.4.3.2 GA Evaluation for Randomly Dispersed Instances

To analyze the efficiency of GA-SW and GA-AS, ten randomly dispersed instances of different sizes are generated. For each instance, two- and three-vehicle scenarios are considered. The budget amount is set to be 60% of the minimum amount needed to visit all nodes, i.e., it is computationally more expensive to determine the optimal solution compared to other scenarios. In some cases where CPLEX runs out of memory while solving the problem, the upper limit of the CPLEX solution is captured. The largest problem, RA-140 with three vehicles, takes 49.394 seconds of computational time using the GAs programmed in Scala. These results are presented in Tables 4.6 and 4.7. Furthermore, the trend of convergence of the proposed genetic algorithms for each instance is shown in Figures 4.3 and 4.4. The optimal profit is also gained by running GAMS using CPLEX solver to calculate the gap of each solution from the optima and to evaluate the

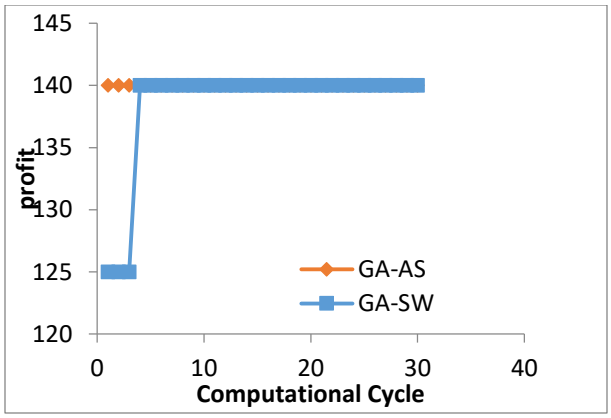
efficiency of each algorithm. The gap is captured in the RE column. For most cases, GA-SW is providing a closer solution to the optima.

Table 4.7. Results of GA Approaches for Different-Size Instances (Two Vehicles)

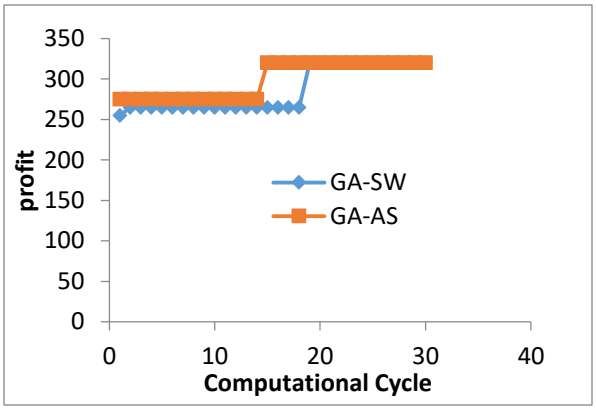
Name	Average Profit		Best Profit		Optimal Profit	RE (%)		CPLEX CPU time (sec)
	GA-SW	GA-AS	GA-SW	GA-AS		GA-SW	GA-AS	
RA-16	140.00	138.50	155	150	155	0	0	4.102
RA-32	286.67	299.00	320	320	320	0	0	461.427
RA-50	431.67	430.00	440	440	440	0	0	4586.659
RA-60	526.50	518.62	540	525	560	0.03	0.06	12,861.185
RA-70	594.75	598.12	600	600	600	0	0	32,396.194
RA-80	642.12	554.62	700	630	700	0	0.1	13,157.18
RA-90	813.25	775.75	820	780	840	0.02	0.07	22,483.948
RA-100	955.80	944.20	960	950	960	0	0.01	9,670.867
RA-120	936.67	931.83	940	945	1020	0.07	0.08	14,752.992*
RA-140	1146.75	1133.42	1155	1135	1200	0.04	0.05	17,199.284*

* Indicates CPLEX ran out of memory at that point

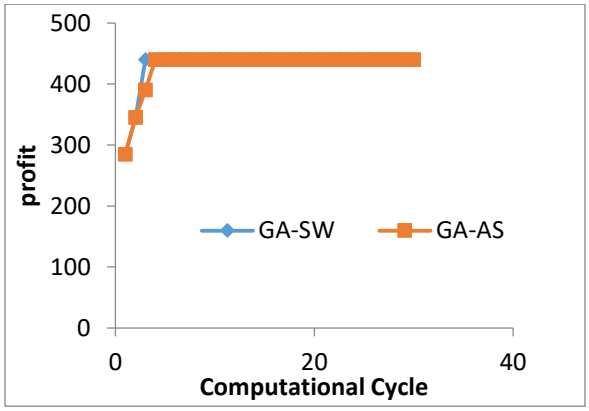
Table 4.7 shows the average profit and best profit obtained from running each GA for a two-vehicles instance: both GAs provide good solutions, and relative error is less than 8% in all instances. In larger-size problems, the optimality gap of GA-SW is less than the optimality gap of GA-AS. Moreover, the average profit of GA-SW is greater than the average profit of GA-AS, which is due to the faster convergence of GA-SW to a better solution. As can be seen, the computational time is very high for CPLEX, while for Scala, it takes 32.572 seconds to solve a large-scale problem (i.e., RA-140), 18.200 seconds to solve a medium-size problem (i.e., RA-90), and 8.125 seconds to solve a small-size problem (i.e., RA-32).



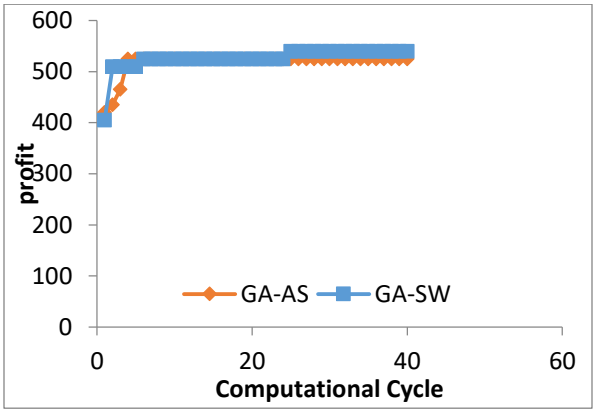
(a) RA-16



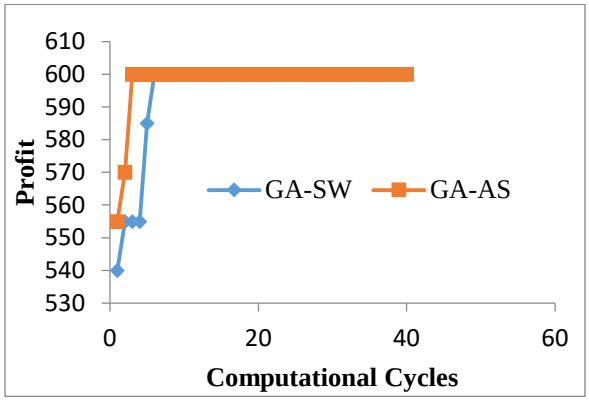
(b) RA-32



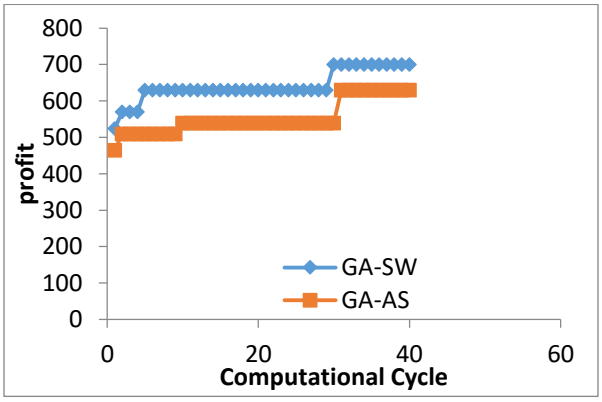
(c) RA-50



(d) RA-60

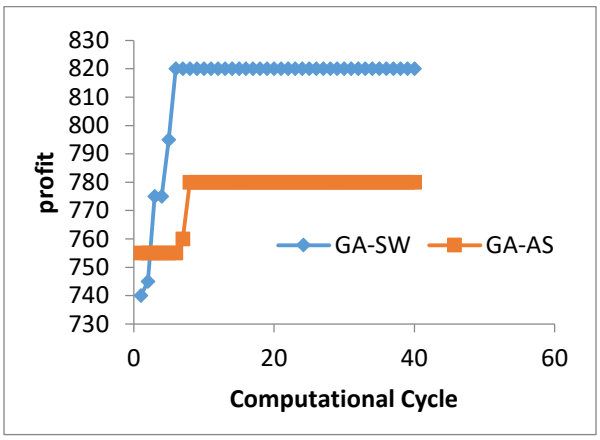


(e) RA-70

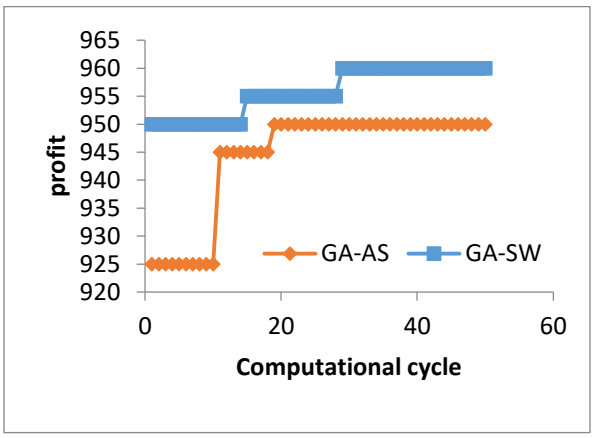


(g) RA-80

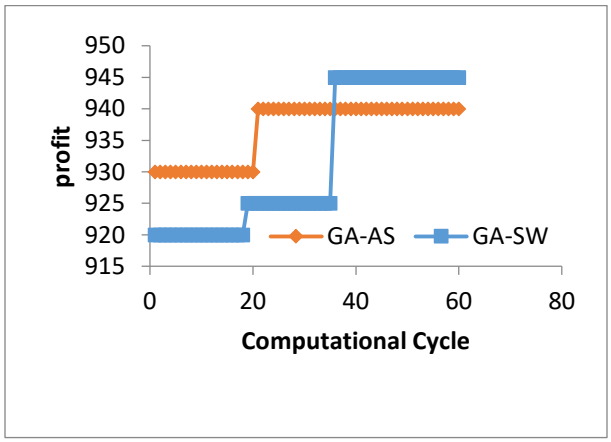
Figure 4.4. Trend in profit changes over different generations for random generated instances with two vehicles



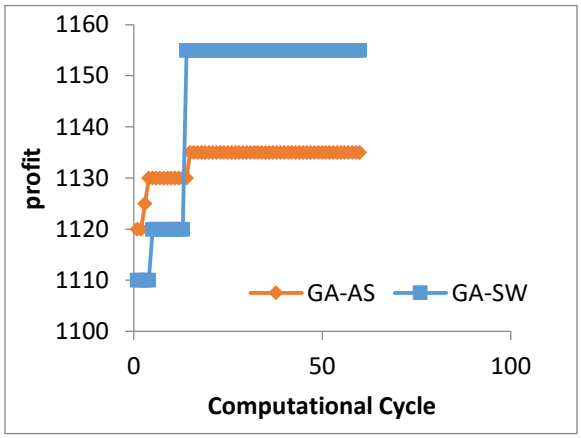
(h) RA-90



(i) RA-100



(j) RA-120



(k) RA-140

Figure 4.4. (continued)

Figures 4.4(a) 4to4.4 (k) also show that GA-SW is providing a better solution in the long-run. Although GA-AS is providing better initial solutions faster, for most instances, GA-SW provides better solutions after a number of generations. The difference between these algorithms is in the initialization phase: GA-AS is constructing solutions cone by cone, while GA-SW considers whole nodes in constructing a solution. Hence, diversity of the initial population is greater for GA-SW, and the solutions escape from local optima when GA-SW is used.

As shown in Figure 4.5, the absolute error of both GAs decreases as the number of computational cycles increases. In most instances, the absolute error of GA-AS is less than the absolute error of GA-SW for the very first computational cycles; however, the absolute error of GA-SW converges to a smaller number compared to GA-AS over the same number of computational cycles.

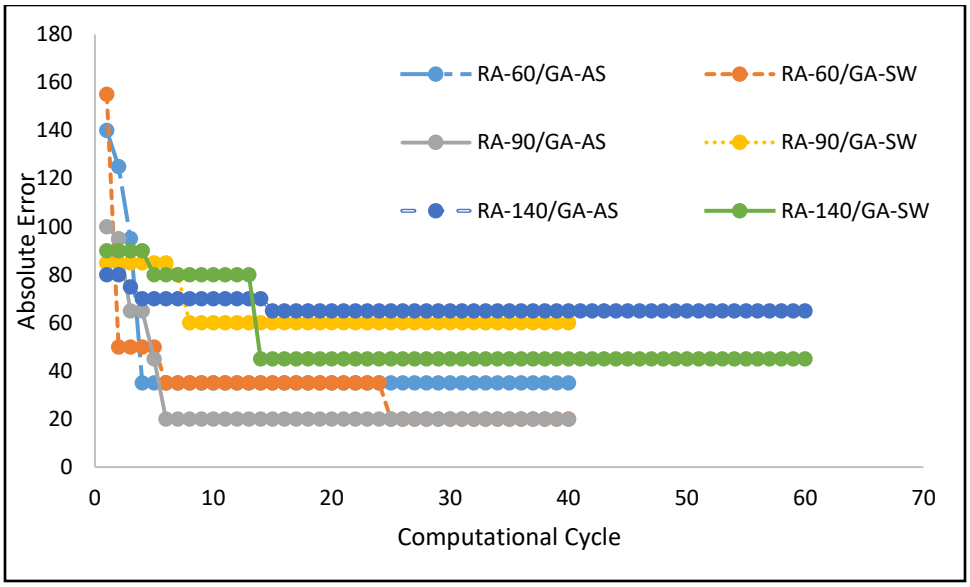


Figure 4.5. Trend of convergence of absolute error for different instances with two vehicles

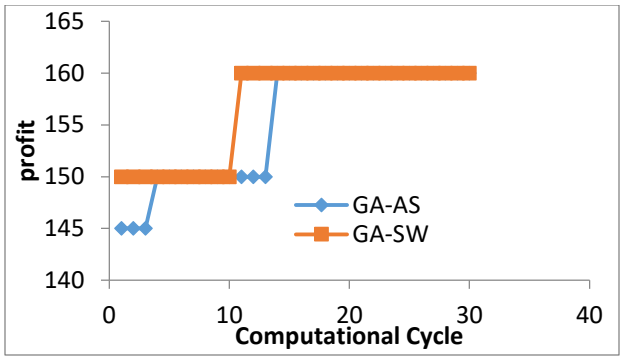
Table 4.8 shows the average profit and best profit obtained from running each GA when there are three vehicles. As the number of vehicles increases, GA-SW is providing a closer solution to optima because diversity of the solutions is higher due to considering all nodes to construct solutions in the initialization phase and convergence happens faster. GA-AS constructs the route cone by cone; therefore, each route is constructed out of fewer number of nodes, and the quality of the initial solution is highly dependent on cone determination. As can be seen, the computational time is very high for CPLEX, while for Scala, it takes 92.284 seconds to solve a large-scale problem (i.e., RA-140), 46.571 seconds to solve a medium-size problem (i.e., RA-90), and 14.225 seconds to solve a small-size problem (i.e., RA-32).

Table 4.8. Results of GA Approaches for Different-Size Instances (Three Vehicles)

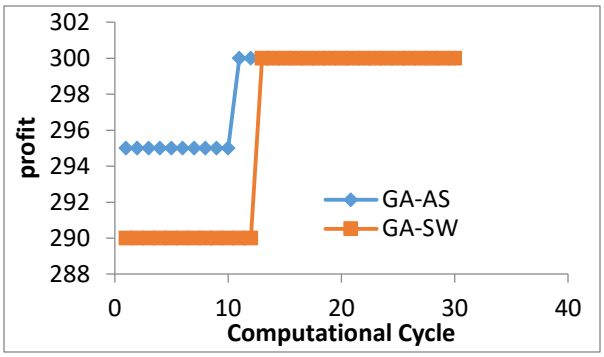
Name	Average Profit		Best Profit		Optimal Profit	RE		CPLEX CPU Time (sec)
	GA-SW	GA-AS	GA-SW	GA-AS		GA-SW	GA-AS	
RA-16	156.670	155.160	160	160	160	0	0	4.510
RA-32	298.330	296.000	300	300	105	0	0	461.414
RA-50	463.500	451.670	470	455	480	0.021	0.052	6.015
RA-60	523.500	522.750	545	535	550	0.009	0.027	10,913.719
RA-70	578.375	572.125	600	575	600	0.000	0.042	3,665.509
RA-80	688.125	679.625	700	690	780	0.103	0.115	14,257.217*
RA-90	770.500	751.750	780	760	780	0.000	0.026	1,489.062
RA-100	985.750	948.750	990	955	1000	0.010	0.045	5,655.106
RA-120	969.000	954.500	975	960	1050	0.074	0.086	27,914.336
RA-140	1190.000	1120.375	1200	1130	1200	0.000	0.058	22,832.515

* Indicates CPLEX ran out of memory at that point

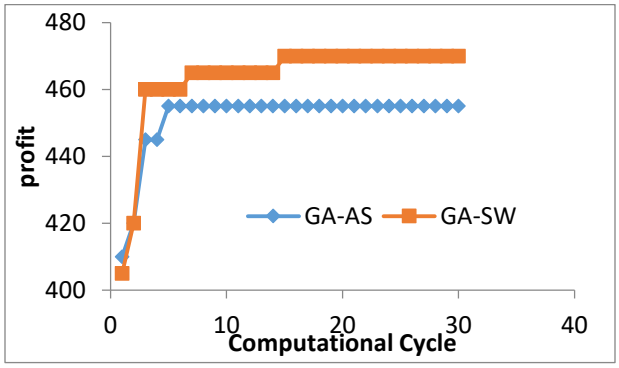
Figures 4.6(a) to 4.6(k) also show that GA-SW is providing better solution in the long run. As can be seen, for the same instances where the number of vehicles increases, GA-SW provides a closer solution to optima. Because GA-AS partitions the plane into a greater number of cones, the number of nodes at each cone is less than the case with two vehicles. Hence, each route is constructed out of fewer nodes, and the diversity of solutions is less than GA-SW solutions in which all routes are constructed considering all nodes. Moreover, GA-SW converges to a better solution faster than GA-AS does, resulting in a higher average profit for GA-SW.



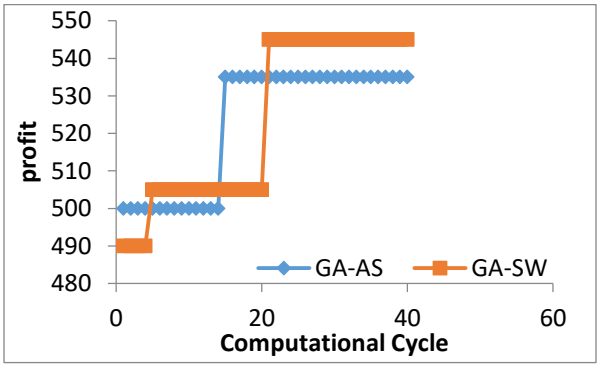
(a) RA-16



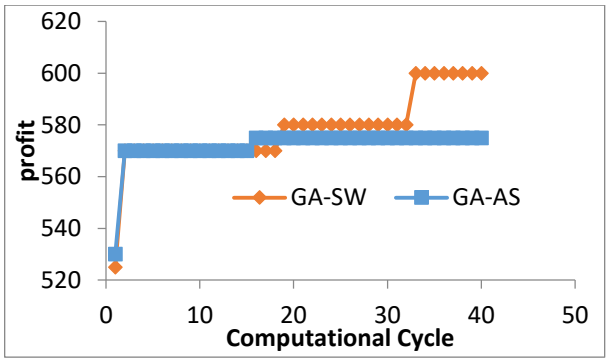
(b) RA-32



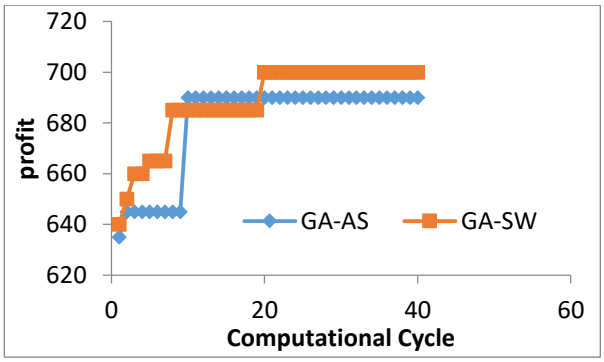
(c) RA-50



(d) RA-60

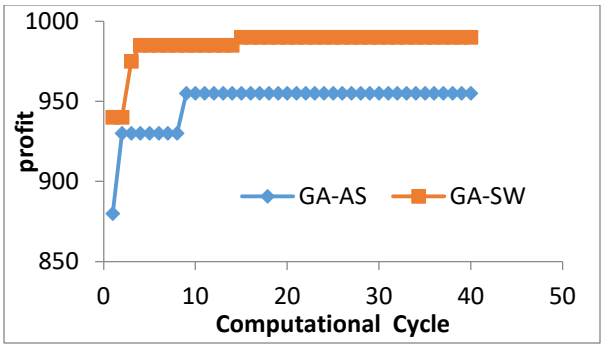


(e) RA-70

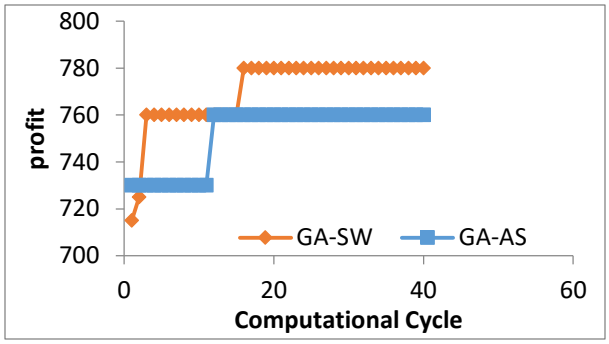


(f) RA-80

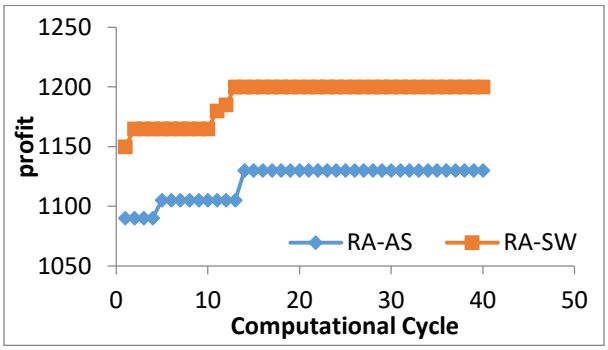
Figure 4.6. Trend in profit changes over different generations for random generated instances with three vehicles



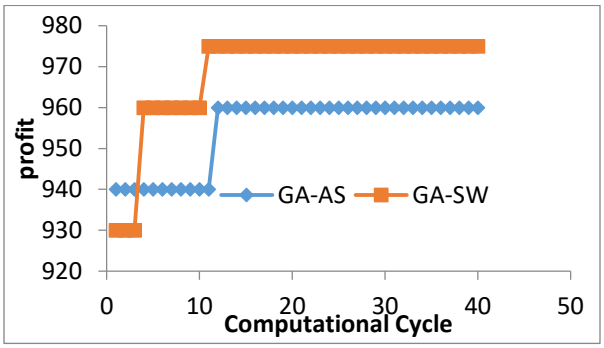
(g) RA-90



(h) RA-100



(i) RA-120



(j) RA-140

Figure 4.6. (continued)

Figure 4.7 shows that the absolute error of GA-SW is considerably smaller than that of GA-AS at the end of the computational cycles. Furthermore, as the number of vehicles increases, GA-SW provides a solution closer to optima, in comparison to GA-AS. Note that GA-AS results in a smaller absolute error in the very first computational cycles.

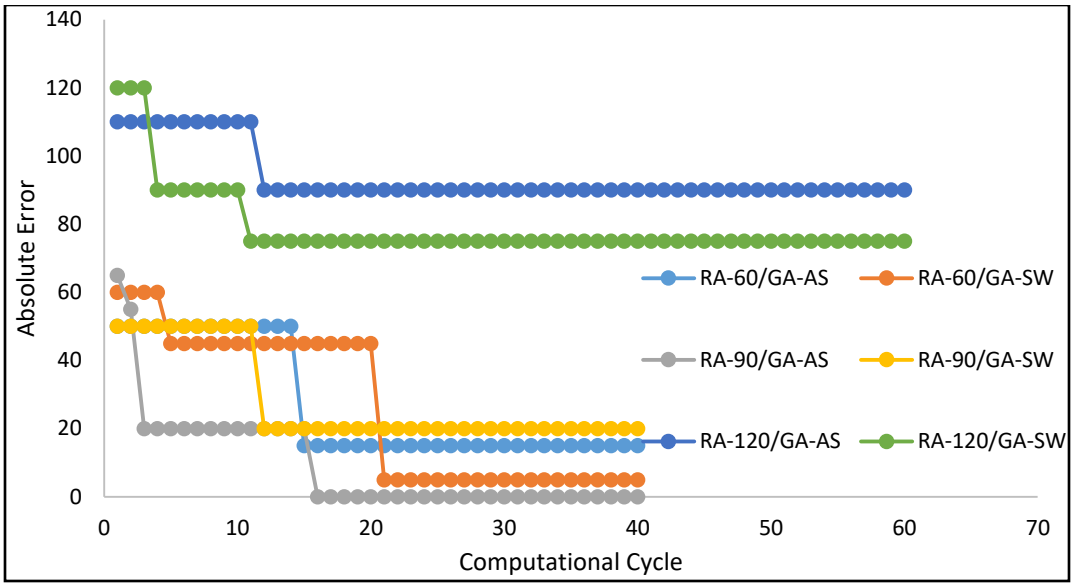
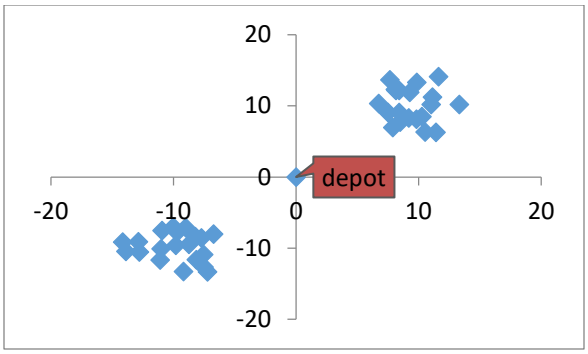


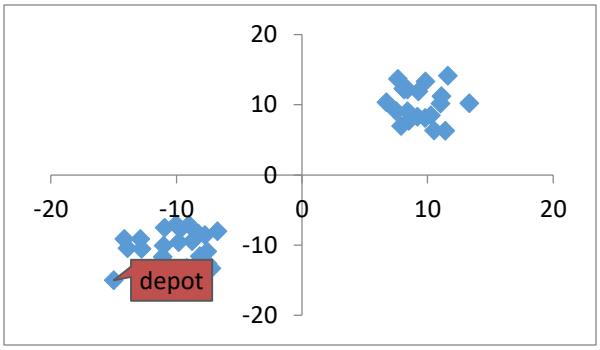
Figure 4.7. Convergence trend of absolute error for different instances with three vehicles

4.4.3.3 Impact of Distribution of Nodes and Clustering on GA Performance

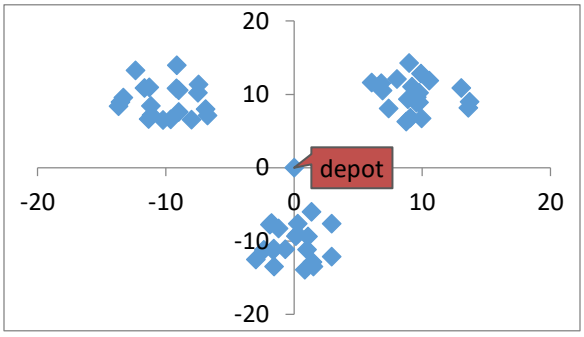
To evaluate the effect of depot location (starting and returning point) on the performance of the GAs, two different node distributions are considered. In the first, two clusters and two vehicles are considered, as shown in Figures 4.8(a) and 4.8(b). In the second, three clusters and three vehicle are considered, as shown in Figures 4.8(c) and 4.8(d). For each of these examples, the impact of the depot location is studied. The first case has all nodes dispersed around the depot in clusters, and the second case has nodes located on one side of the depot in clusters. Both GA-SW and GA-AS cases are run for five times. For Scala, it takes 8.278 seconds on average to solve the problem. The average profit and the best profit of these runs is presented in Table 4.9. To calculate the relative error from the optimal solution, an MILP is solved using the GAMS/CPLEX solver.



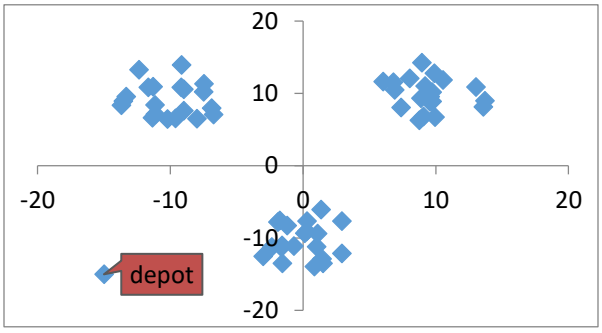
(a) 2-cluster-r



(b) 2-cluster-nr



(c) 3-cluster-r



(d) 3-cluster-nr

Figure 4.8. Different distributed instances

Table 4.9. Results of GA Approaches for Different Distributed Instances

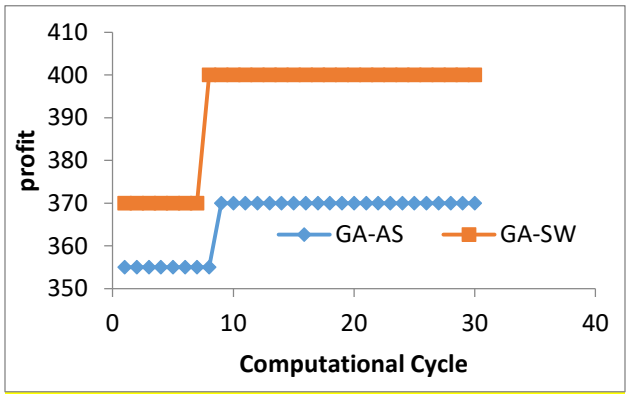
Name	Average Profit		Best Profit		Optimal Profit	Number of Directly Served Nodes			RE (%)		CPLEX CPU Time (sec)
	GA-SW	GA-AS	GA-SW	GA-AS		GA-SW	GA-AS	Optimal	GA-SW	GA-AS	
2-cluster-r	346.33	318	365	365	365	33	33	33	0	0	1.534
2-cluster-nr	393	366	400	370	400	40	34	40	0	0.07	80.259
3-cluster-r	563.75	571.875	575	575	600	55	55	60	0.04	0.04	192.669
3-cluster-nr	592.625	580	595	580	600	56	59	60	0.008	0.03	150.271

* indicates CPLEX ran out of memory at that point

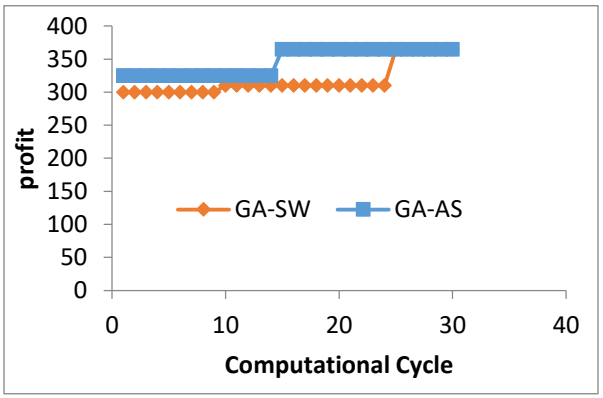
As shown in Figure 4.9(a) and 4.9(c), where nodes are not randomly dispersed around the depot, GA-AS does not initialize good solutions, because dividing the plane into cones is based on the polar angle of nodes calculated regarding the depot position. Consequently, nodes of a cone may be at a far distance from each other, resulting in poor initial solutions. GA-AS constructs

solutions cone by cone, which results in lower diversity, compared to GA-SW, which considers whole nodes to construct solutions. Hence, worse initial solutions along with low diversity results in decreasing the convergence trend to optima

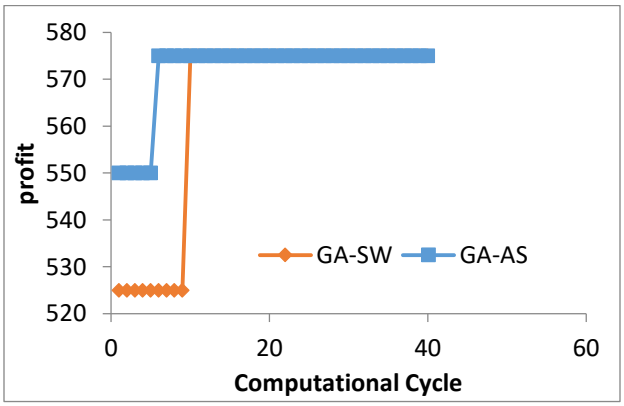
On the other hand, where nodes are well dispersed around the depot, as shown in Figures 4.9(b) and 4.9(d), GA-AS results in better solutions and converging to the local optima faster. The reason for this is that the cones are defined more accurately. Furthermore, when the number of nodes are smaller at each cone, the chance of initializing better solutions increases, compared to GA-SW, which considers all nodes in the initialization phase.



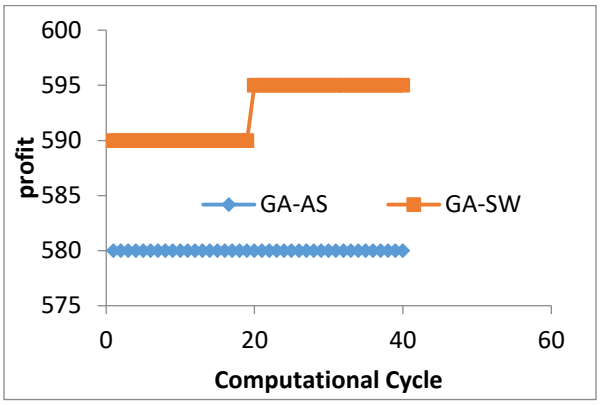
(a) 2-cluster-nr



(b) 2-cluster-r



(c) 3-cluster-r



(d) 3-cluster-nr

Figure 4.9. Trend in profit changes over different generations for clustered instances

We also notice that the computation time is quite large for CPLEX for most cases, while the GA finds a very good solution in a short amount of time. To show the difference, we capture the computation time for each algorithm over its convergence to optima for the RA-90 problem with three vehicles (Figure 4.10). As can be seen, GA-SW finds the optima after 46.571 seconds, while CPLEX finds a solution with absolute error of 10% in 55.83 seconds (i.e., with profit of 695). CPLEX takes 1,176.37 seconds to find the optimal solution, which is considerably longer than the GA-SW computation time.

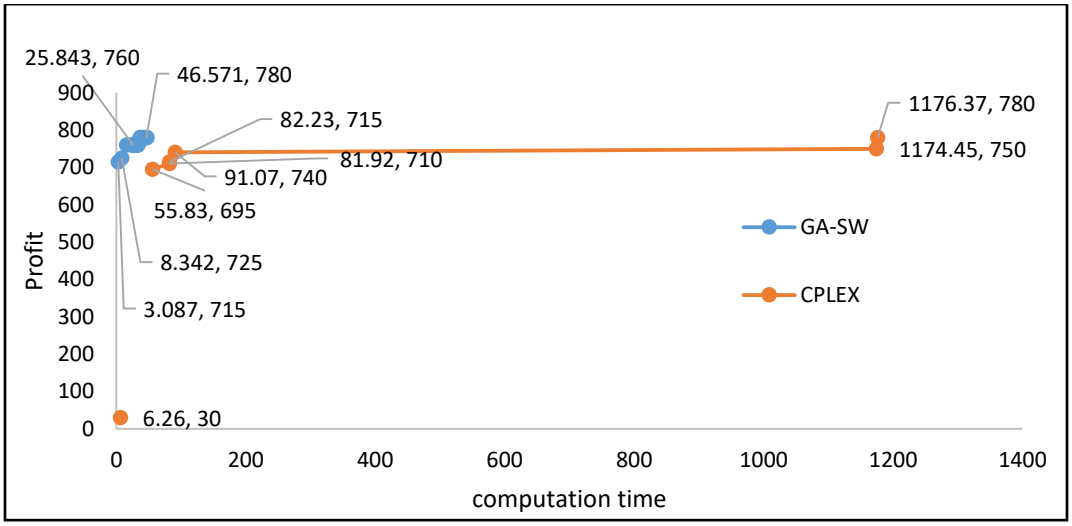


Figure 4.10. Comparison of computation time for CPLEX and GA-SW for RA-90 with three vehicles

Some observations are noted from running GA algorithms for different instances. First, where nodes can be fitted well to cones around the depot, as shown in Figures 4.8(b) and 4.8(d), GA-AS can provide better solutions and converge to optima faster, while in the case of not having fairly distributed cones around the depot, as mentioned earlier, GA-AS initializes a pool of not diverse solutions the quality of which is not good due to poor cone determination. Consequently, this can decrease the pace of converging to optima.

Second, where nodes are randomly dispersed around a depot, GA-SW and GA-AS both can provide good solutions. Although the initial solutions of GA-AS are more often better, in the long run, the convergence to optima is more promising with GA-SW, due to dividing the plane into cones. Where GA-AS limits each route to a specific cone, which decreases the pace of convergence to optima, in GA-SW, the diversity of the solutions in the pool leads to getting closer to optima after a reasonable number of computational cycles.

4.5 Conclusion

A new vehicle routing problem, CVRPWPC, is introduced in this chapter. Due to its complexity, the MILP model cannot find the solution for large-scale problems in a reasonable amount of time. Furthermore, the MILP execution time of an instance is variable due to the budget and tour duration amounts. If these are set to very large or very small, the execution time is lower, compared to the case where the budget and time are constrained at a medium range. In the first case, most cities can be on the route, whereas in the second case, only a very few cities would be on the route.

Since this problem is NP-hard, two genetic algorithms, GA-SW and GA-SA, are proposed. GA-SW provides a better solution after a reasonable number of generations, while GA-AS is converging to optima faster, providing that the nodes are well fitted to cones and the nodes are fairly dispersed around the depot. In the case of not having randomly dispersed nodes, GA-AS can initialize solutions far from optima, and due to limiting the solutions to cones in the initial pool, escaping from the local optima becomes harder.

Parameters for the GAs play an important role in their performance. Generally, with larger parameters, the solutions are better. For instance, with a larger population size and iteration number, the chance of finding a better solution increases. However, for large-scale problems, a

large population size implies a larger computational time. Furthermore, due to having lower diversification of solutions in the initial population in GA-AS, the selection and mutation parameters are greater than their corresponding parameters in GA-SW.

This research can be extended to the case where there are multiple depots and each depot has a fleet of vehicles. Moreover, a fixed cost could be considered for each vehicle, and the number of vehicles on the route could be a variable as well. We address these two problems in the chapters that follow.

4.6 References

[1] Bodin, L., Golden, B., Assad, A., & Ball, M. (1981). The state of the art in the routing and scheduling of vehicles and crews (No. UMTA-MD-11-0004-81-2Final Rpt.).

[2] Baldacci, R., Battarra, M., & Vigo, D. (2008). Routing a heterogeneous fleet of vehicles. In *The vehicle routing problem: Latest advances and new challenges* (pp. 3-27). Springer US.

[3] Toth, P., & Vigo, D. (2002). Models, relaxations and exact approaches for the capacitated vehicle routing problem. *Discrete Applied Mathematics*, *123*(1), 487-512.

[4] Lim, A., & Wang, F. (2005). Multi-depot vehicle routing problem: A one-stage approach. *IEEE Transactions on Automation Science and Engineering*, *2*(4), 397-402.

[5] Kallehauge, B., Larsen, J., Madsen, O. B., & Solomon, M. M. (2005). Vehicle routing problem with time windows. In *Column generation* (pp. 67-98). Springer US.

[6] Gendreau, M., Laporte, G., & Séguin, R. (1996). Stochastic vehicle routing. *European Journal of Operational Research*, *88*(1), 3-12.

[7] Min, H. (1989). The multiple vehicle routing problem with simultaneous delivery and pick-up points. *Transportation Research Part A: General*, *23*(5), 377-386.

[8] Arulselvan, A. (2009). Network model for disaster management (Doctoral dissertation). University of Florida, Gainesville.

[9] Toth, P., & Vigo, D. (1998). Exact solution of the vehicle routing problem. In *Fleet management and logistics* (pp. 1-31). Springer US.

[10] Christofides, N., Mingozzi, A., & Toth, P. (1981). Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Mathematical Programming*, *20*(1), 255-282.

[11] Taillard, É. (1993). Parallel iterative search methods for vehicle routing problems. *Networks*, *23*(8), 661-673.

[12] Kallel, S. A., & Boujelbene, Y. (2013). Heterogeneous vehicle routing problem with profits dynamic solving by clustering genetic algorithm. *International Journal of Computer Science Issues*, *10*(4), 247-253.

[13] Rochat, Y., & Taillard, É. D. (1995). Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, *1*(1), 147-167.

[14] Toth, P., & Vigo, D. (2003). The granular tabu search and its application to the vehicle-routing problem. *Informs Journal on computing*, *15*(4), 333-346.

[15] Cordeau, J. F., Gendreau, M., & Laporte, G. (1997). A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks*, *30*(2), 105-119.

[16] Li, F., Golden, B., & Wasil, E. (2005). Very large-scale vehicle routing: new test problems, algorithms, and results. *Computers & Operations Research*, *32*(5), 1165-1179.

[17] Potvin, J. Y., & Bengio, S. (1996). The vehicle routing problem with time windows part II: genetic search. *INFORMS journal on Computing*, *8*(2), 165-172.

[18] Thangiah, S. R. (1995, July). An adaptive clustering method using a geometric shape for vehicle routing problems with time windows. In *Proceedings of the 6th International Conference on Genetic Algorithms* (pp. 536-545). Morgan Kaufmann Publishers Inc..

[19] Potvin, J. Y., Duhamel, C., & Guertin, F. (1996). A genetic algorithm for vehicle routing with backhauling. *Applied Intelligence*, *6*(4), 345-355.

[20] Salhi, S., Thangiah, S. R., & Rahman, F. (1998). A genetic clustering method for the multi-depot vehicle routing problem. In *Artificial Neural Nets and Genetic Algorithms* (pp. 234-237). Vienna: Springer.

[21] Thangiah, S. R., & Nygard, K. E. (1992, March). School bus routing using genetic algorithms. In *Aerospace Sensing* (pp. 387-398). International Society for Optics and Photonics.

[22] Potvin, J. Y., Dubé, D., & Robillard, C. (1996). A hybrid approach to vehicle routing using neural networks and genetic algorithms. *Applied Intelligence*, *6*(3), 241-252.

[23] Jeon, G., Leep, H. R., & Shim, J. Y. (2007). A vehicle routing problem solved by using a hybrid genetic algorithm. *Computers & Industrial Engineering*, *53*(4), 680-692.

[24] Alba, E., & Dorronsoro, B. (2006). Computing nine new best-so-far solutions for capacitated VRP with a cellular genetic algorithm. *Information Processing Letters*, *98*(6), 225-230.

[25] Wang, C. H., & Lu, J. Z. (2009). A hybrid genetic algorithm that optimizes capacitated vehicle routing problems. *Expert Systems with Applications*, *36*(2), 2921-2936.

[26] Marinakis, Y., & Marinaki, M. (2010). A hybrid genetic–Particle Swarm Optimization Algorithm for the vehicle routing problem. *Expert Systems with Applications*, *37*(2), 1446-1455.

[27] Vidal, T., Crainic, T. G., Gendreau, M., Lahrichi, N., & Rei, W. (2012). A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, *60*(3), 611-624.

[28] Berger, J., & Barkaoui, M. (2003). A new hybrid genetic algorithm for the capacitated vehicle routing problem. *Journal of the Operational Research Society*, *54*(12), 1254-1262.

[29] Potvin, J-Y, & Bengio, S. (1993). A Genetic Approach to the Vehicle Routing Problem with Time Windows, Technical Report, CRT-953, Centre de Recherche sur les Transports, Universite de Montreal, C.P. 6128, Succ. A, Montreal, Canada H3C 3J7.

[30] Thangiah, S. R. (1993). *Vehicle routing with time windows using genetic algorithms*. Artificial Intelligence Lab., Slippery Rock Univ..

[31] Thangiah, S. R., & Nygard, K. E. (1993, March). Dynamic trajectory routing using an adaptive search method. In *Proceedings of the 1993 ACM/SIGAPP symposium on Applied computing: states of the art and practice* (pp. 131-138). ACM.

[32] CPLEX, I. I. (2009). V12. 1: User's Manual for CPLEX. *International Business Machines Corporation*, *46*(53), 157.

[33] Gillett, B. E., & Miller, L. R. (1974). A heuristic algorithm for the vehicle-dispatch problem. *Operations Research*, *22*(2), 340-349.

[34] Fisher, M. L., & Jaikumar, R. (1981). A generalized assignment heuristic for vehicle routing. *Networks*, *11*(2), 109-124.

[35] Defryn, C., Sörensen, K., & Cornelissens, T. (2016). The selective vehicle routing problem in a collaborative environment. *European Journal of Operational Research*, *250*(2), 400-411.

[36] Hachicha, M., John Hodgson, M., Laporte, G., & Semet, F. (2000). Heuristics for the multi-vehicle covering tour problem. *Computers & Operations Research*, *27*(1), 29-42.

[37] Baker, B. M., & Ayechew, M. A. (2003). A genetic algorithm for the vehicle routing problem. *Computers & Operations Research*, *30*(5), 787-800.

[38] Hachicha, M., John Hodgson, M., Laporte, G., & Semet, F. (2000). Heuristics for the multi-vehicle covering tour problem. *Computers & Operations Research*, *27*(1), 29-42.

[39] Odersky, M& al. (2004). *An overview of the Scala programming language*. EPFL Lausanne, Switzerland

[40] Brooke, A., Kendrick, D., Meeraus, A., Raman, R., & America, U. (1998). The general algebraic modeling system. *GAMS Development Corporation*.

CHAPTER 5

SELECTIVE MULTIPLE TRAVELING SALESMAN PROBLEM WITH BUDGET AND TIME CONSTRAINTS

Abstract

This chapter introduces a new variant of the multiple traveling salesman problem, or multiple traveling salesman problem with partial coverage. The goal here is to maximize the profit (served demand), subject to a limited budget. Since there is an assignment cost for having a salesman on route, the number of salesmen is not fixed. The tour time for each salesman cannot be more than a time limit. Furthermore, each node has a limited capacity to serve other nodes, and there is a predetermined distance for each node to travel in order to obtain the service. Two genetic algorithms, GA-2OPT and GA-IR, are designed to solve the problem. In GA-2OPT, a cross-route and greedy 2-OPT mutation is implemented. In cross-route mutation, some parts of two routes are swapped. In a greedy 2-OPT mutation, each route is improved based on a greedy 2-OPT algorithm in terms of travel cost and travel time. In GA-IR, a cross-route mutation and in-route mutation are implemented. In the in-route mutation, two random nodes are swapped, and in the cross-route mutation, some parts of the two routes are swapped. Both GAs are coded in Scala programming language. To measure the efficiency of the proposed GAs, the problem is solved as an MILP. A variety of instances with different distributions and sizes are generated by a web-based software to perform GA approaches and MILP. The results are presented and analyzed.

5.1 Introduction

The multiple traveling salesman problem is a generalization of the traveling salesman problem, which aims to find a set of routes for *m* vehicles starting from and returning to the same depot. The MTSP is a relaxation of the vehicle routing problem whereby the capacity constraints

are removed [1]. This problem can be applied to the case where a set of tours are planned for a group of doctors in an undeveloped region to serve the maximum number of patients. The amount of time each doctor can spend is limited. Due to the limited budget and cost for assigning a doctor, the number of traveling doctors is not fixed. The service capacity of each node is determined. Some patients from other nodes can be served by nodes on the tour, providing that the distance between the nodes is not greater than a predetermined distance. In disaster management problems, one objective can be to maximize the number of served injured people by trained medical staff who must travel to different locations affected by the disaster, assuming that injured people could be transferred to locations where the medical travelers could go, subject to limited time and budget constraints [2].

The MTSP can have a single depot from which all vehicles start and to which all vehicles return, or it can have multiple depots, whereby a number of vehicles start from a specific depot and can either return to the same depot (fixed destination problem) or any other depot (non-fixed destination problem). The MTSP can also be classified based on the number of vehicles as well: when the number of salesmen is not fixed, there is a fixed cost associated with utilizing a vehicle. Thus, one objective for this problem can be to minimize the number of vehicles serving the demand, in order to minimize the cost of the transportation fleet. In addition, when some nodes need to be visited in a specific time period, the problem is referred to as the multiple traveling salesman problem with time windows [1].

In this chapter, the new MTSPWPC is introduced. In this problem, due to limited resources, entire nodes cannot be visited and served directly by all of the vehicles. However, each node has a certain capacity to serve other nodes that are not on the route indirectly. If the distance between a node on the route and a node off the route is less than a predetermined distance, then the node

off the route can be served indirectly by a node on the route. The MTSPWPC is defined on an undirected graph $G = (V^K \cup W^k \cup Z, E)$, where $V^K \cup W^k \cup Z$ is the vertex set, and $E = \{(v_i^k, v_j^k): v_i^k, v_j^k \in V^K \cup W^k \cup Z^k, i < j\}$ is the edge set, where i and j are nodes i and j, and k represents a vehicle. The set of V^k is the set of nodes served directly by vehicle k. The set of W^k is the set of nodes served indirectly by the nodes of set V^K. The set Z is the set of nodes not being served. A cost c_{ij}^k and a travel time t_{ij}^k are associated with each edge (v_i^k, v_j^k). A demand d_i and a capacity cap_i are associated with each vertex v_i. A fixed cost fc_k is associated with assigning a vehicle k. In this problem, there is a limited budget for the total cost of assigning the vehicles and traveling cost to visit all nodes on the route. Furthermore, there is a limited time for each vehicle to serve the nodes. Hence, each vehicle can serve a subtour of the cities subject to its travel time and travel budget. A prize is associated with each vertex, which is basically total demand of the nodes served directly and indirectly by vehicle k. The aim of the MTSPWPC is to create a set of subtours of maximal profit that can serve the nodes of $V^K \cup W^k$ subject to budget and time constraints.

The problem is different than the CVRPWPC, which was introduced in Chapter 4) since in MTSPWPC, the number of vehicles is not fixed, and the goal is to minimize the number of vehicles while maximizing the profit, i.e., served demand.

In the literature, considerable attention is given to solving traveling salesman problems, some involving exact algorithms, such as branch and bound, and cutting planes, and some involving heuristic algorithms, such as tabu search, neural networks, etc. [3, 4, 5, 6]. There are several exact algorithms to solve the MTSP. Ali and Kennington [7] solve the problem directly without converting the problem into a TSP using a branch and bound algorithm, where a Lagrangian relaxation-based algorithm is used by relaxing the degree constraints, and a Lagrangian

dual is solved by using a subgradient algorithm. Gromicho, Paixão, and Bronco [8] solve the MTSP by relaxing the subtour elimination constraints based on a quasi-assignment relaxation. Laporte and Nobert [9] present one of the first approaches to solving the MTSP without transforming it to a TSP by relaxing some constraints. Transformation of the problem will result in many suboptimals, and then finding the optimal becomes complex. There is a related fixed cost for each traveler, which is considered when a route is constructed. In their proposed algorithm, first, the problem is solved by relaxing the subtour elimination constraints to find a solution, and then, the subtour elimination constraints are added to existing subtours.

Gavish and Srikanth [10] propose an algorithm to solve large-scale MTSPs. Their algorithm is a branch-and-bound algorithm where its lower bound is found solving a Lagrangian problem by relaxing the degree constraints

Because the MTSP is NP-hard, when the size of the problem increases, the computational time increases exponentially. Due to the problem's complexity, metaheuristics are used to solve large-scale problems. Feng, Bao, and Ye [11] use a combination of methods, ant colony optimization and swarm optimization, to solve the MTSP. Nallusamy, Duraiswamy, Dhanalaksmi, and Parthiban [12] use a combination of k-means algorithms, a shrink-wrap algorithm, and metaheuristics to solve the MTSP.

Metaheuristics, such as genetic algorithms, is used to solve the MTSP in a reasonable amount of time [13]. In vehicle-scheduling problems, which are a special type of MTSP problem, capacity constraints and time windows constraints are considered. Tang, Liu, Rong, and Yang [14] solve the MTSP using a GA by adding dummy nodes and converting the MTSP to a TSP. Carter and Ragsdale [15] present an approach with a two-part chromosome to solve an MTSP in path planning, which eliminates redundant constraints and results in reducing the search space size.

In this chapter, we model the MTSPWPC as a mixed-integer linear program and solve the problem as a multiple traveling salesman problem without transforming the MTSP into a TSP. Since the problem is NP-hard, two different genetic algorithms are designed to solve the problem, the size of which is large.

In the proposed genetic algorithms, we take advantage of the fact that the MTSP is a relaxation of the CVRPWPC, and we use a similar methodology for the crossover of the parents as used in the GA designed for the CVRPWPC. Because the number of vehicles depends on the budget, a sweep method is used to construct the initial solutions. The proposed GAs are different in mutation. In one of the algorithms, we use a cross-route and in-route mutation, while in the other algorithm, we use a cross-route and GA-2OPT. After implementing the mutations, an insertion is implemented for each solution to add more nodes on the route within the released budget. This insertion is based on the closeness of the polar angles of nodes not served to the average polar angle of nodes on each route.

This chapter contributes to the literature in several ways: First, we propose a new multiple traveling salesman problem in which not all customers must be visited. Furthermore, customers may travel to other nodes that are served by a fleet of vehicles to get served. In addition, a genetic algorithm is proposed, and two different mutation procedures are developed to improve the current solution in terms of cost of traveling. The problem is solved as a mixed-integer linear program for medium-size problems, in order to have a benchmark to evaluate the performance of the proposed metaheuristics.

The remainder of the paper is organized as follows. In section 5.2, the mathematical model is presented. In section 5.3, the proposed genetic algorithms are explained. Computational results and analysis in section 5.4 are followed by conclusions in section 5.5.

5.2 Mathematical Model for MTSPWPC

The MTSPWPC is a selective multiple traveling salesman problem in which there is a set of cities with demands and there is a fleet of vehicles with associated fixed costs. Due to limited budget, all vehicles may not be used, and each used vehicle may not be on a route longer than a predetermined amount of time. The objective here is to maximize the served demand by constructing a set of optimal subtours subject to limited budget and limited tour time. Each vehicle can serve nodes that can serve other nodes partially, if the distance of the served node to the node on a path is less than a predetermined coverage distance. Because each node can serve other nodes, the profit that can be gained from visiting a node in a tour depends on the nodes visited previously. Because the nodes which are visited previously might serve some of the nodes and this newly visited node cannot serve them anymore. Since those nodes are already served by previously visited nodes.

We provide the notation, parameters, variables, and mathematical model for the MTSPWPC as follows:

Sets

I: set of nodes

K: set of vehicles

Parameters

i,j∈I: nodes indices

k∈K: vehicle index

c_{ij}^{k}: $cost\ of\ traveling\ from\ node\ i\ to\ node\ j\ by\ vehicle\ k$

L_{ij}: $distance\ between\ node\ i\ and\ node\ j$

DIS_j: $maximum\ distance\ which\ people\ will\ travel\ to\ come\ to\ city\ j$

CAP_i: $capacity\ of\ node\ i$

E_j: $number\ of\ people\ who\ travel\ to\ city\ j$

D_i: $demand\ of\ node\ i$

t_{ij}^k: $travel\ time\ between\ node\ i\ and\ node\ j\ by\ vehicle\ k$

tp_i^k: $service\ time\ in\ node\ i\ by\ vehicle\ k$

FC: $fixed\ cost\ of\ assigning\ vehicle$

B: $budget$

T_{tour}^k: $maximum\ time\ vehicle\ k\ can\ be\ on\ \ route.$

Variables

$$r_k = \begin{cases} 0 & if\ vehicle\ k\ is\ not\ used \\ 1 & if\ vehicle\ is\ used \end{cases}$$

$$z_i^k = \begin{cases} 1 & if\ node\ i\ gets\ served\ by\ vehicle\ k \\ 0 & if\ node\ i\ does\ not\ get\ served\ by\ vehicle\ k \end{cases}$$

$$w_i^k = \begin{cases} 1 & if\ node\ i\ gets\ served\ by\ vehicle\ k\ indirectly \\ 0 & if\ node\ i\ does\ not\ get\ served\ by\ vehicle\ k\ indirectly \end{cases}$$

$$v_i = \begin{cases} 1 & if\ node\ i\ \ is\ not\ on\ route\ norgets\ served\ indirectly \\ 0 & O.W. \end{cases}$$

$$x_{ij}^k = \begin{cases} 1 & if\ node\ i\ proceed\ to\ node\ j\ by\ vehicle\ k\ immediately \\ 0 & O.W. \end{cases}$$

$$y_{ij}^k = \begin{cases} 1 & if\ node\ j\ is\ assigned\ to\ node\ i \\ 0 & O.W. \end{cases}$$

u_{ik}: $auxilary\ variable$

The mathematical model for the MTSPWPC is as follows:

Objective

$$\text{Max} \sum_{k\in K} \sum_{i\in I} z_i^k d_i + \sum_{k\in K} \sum_{i\in I} \sum_{j\in J} y_{ij}^k e_j \qquad (5.1)$$

Constraints

$$d_i \sum_{k\in K} z_i^k + \sum_{k\in K}\sum_{j\in J} y_{ij}^k e_j \le CAP_i \sum_{k\in K} z_i^k \ , i \in I \tag{5.2}$$

$$\sum_{j\in J, j\neq i} x_{ij}^k = z_i^k \ \ i \in I \ , k \in K \tag{5.3}$$

$$\sum_{i\in I, i\neq j} x_{ij}^k = z_j^k \ \ j \in I \ , k \in K \tag{5.4}$$

$$\sum_{j\in J} x_{ij}^k = r_k \ \ i = 0, k \in K \tag{5.5}$$

$$\sum_{j\in J} x_{ji}^k = r_k \ \ \ i = 0, k \in K \tag{5.6}$$

$$\sum_{k\in K} z_i^k + \sum_{k\in K} w_i^k + V_i = 1 \ \ \forall \ i \in I \tag{5.7}$$

$$z_i^k \le r_k \ \ \ \forall k \in K, \forall i \in I \tag{5.8}$$

$$\sum_{i\in I} y_{ij}^k = w_j^k \ \ \ \forall j \in I, \forall k \in K \tag{5.9}$$

$$\sum_{j\in J}\sum_{i\in I} t_{ij}^k x_{ij}^k + \sum_{i\in I} tp_i^k z_i^k \le \ T_{tour}^k \ \ \ \forall k \in K \tag{5.10}$$

$$\sum_{k\in K}\sum_{j\in J}\sum_{i\in I} c_{ij}^k x_{ij}^k + \sum_{k\in K} r_k FC \le B \tag{5.11}$$

$$z_i^k \ge y_{ij}^k \ \ \ \forall i, j \in I, k \in K \tag{5.12}$$

$$L_{ij} y_{ij}^k \le DIS_j \ \ \ \forall \ i, j \in I, k \in K \tag{5.13}$$

$$u_{ik} - u_{ik} + n x_{ij}^k \le (n-1) r_k + M(1 - r_k) \ \ \forall k \in K, \forall \ i \in I \tag{5.14}$$

$$u_{ik}: integer, 0 \le u_{ik} \le n-2; \quad r_k, z_i^k, w_i^k, v_i, x_{ij}^k y_{ij}^k : binary \tag{5.15}$$

Equation (5.1) is the objective function, which maximizes the total profit (served demand) of nodes on the routes for all vehicles and the partial profit (served demand) of other nodes getting served by the nodes on the route. Constraint (5.2) implies that the demand of each node on a route and the demand of the other nodes served by that node is less than the capacity of that node. Constraints (5.3) and (5.4) ensure that each vehicle departs the depot and returns back to the depot

only once. Constraints (5.5) and (5.6) imply that from each node on a route, a vehicle may travel to only one of the other nodes, and each node on a route can be traveled from only one node by vehicle *k*. Constraint (5.7) ensures that each node can have just one of the three possible states: either the node is on the route, the node is not on the route but is being served partially by one of the nodes on the route, or the node is not on the route and is not served at all. Constraint (5.8) implies that if node *i* is served by vehicle *k*, then vehicle *k* must be used. Constraint (5.9) implies that each node can receive the service indirectly from one of the other nodes. Constraints (5.10) and (5.11) ensure that the time and cost of traveling and the cost of the vehicle's assignment, respectively, do not exceed the budget. Constraint (5.12) implies that if node *j* is getting service from node *i*, then node *i* must be on a route. Constraint (5.13) limits the distance between the serving node and the served node. Constraint (5.14) is the subtour elimination constraint for each route.

The MTSPWPC has $2\,[I]^2[K] + 3[I][K] + [I] + [K]$ variables, where $[I][K]$ are integers and the remaining are binary variables. Moreover, there are $2\,[I]^2[K] + 5[I][K] + 2[I] + 3[K] + 1$ constraints.

Proposition 1. The complexity of the MTSPWPC is $\sum_{y=1}^{m} \binom{n}{x_1,\dots,x_y} \frac{(x_1-1)!}{2} \times \dots \times \frac{(x_y-1)!}{2} \times 2^{n-(x_1+\cdots+x_y)}$, such that $x_1 + \cdots + x_y \leq n$, $y \in (1,..,m)$, where *m* is the maximum number of vehicles, and *n* is the number of nodes.

Proof:

Because this is a selective problem, subject to budget, there can be up to *m* number of vehicles serving the nodes. The number of visited nodes by a typical vehicle, denoted by *y*, can be a variable denoted by x_y . The number of different ways that *n* nodes can be assigned to *y*

vehicles is $\binom{n}{x_1,\ldots,x_y}$. Each vehicle can have $\frac{(x_y-1)!}{2}$ Hamiltonian cycles for an asymmetric distance matrix. For the remaining nodes that are not served directly (i.e. $n-(x_1+\cdots+x_y)$), there are two possibilities: either assigned to one of the nodes on the route or not. Then, there are $2^{n-(x_1+\cdots+x_y)}$ cases of assigning nodes. Since the number of vehicles can be different and the range is between 1 and *m*, in total, there are $\sum_{y=1}^{m}\binom{n}{x_1,\ldots,x_y}\frac{(x_1-1)!}{2}\times\ldots\times\frac{(x_y-1)!}{2}\times 2^{n-(x_1+\cdots+x_y)}$ ways to be evaluated.

Proposition 2. The MTSPWPC is NP-hard.

Proof:

The TSP is NP-hard [8]. In an asymmetric TSP, there are $\frac{(n-1)!}{2}$ cycles to be evaluated to find the best Hamiltonian cycle. Complexity of the MTSPWPC is $\sum_{y=1}^{m}\binom{n}{x_1,\ldots,x_y}\frac{(x_1-1)!}{2}\times\ldots\times\frac{(x_y-1)!}{2}\times 2^{n-(x_1+\cdots+x_y)}$, which is greater than $\frac{(n-1)!}{2}$ cycles. Therefore, the MTSPWPC is NP-hard.

Because this problem is NP-hard and it is difficult to solve large-scale problems, two genetic algorithms are designed to solve it. Details of the algorithm are explained in section 5.3.

5.3 Genetic Algorithm

To solve this large-scale problem, we propose two new genetic algorithms (i.e., GA-2OPT and GA-IR), which have initial solutions based on structured chromosomes that are generated using a sweep algorithm [16]. The number of routes (i.e., number of vehicles used to service the nodes) is not fixed and depends on the total budget, since there is a fixed cost associated to the addition of a route. Chromosomes are selected based on a roulette wheel selection for the cross-over operation. The fitness of each chromosome is a function of the maximum profit that can be gained from a route.

To conserve diversity, mutation is implemented. Two different mutation approaches are presented. The first is a combination of two different mutation methods, in-route mutation and cross-route mutation [17]. Here, some stops of each individual route are swapped, while in the second method, some parts of randomly selected routes are exchanged with each other. The second method is a cross-route and 2-OPT greedy mutation, which attempts to improve a given route locally based on a greedy 2-OPT algorithm in order to save cost [18]. After implementing mutation, an insertion is performed on each route out of the nodes that are not served. This insertion is based on the closeness of the polar angle of those nodes to the average polar angle of nodes on the route. The framework of this proposed genetic algorithm is presented in Table 5.1.

Table 5.1. Genetic Algorithm for MTSPWPC

Step	Pseudo Code of Proposed Genetic Algorithm
1	Set number of generations *N* and the population size
2	Set selection probability p_s, crossover probability p_c, and mutation probability p_m
3	Generate initial population based on sweep algorithm
	3.1 Check feasibility of each chromosome
	3.2 Modify infeasible chromosome to obtain feasible chromosome
4	While (the number of iterations is less than *N*)
	4.1 Evaluate each chromosome according to its fitness function
	4.1.1 Solve assignment problem to find fitness for each path
	4.2. Select chromosomes according to their fitness using roulette wheel selection until pool is full.
	4.3 Execute crossover operation
	4.3.1 Check feasibility of each chromosome
	4.3.2 Modify infeasible chromosome to obtain a feasible chromosome
	4.4 Execute mutation
	4.4.1 Find new order of cities
	4.4.2 Add more cities regarding their polar angles subject to released budget
5	End

By setting the number of generations, the population size of each generation, selection, crossover, and mutation-rate parameters, the proposed genetic algorithms find a good solution in a reasonable amount of time.

5.3.1 Initialization

Several chromosome representations can be used for the MTSP. The first approach is a single chromosome technique [19] to capture all routes for all vehicles. The length of each chromosome is $m+n-1$, where m is the number of salesmen, and n is the number of locations. Each tour is represented by a permutation of the number of cities separated with a negative integer from the other route. The sequence of nodes in each route is the sequence of node visitations for each traveler.

Another route representation may utilize two chromosomes at the same time [20, 21] with a chromosome that has a length of n. The first chromosome is a permutation of nodes, and the second chromosome assigns a salesman to each node at the same location of the first chromosome.

Another approach for route representation has a two-part chromosome representation [15]. This method reduces the search space by eliminating redundant solutions. The first n elements are the permutation of n cities and the last m elements contain the number of nodes visited by each traveler sequentially.

Finally, multi-chromosome representation can be utilized for representing a solution [22]. Here, each solution or individual contains more than one chromosome, and each chromosome represents a route for a particular salesman.

In this algorithm, we use a multi-chromosome representation in which every route (vehicle or salesman) is represented by a chromosome, and there is a vehicle ID associated with each route.

To initialize the chromosomes, the cost of adding a traveler must be considered. We use the sweep approach proposed by Gillett and Miller [16] to construct the routes. In implementing the sweep method, the polar angles of all nodes are calculated. Next, for a particular salesman, a node is randomly assigned to a salesman. The order of nodes on the route is based on the polar angles. This process will be done until the time limit is violated. Adding a new route is implemented if the total cost of traveling in previous routes plus the fixed cost of routes does not exceed the total budget.

5.3.2 Fitness Evaluation and Selection

In a genetic algorithm, each solution is evaluated based on the fitness function to select the chromosomes for crossover and mutation. Fitness is a function of the problem objective, which is the maximum number of customers served as follows:

$$f = \exp(obj) \tag{5.16}$$

where f denotes the fitness of the chromosome, and obj represents the maximum number of served customers of the tour.

The maximum number of served nodes comes from optimally assigning nodes that are not on the route to nodes that are on route. This is an optimization problem where a set of nodes should be assigned to another set of nodes with a certain serving capacity. Furthermore, the distance of nodes assigned to a node on route cannot be more than a predetermined distance.

To find the maximum number of served customers, the following assignment problem is solved as a mixed-integer linear program:

Parameters

$$\overline{z_i} = \begin{cases} 1 & if\ node\ i\ is\ on\ any\ given\ route \\ 0 & o.w. \end{cases}$$

Variables

$$hi = \begin{cases} 1 & if\ node\ i\ is\ on\ any\ route\ but\ served\ indirectly \\ 0 & o.w. \end{cases}$$

$$q_{ij} = \begin{cases} 1 & if\ node\ j\ is\ assigned\ to\ node\ i \\ 0 & o.w. \end{cases}$$

Model

$$\max \sum_{i=1}^{n} D_i \bar{z_i} + \sum_{j=1}^{n} \sum_{i=1}^{n} \mathrm{E}_j q_{ij} \tag{5.17}$$

Subject to

$$\bar{z_i} + h_i + v_i = 1 \qquad \forall\, i \in I \tag{5.18}$$

$$\sum_{i=1}^{n} q_{ij} = h_j \qquad \forall j \in I \tag{5.19}$$

$$\bar{z_i} \geq q_{ij} \qquad \forall i, j \in I \tag{5.20}$$

$$\sum_{j=1}^{n} E_j q_{ij} + D_i \bar{z_i} \leq CAP_i \bar{z_i} \qquad \forall i \in I \tag{5.21}$$

$$L_{ij} q_{ij} \leq DIS_j \qquad \forall\, i, j \in I \tag{5.22}$$

$$q_{ij}, hi: binary\ variables$$

Constraint (5.18) ensures that each node takes only one of three possible states: the node is on the route, the node is not on the route but is being served partially by one of the nodes on the route, or the node is not on the route and is not served indirectly. Given a chromosome, we already know which nodes are on the route, i.e., $\bar{z_i}$ is known. Constraint (5.19) ensures that each node can be served indirectly by only one of the other nodes that is on the route. Constraint (5.20) implies that if node *i* is served by node *j*, then node *j* must be on the route (otherwise it cannot serve node *i*). Constraint (5.21) restricts the number of customers served directly and indirectly in each node by the capacity of that node. Constraint (5.22) limits the distance between the serving node and the served node. Note that $\bar{z_i}$ are known for each path and are fixed. In this problem, there are $2\,[I] + [I]^2$ binary variables and $3\,[I] + 2[I]^2$ constraints. Compared to the original problem, there

are $2\,[I]^2[K] + 3[I][K] - [I] + [K] - [I]^2$ less number of variables and $2\,[I]^2[K] + 5[I][K] - [I] + 3[K] - 2[I]^2 + 1$ less number of constraints.

After computing the fitness, the roulette wheel selection is implemented to select the parents to crossover.

5.3.3 Crossover

To implement the crossover operation, since the number of routes are different in each solution given the fixed cost of utilizing additional vehicles, the crossover is done among individuals that have the same number of routes. The parent with the best fitness would be chosen as the first parent. Then a two-point crossover is performed to generate the offspring. In this crossover, the parents must be transformed to a new chromosome representation, whereby each chromosome includes the vehicle number serving the city located in the corresponding sequence of genes. For instance, in ch_1^p, the second gene is 1, which implies that the second city is served by vehicle 1, and because the seventh gene is 0, i.e., city 7 is not served (Figure 5.1).

The key point is to number the vehicles according to the polar angle, which may imply that all cities in the same area are served by the same vehicle. For the first parent, vehicle numbering will be performed sequentially, depending on the number of routes existing in that solution. To number the vehicles of a second parent, the average coordinates of visited cities by each vehicle are calculated for both parents, and the polar angles of those points are calculated. To number the vehicles of the second parent, the difference between the polar angles measured from the visited cities by that vehicle to the first parent vehicles is calculated. Vehicles of the second parent are numbered according to this difference (i.e., a vehicle that has a smaller difference between the polar angles is numbered the same as the first parent vehicle) Then, two points are randomly generated to determine the crossover points. The first offspring inherits the genes of the first parent

located to the left side of first crossover point and to the right side of the second crossover point, while receiving the genes of the second parent located between the two crossover points. The second child gets the genes opposite of the first offspring. After generating the offspring, the chromosomes must be recoded again, in order to have the same representation as for the other chromosomes. Note that, in initialization, cities are sorted based on their polar angles. Then, transition to the previous path representation can be performed easily.

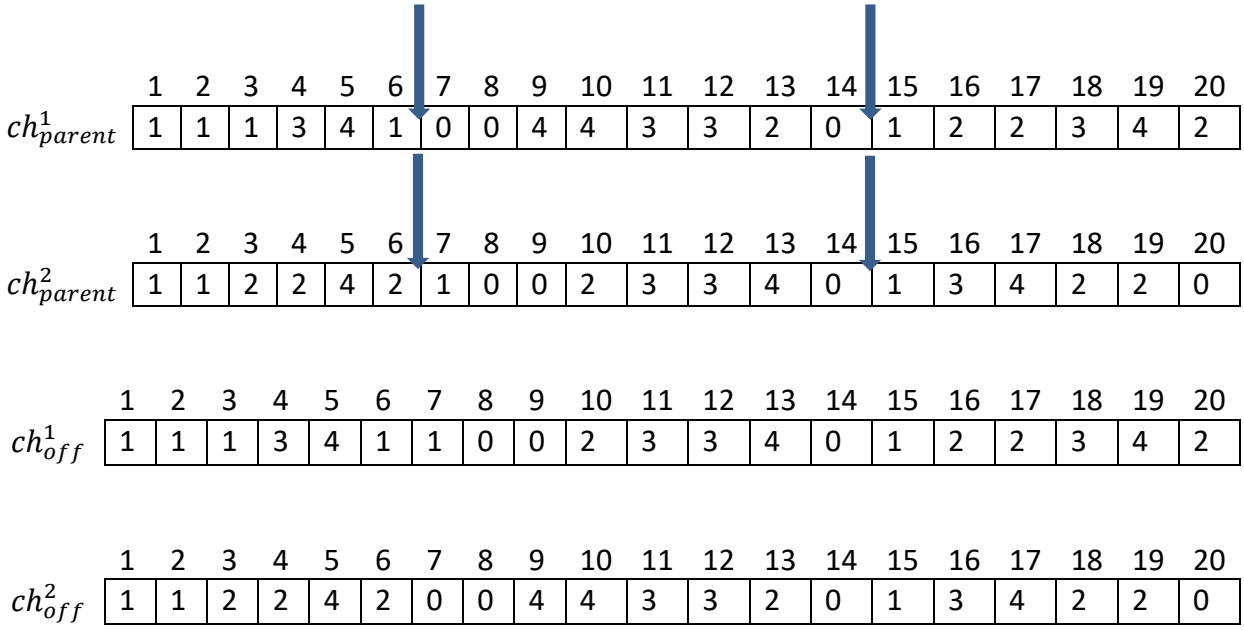


Figure 5.1. Crossover

After creating the offspring, the time limit and budget constraints should be checked with respect to feasibility. In the case of infeasibility, some cities are dropped randomly from the current route, in order to achieve feasibility. The infeasibility of repeated nodes on the same route cannot occur in this new representation of chromosomes, because each node can receive service from a vehicle only once or it will not receive service at all.

5.3.4 Mutation

We propose two different mutation methods: “cross-route and in-route” mutation, and “cross-route and greedy 2-OPT” mutation. In the first type of mutation, there are some exchanges

of visited locations among the routes and also order of locations in a route. In the second method, in addition to some exchanges among the routes, for each route, a greedy 2-OPT algorithm is implemented to improve the solution.

5.3.4.1 Cross-Route and In-Route Mutation

We implement the cross-route and in-route mutation to improve the current solution by using the list of current directly served nodes, and then we add off-route nodes if this does not cause any budget violation.

In the cross-route operation (Figure 5.2), two different parts of two randomly chosen routes in a given solution are switched; if the result is infeasible due to exceeding the time limit or budget, then we split the infeasible route into two routes, as long as this does not violate the budget constraint. After swapping different parts of the routes and creating new routes, the feasibility check of each route is performed, and if the solution is not feasible due to a time limit violation, some nodes will be dropped off the route, in order to make it feasible.

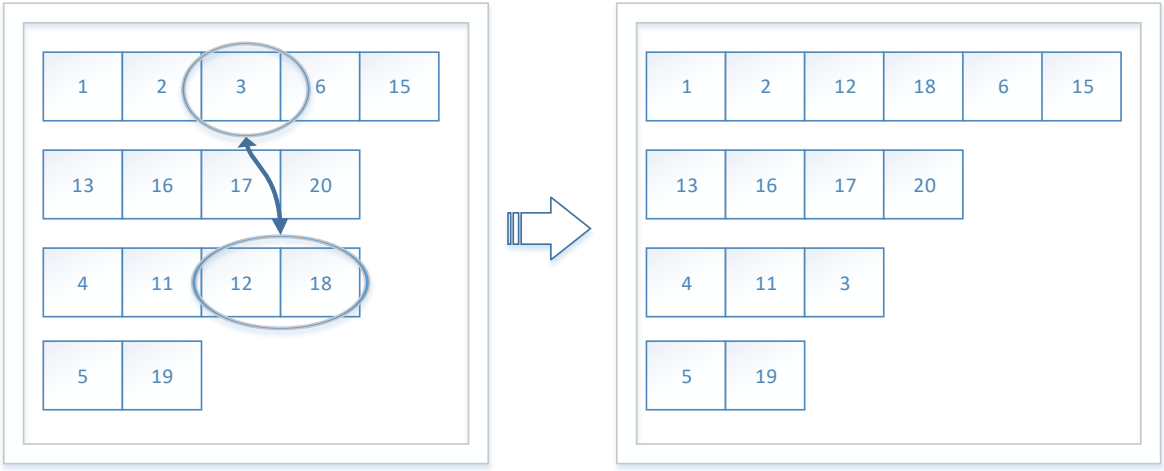


Figure 5.2. Cross-route mutation

Afterwards, an in-route mutation is implemented as well to improve the route itself. In this mutation, a regular swap mutation is implemented in a given route. Two random stops of a route are selected and swapped. If the result is feasible regarding time limit, then the solution is kept.

During the addition of off-route nodes operation, which is after implementing the cross-route and in-route operations, the polar angles of nodes that are not served are calculated, and a node with the closest polar angle to the average polar angle of the nodes on the route is added. Thereafter, a feasibility check is performed, and if the solution is not feasible, then a repair-feasibility operation is performed, and some locations may be dropped off the route until the solution becomes feasible.

5.3.4.2 Cross-Route and Greedy 2-OPT Mutation

Instead of an in-route mutation, a greedy 2-OPT algorithm that considers pairwise exchanges is performed. It starts with exchanging nodes 1 and 2. If the exchange improves the solution, then the exchange is implemented; otherwise, it will evaluate the exchange of nodes 1 and 3, and so on until the exchange improves the solution. Afterwards, it will start with node 3, and the procedure is repeated until whole pairwise comparisons are done and no improvement is no longer possible. In the greedy 2-OPT algorithm, the exchange is permanent. Hence, it is faster than the 2-OPT algorithm. In addition, it gives a good solution close to the 2-OPT algorithm, and the computational time is considerably less than the 2-OPT algorithm.

In this mutation method, we implement a greedy 2-OPT algorithm on the solution gained from crossover to improve the solution, if possible. Within the released budget, an attempt is made to add some tasks to the route. First, the polar angles of nodes that are not served are calculated, and the average polar angle of nodes that are on the route is calculated as well. Then a node with a polar angle closer to the average polar angle of the nodes on the route is added, and feasibility constraints are checked thereafter.

Table 5.2. Greedy 2-OPT Algorithm

Step	Pseudo Code of Greedy 2-OPT operation
1	Perform 2-OPT operation for each route of selected chromosome.
	1.1 Exchange node *i* and j, $i<j$. Compare resulting z with the current z^*. If $z > z^*$, j<n then repeat step 1.1; otherwise, go to 1.2.
	1.2 If $z < z^*$, $j = j + 1$ and go to step 1.1. If $z > z^*$ and $j = n$, set $i = i + 1$, $j = j + 1$ and go to step 1.1; otherwise, the current solution is the best solution and terminate the procedure.
2	Insert a node to the optimized solution.
	2.1 Calculate the polar angles of nodes not served and the average polar angle of routes.
	2.2 For each route, add a node that has the closest polar angle to the average polar angle of the route.
	2.3 While the solution is feasible, go to step 2.1.
3	End

5.4 Computational Results and Analysis

In this chapter, we use two different genetic algorithm approaches. The first, GA-IR, uses "cross-route and in-route" mutation, and the second, GA-2OPT, uses "cross-route and greedy 2-OPT" mutation. Proposed GAs are implemented in Scala programming language [23]. Problem instances are randomly generated and solved to optimality using a model in the GAMS [24] linked with the IBM CPLEX [25] solver. Furthermore, the same instances are solved using the proposed metaheuristics. The metaheuristic solutions are compared to the optimal solutions.

5.4.1 Selection of Control Parameters

In order to have an efficient genetic algorithm, several parameters should be fine-tuned. The main parameters are as follows:

- Size of population (sp) implies how many chromosomes are in the population in any generation.
- Selection probability (p_s) indicates how often crossover is performed.
- Mutation probability (p_m) indicates how often parts of the chromosome are mutated.
- Elitism probability (p_e) indicates the rate of saving the best solutions of previous generation in the new generation.

The probabilities are set such that the sum of p_s, p_m, and p_e are equal to one. The number of computational cycles (N_{max}) is set such that the final solution is no longer changed, and the CPU time is practical. The GA is run five times for each parameter combination (see Table 5.3).

Table 5.3. GA Control Parameters Set

Control Parameters	**Values**					
Size of Population (sp)	100,50,30	100,50,30	100,50,30	100,50,30	100,50,30	100,50,30
Selection Probability (p_s)	0.7	0.65	0.6	0.55	0.50	0.50
Mutation Probability (p_m)	0.2	0.25	0.3	0.35	0.4	0.45
Elitism Probability (p_e)	0.1	0.1	0.1	0.1	0.1	0.05

The fine tuning results in parameter selection that provides the highest profit. For different-size MTSPWPC problems, the proposed parameters for both GAs are optimized, as shown in Table 5.4. Note that for larger problem sizes, the number of GA iterations increases and the population size decreases. The selection and crossover parameters also vary slightly. Furthermore, larger problem size requires higher mutation probability.

Table 5.4. Optimized GA Parameters

Scale of TSP	N_{max}	sp	p_s	p_m	p_e
$n \leq 50$	30	100	0.6	0.3	0.1
$50 < n \leq 90$	40	100	0.55	0.35	0.1
$90 < n \leq 200$	60	30	0.55	0.40	0.05

5.4.2 Experimental Design

The web-based geographic midpoint calculator, Geomidpoint.com is used to generate the problem data in order to have different distributions of potential customer locations to estimate the performance of the proposed algorithms. With this software, by providing the latitude and longitude of the depot, cities are randomly generated around the depot within a certain distance. The distance between cities is calculated using the Euclidean distance

Each experiment is solved using three different methods: GA-AS, GA-SW, and integer linear program on a computer with 8 GB RAM, Core™ i7 CPU, and 465 GB hard disk space.

The efficiency of the proposed metaheuristics is evaluated using ten random problems. A predetermined demand and capacity, which are constant and equal, are assigned to each node. The cost of traveling is assumed to be $0.15 per mile. The budget amount is set such that the computational time for solving the problem with CPLEX is large. In some cases, the upper limit of the CPLEX solution is captured, where CPLEX fails to solve the problem to optimality due to memory problems. Moreover, each instance is solved considering a small or large fixed cost of utilizing a vehicle.

For each experiment (i.e., problem instance), the problem has been solved using two genetic algorithms, GA-IR and GA-2OPT, and the CPLEX solver (especially for small-size problems). For checking the quality of the solutions using GA-IR and GA-2OPT, solutions

obtained by these metaheuristics are compared with the exact solution found by the mixed-integer linear programming by calculating the relative error:

$$RE = \frac{optimal\ profit\ /upper\ bound - best\ profit}{optimal\ profit\ upper\ bound} \tag{5.22}$$

As it can be seen in this formula, the relative error is calculated relative to the upper bound of the optimal profit, because in some cases, the problem cannot be solved to optimality by CPLEX. In this case, the upper bound of the optimal profit calculated by CPLEX is used instead. In the case of finding an optimal solution, the optimal profit is used instead of its upper bound.

The genetic algorithms are run five times, and the average profit and best profit are recorded. In general, over the same number of generations, the GA-2OPT algorithm is providing an optimal solution faster than the GA-IR algorithm for random distributed nodes. GA-2OPT uses a greedy 2-OPT algorithm to improve the routes, while GA-IR simply swaps some random nodes in a route, which does not necessarily improve the order of nodes. Therefore, GA-2OPT performs better over the same number of computational cycles. Moreover, the amount of fixed cost can affect the amount of profit that can be gained. As the fixed cost amount increases, the number of vehicles on the route decreases, and fewer routes will be constructed.

5.4.3 Numerical Results

The sections will present investigative findings for the following: (a) the impact of budget amount on the optimal solution, (b) the impact of problem size on GA-IR and GA-2OPT performance, and (c) the impact of fixed cost on GA-IR and GA-2OPT performance.

5.4.3.1 Effect of Budget Amount on Optimal Solution for 16-Node Instance

For a 16-node problem instance, we investigate the impact of the budget amount on the optimal solution and CPU time needed by CPLEX to determine the optimal solution. First, we calculate the minimum budget required to visit all nodes as a multiple traveling salesman problem

($4,824), which also provides an upper bound to the maximum profit. We then limit the total budget for each instance as a percentage of this budget (e.g., 10% to 90% of $4,824) to determine the total profit (i.e., served demand), as shown in Table 5.5.

As shown in Table 5.5, the most difficult instance of the problem is when the budget is around 60% of the required budget for visiting all nodes. All other instances have shorter CPU times, which shows that they are easier instances to solve. After the 80% budget case, there is a rise in computation time, which could be because of having more vehicles, which may increase the complexity of the problem.

Table 5.5. Profit and Execution Time for Exact Method under Different Budget Amounts (Budget = $4,824)

Percent of Minimum Cost	**Profit**	**Number of Served Nodes**	**Number of Indirectly Served Nodes**	**Number of Not-Served Nodes**	**Number of Vehicles on Route**	**CPLEX Time (sec)**
10	0	0	0	16	0	1.706
20	30	3	3	10	1	1.868
30	75	5	5	6	1	2.287
40	105	7	7	2	1	4.886
50	125	9	7	0	1	3.758
60	125	9	7	0	2	30.359
70	135	11	5	0	2	3.769
80	145	13	3	0	2	2.961
90	150	14	2	0	2	6.176
100	160	16	0	0	2	18.723

The changes in profit while solving each problem via CPLEX using branch and bound is also evident in Table 5.5. As can be expected, increasing the budget percentage results in gaining more profit by virtue of having a larger feasible region. As it can be seen, after increasing the budget to 60%, the number of vehicles increases and, consequently, profit amount grows.

The increase in budget results in visiting more nodes on each route. Consequently, the number of nodes served indirectly increases as well. As it can be seen in Table 5.5, after visiting more than half of the population (i.e., number of served nodes is nine) the number of not-served nodes drops to zero.

5.4.3.2 GA Evaluation for Randomly Dispersed Instances

To evaluate the performance of both GAs, ten different-size randomly dispersed instances are generated. For each instance, a low and high fixed cost scenario is run. The budget amount is set such that it is computationally expensive to solve the problem with CPLEX. In some cases where CPLEX fails to solve the problem, the upper bound of the CPLEX solution is captured. The largest problem (RA-140) takes 95.370 seconds to be solved in Scala. When the fixed cost is small, it is expected to have the option of using more vehicles, which results in more profit. Having a large fixed cost may result in using fewer vehicles and collecting less profit.

The results of different size instances with small fixed cost is presented in Tables 5.6 and Figure 5.3. Moreover, the results of different size instances with large fixed cost is presented in Tables 5.7 and Figure 5.4. The optimal profit is also gained by running GAMS using CPLEX solver to calculate the gap of each solution from the optima and to evaluate the efficiency of each algorithm. Furthermore, the trend of convergence of each instance is shown in Figures 5.3 and 5.4, and the trend of absolute error (i.e., difference between optimal profit and best profit of GAs) of some instances is shown in Figures 5.5 and 5.6. For most cases, the GA-2OPT algorithm provides a closer solution to optima.

Table 5.6 shows the average profit and best profit obtained by running each GA. Both GAs are providing good solutions based on relative error. In large-size problems, the optimality gap of GA-2OPT is less than the optimality gap of GA-IR. Moreover, the average profit of GA-2OPT is

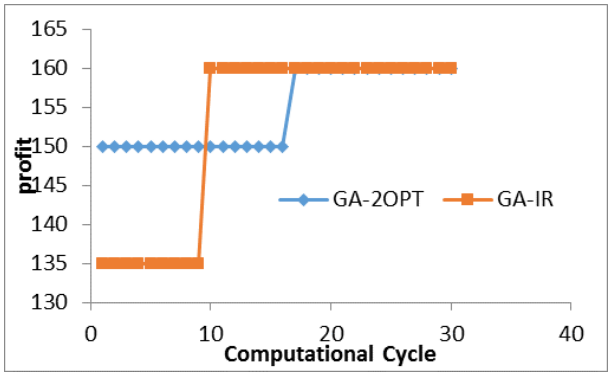
greater than the average profit of GA-IR. This may be due to the faster convergence of GA-2OPT to a better solution.

Table 5.6. Results of GA Approaches for Different-Size Instances (Small Fixed Cost)

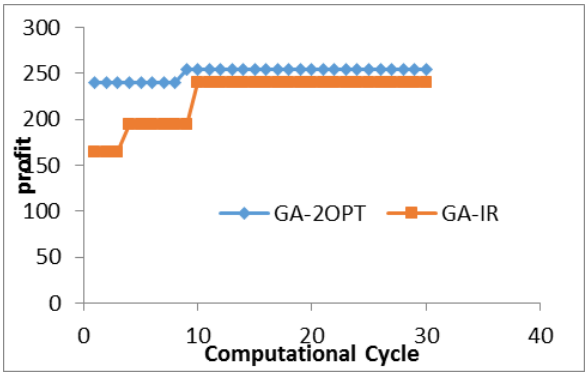
Problem	Average Profit		Best Profit		Optimal Profit	RE (%)		CPLEX CPU Time (sec)
	GA-2OPT	GA-IR	GA-2OPT	GA-IR		GA-2OPT	GA-IR	
RA-16	154.67	152.5	160	160	160	0	0	18.357
RA-32	251	223.5	255	240	255	0	0.05	124,514.829
RA-50	424.33	408.5	425	410	430	0.01	0.046	20,061
RA-60	507.125	458.25	510	460	520	0.019	0.1	22,130.885
RA-70	564.125	543.875	565	545	565	0	0.03	28,988.546
RA-80	588	568.875	610	600	625	0.02	0.04	6,673.753*
RA-90	740.875	738.3	750	745	860	0.1	0.1	22,372.742*
RA-100	893.7	852.8	895	860	920	0.03	0.06	7,609.923
RA-120	962.1	957.8	965	960	990	0.02	0.03	13,953.82*
RA-140	1044.9	1036.4	1065	1060	1100	0.03	0.04	38,626.773

* indicates CPLEX out of memory

Figures 5.3(a) to 5.3(j) also show that GA-2OPT is providing a better solution in the long run. GA-IR is delivering good solutions for most cases, but the solutions of GA-2OPT are considerably better than the solutions of GA-IR. Because they both use the same method to initialize solutions, they have the same diversity. However, GA-2OPT has the benefit of greedy 2-OPT improvement. Hence, the quality of its solutions improves much faster than those of GA-IR.

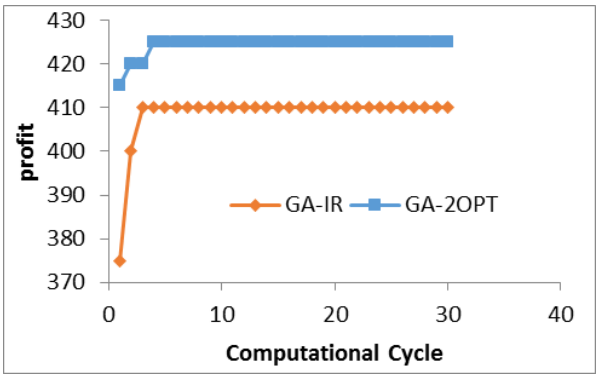


(a) RA-16

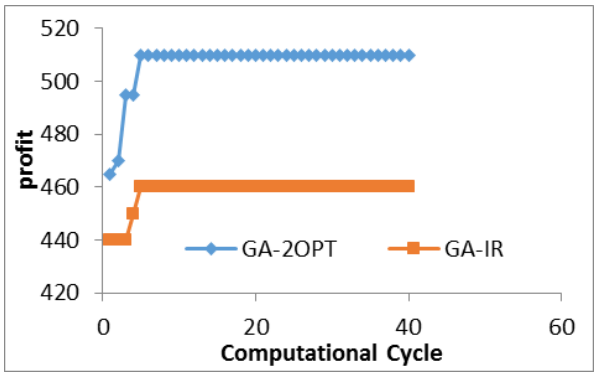


(b) RA-32

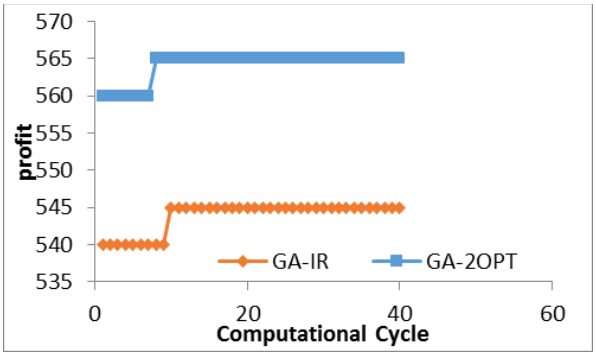
Figure 5.3. Trend in profit changes over different generations for instances with small fixed cost



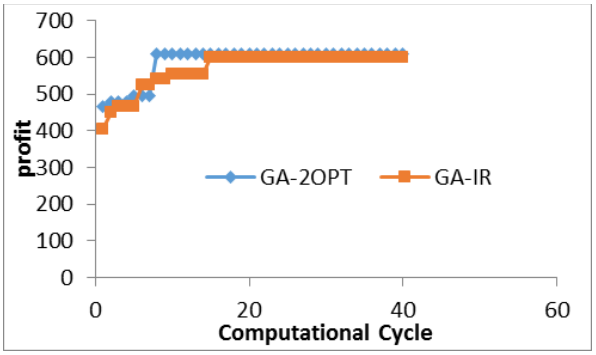
(c) RA-50



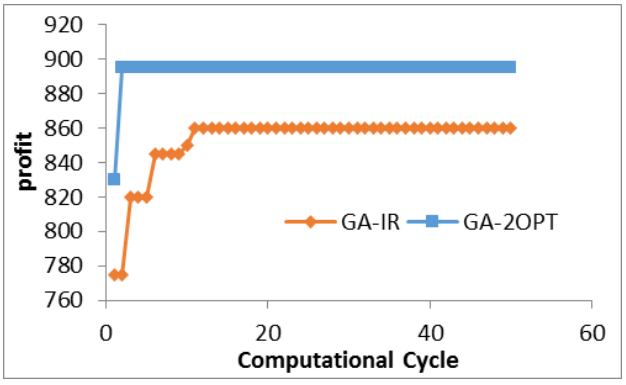
(d) RA-60



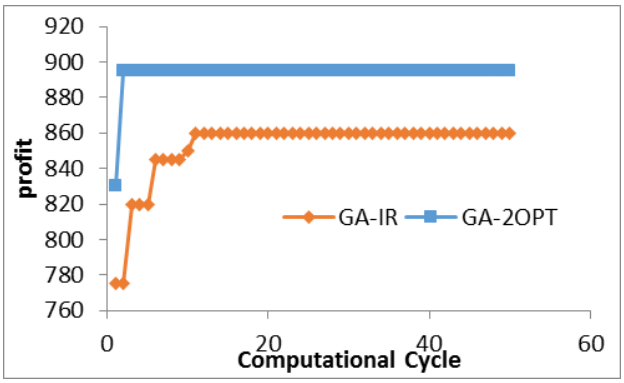
(e) RA-70



(f) RA-80

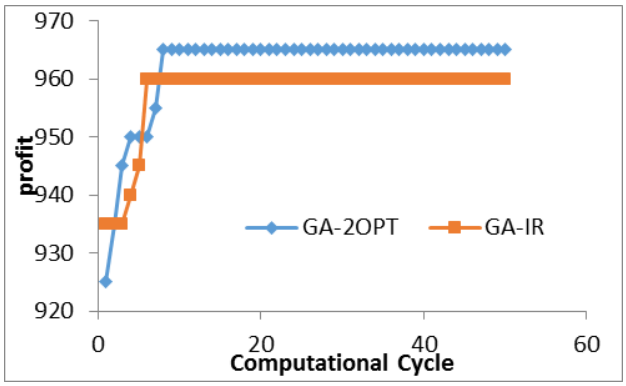


(g) RA-90

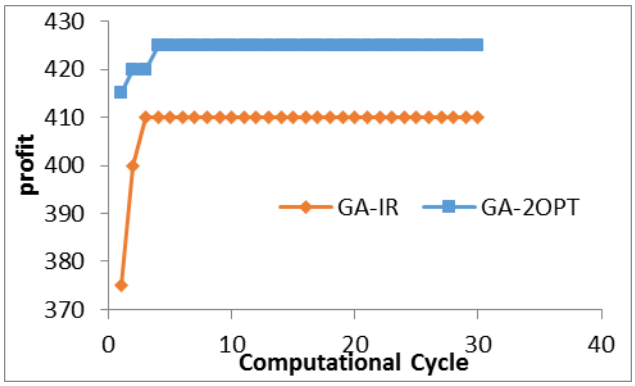


(h) RA-100

Figure 5.3. (continued)



(i) RA-120



(j) RA-140

Figure 5.3. (continued)

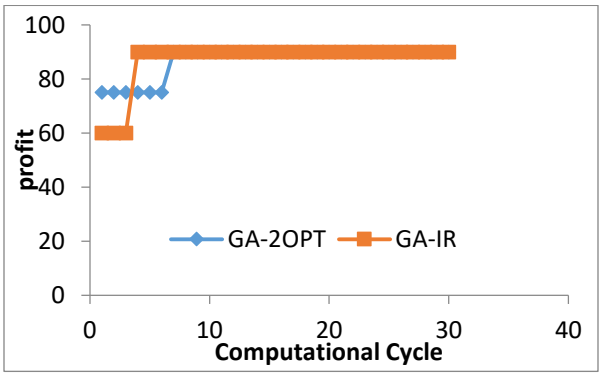
Table 5.7 shows the average profit and best profit obtained from running each GA with large fixed cost for each different instance. As the fixed cost increases, GA-2OPT is providing a closer solution to optima, because improving each solution is increasing the possibility of having more nodes on the route. Furthermore, the optimal profit of each instance with large fixed cost is less than the optimal profit of the cases with a small fixed cost, due to having a smaller budget to visit the nodes.

Table 5.7. Results of GA Approaches for Different-Size Instances (Large Fixed Cost)

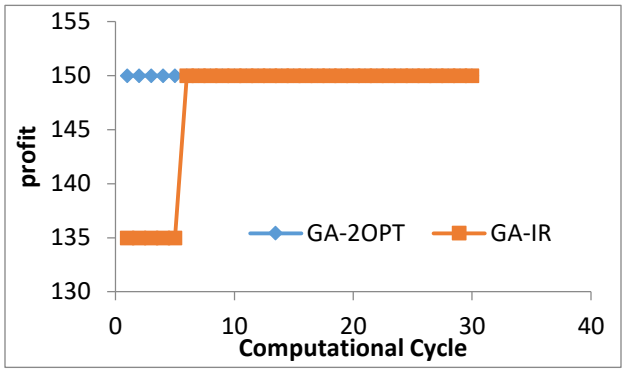
Problem	Average Profit		Best Profit		Optimal Profit	RE (%)		CPLEX CPU Time (sec)
	GA-2OPT	GA-IR	GA-2OPT	GA-IR		GA-2OPT	GA-IR	
RA-16	87	87	90	90	90	0	0	31.957
RA-32	150	147.5	150	150	150	0	0	98,266.118
RA-50	202.5	177.5	210	195	215	0.02	0.09	3,379.035
RA-60	299.75	299.75	300	285	300	0	0.05	652.242
RA-70	347	324.5	300	300	350	0	0	13035.015
RA-80	450.375	441.75	465	450	470	0.01	0.04	16,673.753
RA-90	585	549	600	555	600	0	0.07	7,078.195*
RA-100	736.2	692.1	750	705	750	0	0.06	5,655.106*
RA-120	779.1	750	780	750	820	0.05	0.08	3,731.656*
RA-140	963.9	912	975	915	980	0.005	0.06	38,495.223

* indicates CPLEX out of memory

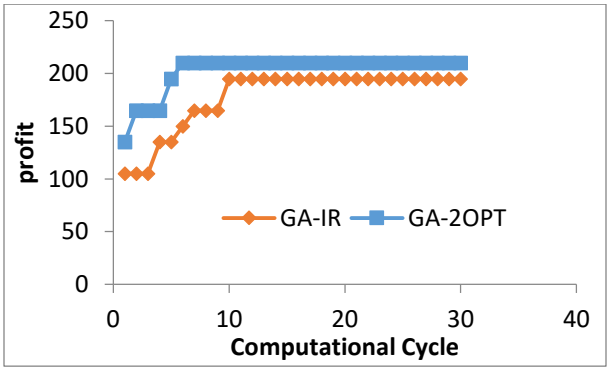
Figures 5.4a-5.4j also show that GA-2OPT is providing better solution in long-run for instances with large fixed cost. GA-IR is providing good solutions for most cases but the solution of GA-2OPT is considerably better than the solutions of GA-IR especially for instances with larger sizes.



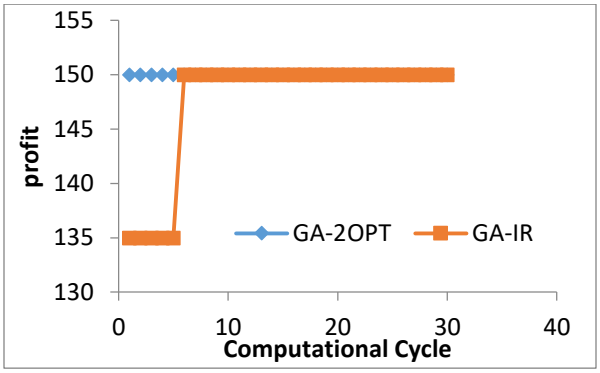
(a) RA-16



(b) RA-32

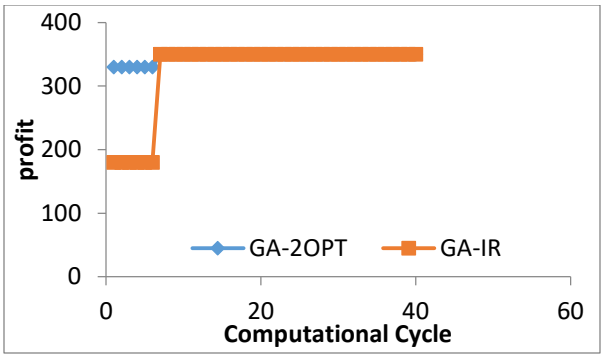


(c) RA-50

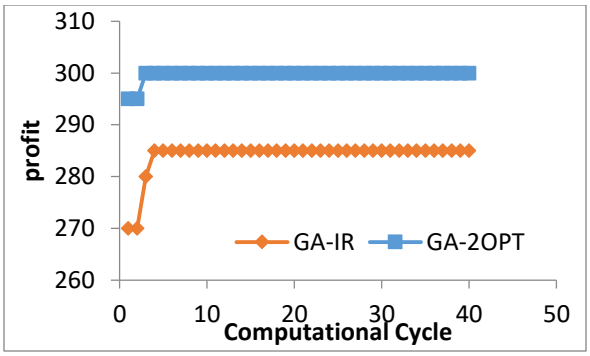


(d) RA-60

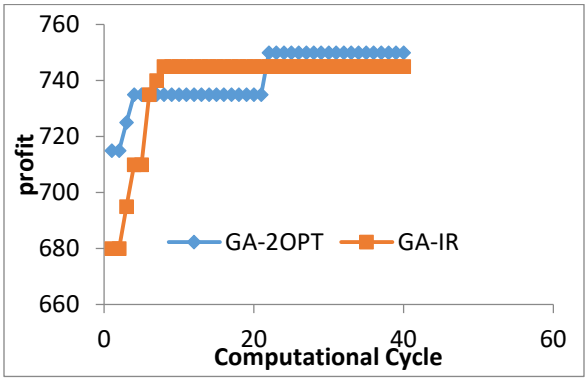
Figure 5.4. Profit changes trend over different generations for instances with large fixed cost



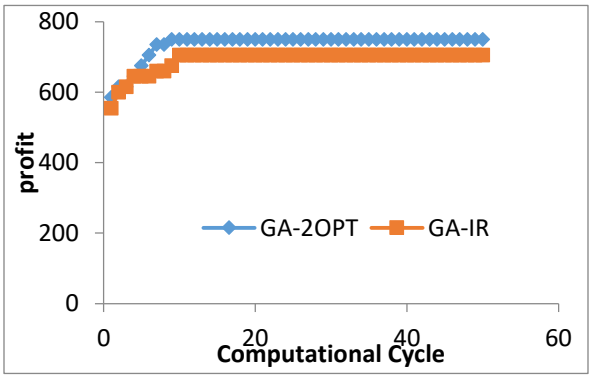
(e) RA-70



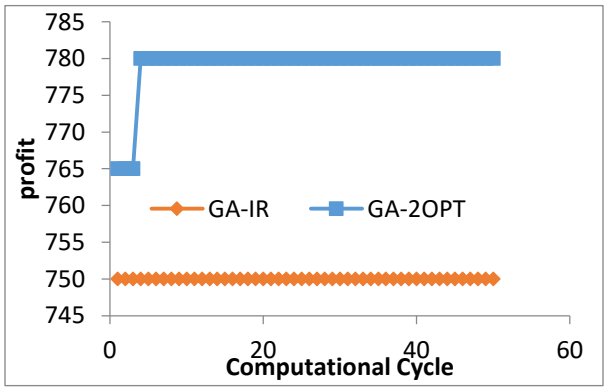
(f) RA-80



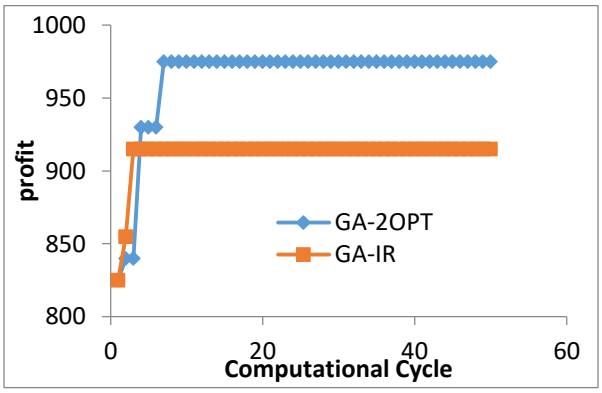
(g) RA-90



(h) RA-100



(i) RA-120



(j) RA-140

Figure 5.4. (continued)

As can be seen in Figure 5.5, the absolute error of different instances converges to a small number close to zero when the GA-2OPT algorithm is used. Moreover, the difference of absolute

error of both GAs for larger instances such as RA-100 is large, which implies that GA-2OPT is performing better as the size of the problem increases.

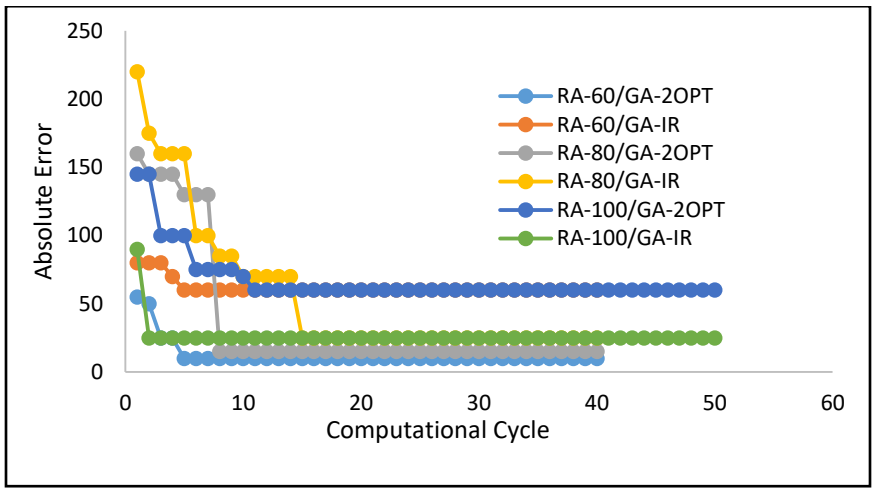


Figure 5.5. Trend in absolute error/small fixed cost

As it can be seen in Figure 5.6, the absolute error of different instances converges to zero when the GA-2OPT algorithm is used. Furthermore, the average absolute error of the GA-IR algorithm for most instances is greater than the average absolute error of the GA-2OPT algorithm. Because the GA-2OPT uses a greedy 2-OPT optimization to improve the solutions in mutation, the absolute error of initial solutions is lower than that of the GA-IR, and consequently the average absolute error of GA-2OPT solutions is less than that of GA-IR solutions.

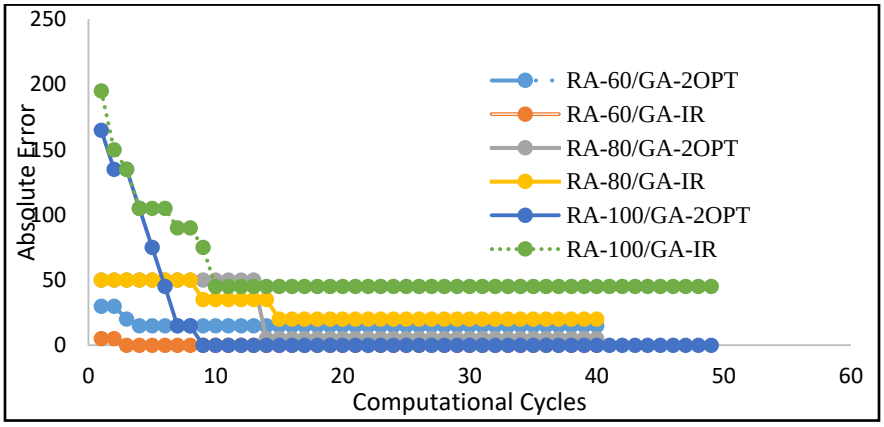


Figure 5.6. Trend in absolute error/large fixed cost

5.4.4.3 Evaluation of Effect of Fixed Cost on Final Result

To evaluate the effect of fixed cost amount on the quality of the solutions obtained from the GA-IR and GA-2OPT algorithms, one instance with 32 nodes (RA-32) is selected. The average total cost between two cities is $400. The fixed cost amount is varied in increments of $100 starting at $100 ending at $900. Profit is expressed in terms of served demand. The results of each algorithm are shown in Table 5.8. The average profit and best profit obtained from running each GA with a different fixed cost for each different instance along with the corresponding relative error are captured in this table. For most cases, GA-2OPT is providing a solution close enough to optima, and its performance is better than that of GA-IR.

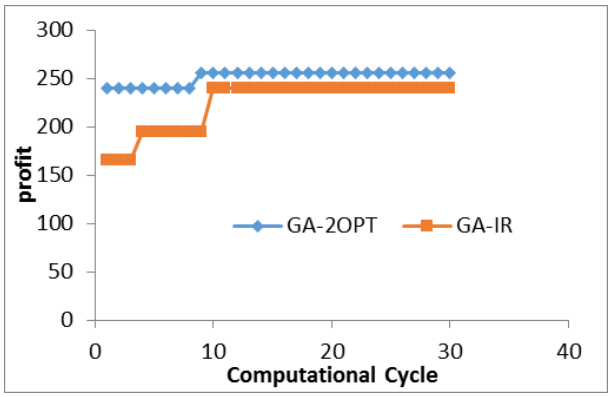
Table 5.8. Results of GA Approaches for Different Fixed-Cost Amounts (RA-32)

Fixed Cost Amount	Average Profit		Best Profit		Optimal Profit	Number of Vehicles	RE (%)		CPLEX CPU Time (sec)
	GA-2OPT	GA-IR	GA-2OPT	GA-IR			GA-2OPT	GA-IR	
100	251	223.5	255	240	255	2	0	0.05	129,734.749
200	240	210	240	210	255	2	0.05	0.1	98,266.118
300	226.5	224	240	225	250	2	0.04	0.1	335,158.062
400	231	216	240	240	240	1	0	0	42,061.90*
500	199	171	210	180	240	1	0.1	0.2	44,566.55*
600	202	172	210	180	210	1	0	0.1	49,595.786
700	172	147.5	180	165	195	1	0.08	0.1	441,711.269
800	165	165	165	165	165	1	0	0	338,596.628
900	150	147.5	150	150	150	1	0	0	124,514.829

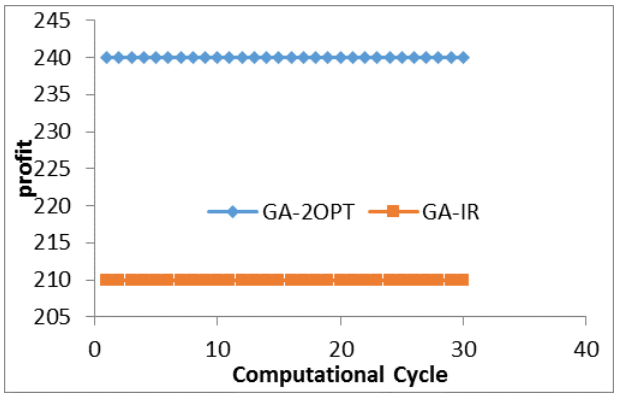
* Indicates CPLEX out of memory

As shown in Figure 5.7, for cases where the fixed cost has an impact on changing the optima, finding a good solution is becomes more challenging for both GAs. For example, switching from a fixed cost of $100 to a fixed cost of $200 increases the relative error for both

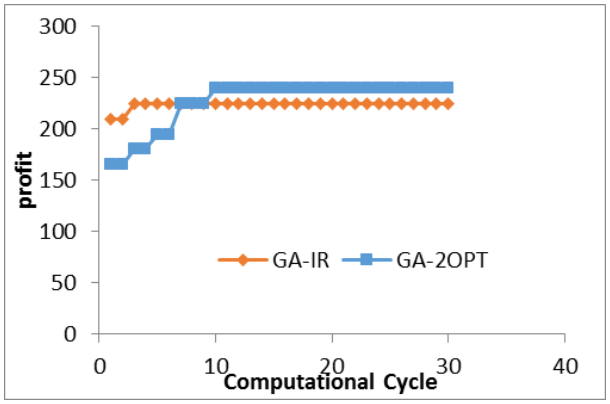
GAs. As shown, the same number of vehicles is used, and the same profit is obtained (i.e. problem becomes tighter). In this case, the GA-2OPT performance is better than the GA-IR performance. On the other hand, for instances with large fixed costs (i.e., $800 and $900), since the feasible region is shrinking, the performance of both GAs improves significantly (i.e., relative error for $800 and $900 is zero.)



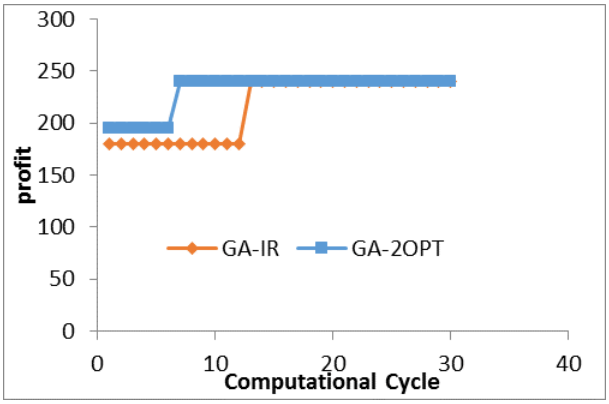
(a) RA-32-100



(b) RA-32-200

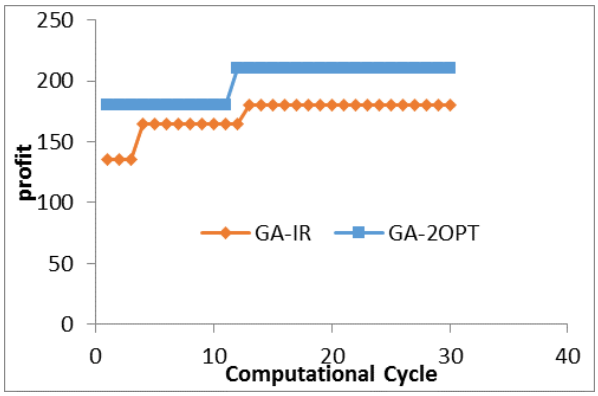


(c) RA-32-300

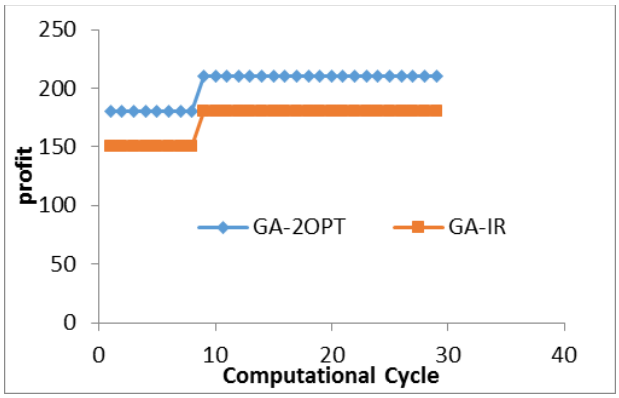


(d) RA-32-400

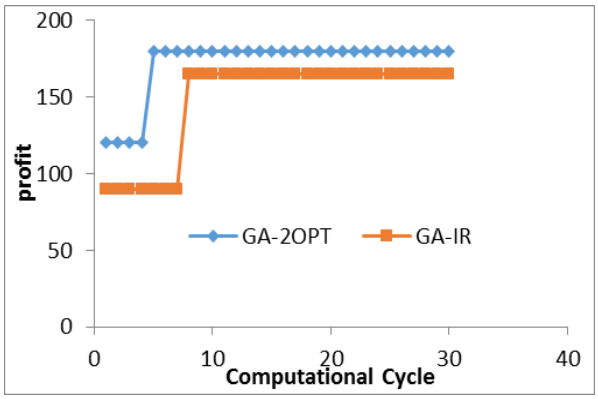
Figure 5.7. Trend of profit changes over different generations for instances with different fixed costs



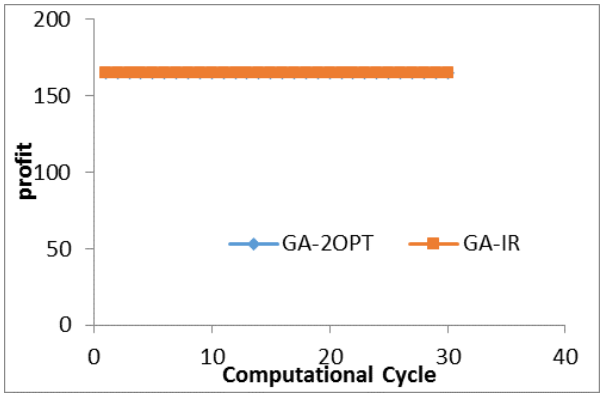
(e) RA-32-500



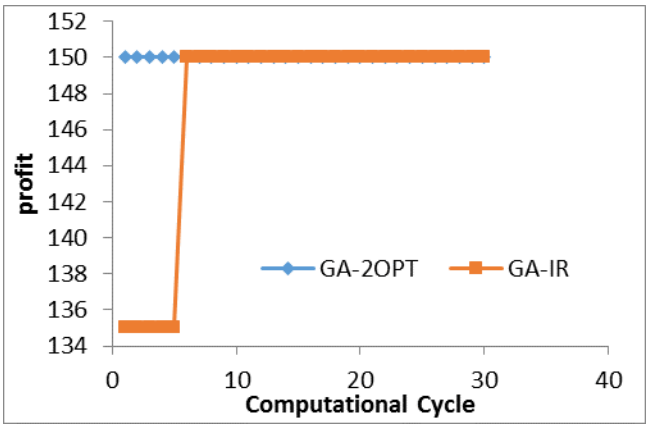
(f) RA-32-600



(g) RA-32-700



(h) RA-32-800



(i) RA-32-900

Figure 5.7. (continued)

As it was shown in Table 5.8 previously, the CPLEX time to find the optimal solution is considerably larger than the computational time for GAs to find a reasonable good solution. Figure 5.8 shows the trend of GA-2OPT and CPLEX while finding the optimal solution. CPLEX finds the optimal solution of 210 after 12,542 seconds (and verifies this solution in more than 20,000 seconds), while GA-2OPT reaches the same solution after 5.224 seconds.

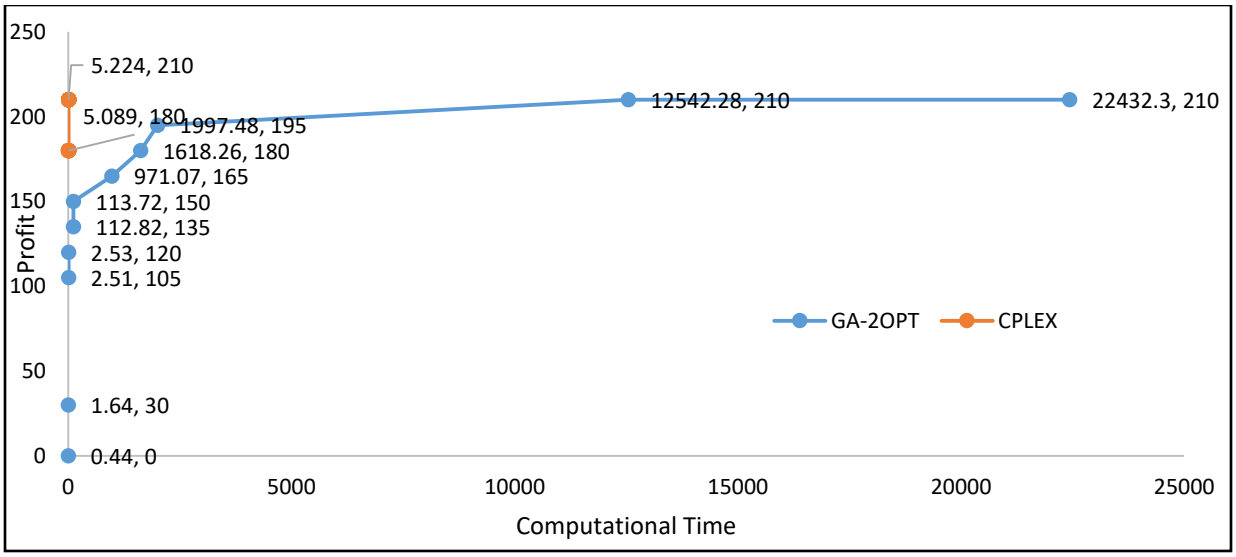


Figure 5.8. Computational time comparison between CPLEX and GA-2OPT

Regardless of the level of the fixed cost, GA-2OPT is providing a better solution and converges to optima faster than GA-IR. Note that both GAs initialize the pool of solutions using the sweep method, which provides a fair diverse pool of initial solutions. The greedy 2-OPT mutation improves the solutions at any iteration by providing a better sequence of nodes with lower travel cost/budget, thus providing an opportunity to add more nodes to the route, and consequently the number of served customers improves as well.

When the fixed cost increases, the feasible region tightens because of the tightening budget constraint. The performance of both proposed GAs improves; however, GA-2OPT can provide a closer solution to optima compared to GA-IR.

5.5 Conclusion

A new multiple traveling salesman problem with partial coverage is proposed. In MTSPWPC, the objective is to maximize the profit that can be gained from visiting nodes by finding a set of subtours for multiple salesman. Due to limited resources and the cost of a salesman assignment, the number of subtours is not fixed and entire nodes might not get served. Because this problem is NP-hard, the MILP cannot find the solution for large-scale problems in a reasonable amount of time. Moreover, the MILP execution time for an instance is variable due to budget and tour duration amounts. If these are set to be very large or very small, the execution time would be considerably lower, compared to the case where the budget and time are constrained at a medium range. In the first case, most cities can be on the route, whereas in the second case, only a very few cities can be on the route. Although, for some large budgets, the computational time increases due to increasing the size of the feasibility region and hence complexity of the problem by adding more vehicles. This increase is still less than in cases with a medium-size budget.

Hence, two genetic algorithms—GA-2OPT and GA-IR—were designed. GA-2OPT always provides a closer solution to optima after a reasonable number of generations. Even for cases where the feasible region is tight and fixed cost is playing a critical role in finding the best solution, GA-2OPT provides a better solution. Because diversity of the initial population is supported by using a sweep method for both GAs, using the greedy 2-OPT algorithm in GA-2OPT helps to improve the solutions in terms of travel cost, and more nodes can be inserted on a given route, while in GA-IR, two random nodes are swapped on a given route, but this does not necessarily improve the solution. Therefore, GA-2OPT performs better than GA-IR.

In addition, genetic algorithm parameters play an important role in GA performance. By setting these parameters correctly, the convergence of algorithms to a good solution is enhanced considerably.

This research contributes to the literature by introducing a new class of selective multiple traveling salesman problems. Furthermore, the genetic algorithms designed to solve this problem are combining structured solutions and other heuristics to provide a good solution in a short computational time. This research can be extended to the case where there are multiple depots, and each depot has multiple travelers with a fixed cost considered for each vehicle. Moreover, for each node, a time window can be considered. Then each node must be visited within its time frame.

5.6 References

[1] Bektas, T. (2006). The multiple traveling salesman problem: An overview of formulations and solution procedures. *Omega*, *34*(3), 209-219.

[2] Arulselvan, A. (2009). Network model for disaster management (Doctoral dissertation). University of Florida, Gainesville.

[3] Finke, G., Claus, A., & Gunn, E. (1984). A two-commodity network flow approach to the traveling salesman problem. *Congressus Numerantium*, *41*(1), 167-178.

[4] Miliotis, P. (1978). Using cutting planes to solve the symmetric travelling salesman problem. *Mathematical Programming*, *15*(1), 177-188.

[5] Bhide, S., John, N., & Kabuka, M. R. (1993). A Boolean neural network approach for the traveling salesman problem. *IEEE Transactions on Computers*, *42*(10), 1271-1278.

[6] Glover, F. (1990). Artificial intelligence, heuristic frameworks and tabu search. *Managerial and Decision Economics*, *11*(5), 365-375.

[7] Ali, A. I., & Kennington, J. L. (1986). The asymmetric M-travelling salesmen problem: A duality based branch-and-bound algorithm. *Discrete Applied Mathematics*, *13*(2-3), 259-276.

[8] Gromicho, J., Paixão, J., & Bronco, I. (1992). Exact solution of multiple traveling salesman problems. In *Combinatorial optimization* (pp. 291-292). Berlin Heidelberg: Springer.

[9] Laporte, G., & Nobert, Y. (1980). A cutting planes algorithm for the m-salesmen problem. *Journal of the Operational Research Society*, *31*(11), 1017-1023.

[10] Gavish, B., & Srikanth, K. (1986). An optimal solution method for large-scale multiple traveling salesmen problems. *Operations Research*, *34*(5), 698-717.

[11] Feng, H. K., Bao, J. S., & Ye, J. (2009). Particle swarm optimization combined with ant colony optimization for the multiple traveling salesman problem. In Materials Science Forum (Vol. 626, pp. 717-722). Trans Tech Publications.

[12] Nallusamy, R., Duraiswamy, K., Dhanalaksmi, R., & Parthiban, P. (2009). Optimization of Non-Linear Multiple Traveling Salesman Problem Using K-Means Clustering, Shrink Wrap Algorithm and Metaheuristics. *International Journal of Nonlinear Science*, *8*(4), 480-487.

[13] Zhang, T., Gruver, W. A., & Smith, M. H. (1999). Team scheduling by genetic search. In *Intelligent Processing and Manufacturing of Materials, 1999. IPMM'99. Proceedings of the Second International Conference on* (Vol. 2, pp. 839-844). IEEE.

[14] Tang, L., Liu, J., Rong, A., & Yang, Z. (2000). A multiple traveling salesman problem model for hot rolling scheduling in Shanghai Baoshan Iron & Steel Complex. *European Journal of Operational Research*, *124*(2), 267-282.

[15] Carter, A. E., & Ragsdale, C. T. (2006). A new approach to solving the multiple traveling salesperson problem using genetic algorithms. *European Journal of Operational Research*, *175*(1), 246-257.

[16] Gillett, B. E., & Miller, L. R. (1974). A heuristic algorithm for the vehicle-dispatch problem. *Operations Research*, *22*(2), 340-349.

[17] Király, A., & Abonyi, J. (2011). Optimization of multiple traveling salesmen problem by a novel representation based genetic algorithm. In *Intelligent Computational Optimization in Engineering* (pp. 241-269). Berlin Heidelberg: Springer.

[18] Byung-In, K., Jae-Ik, S., & Min, Z. (1998). Comparison of TSP Algorithms–Project for Models in Facilities Planning and Materials Handling.

[19] Potvin, J. Y., Lapalme, G., & Rousseau, J. M. (1989). A generalized k-opt exchange procedure for the MTSP. *INFOR: Information Systems and Operational Research*, *27*(4), 474-481.

[20] Malmborg, C. J. (1996). A genetic algorithm for service level based vehicle scheduling. *European Journal of Operational Research*, *93*(1), 121-134.

[21] Park, Y. B. (2001). A hybrid genetic algorithm for the vehicle scheduling problem with due times and time deadlines. *International Journal of Production Economics*, *73*(2), 175-188.

[22] Király, A., & Abonyi, J. (2010). A novel approach to solve multiple traveling salesmen problem by genetic algorithm. In *Computational Intelligence in Engineering* (pp. 141-151). Berlin Heidelberg: Springer.

[23] Odersky, M& al. (2004). *An overview of the Scala programming language*. EPFL Lausanne, Switzerland

[24] Brooke, A., Kendrick, D., Meeraus, A., Raman, R., & America, U. (1998). The general algebraic modeling system. *GAMS Development Corporation*.

[25] CPLEX, I. I. (2009). V12. 1: User's Manual for CPLEX. *International Business Machines Corporation*, *46*(53), 157.

CHAPTER 6

NEW MULTIPLE DEPOT VEHICLE ROUTING PROBLEM WITH BUDGET AND TIME CONSTRAINTS

Abstract

This chapter introduces a new variant of the multiple depot vehicle routing problem, called the multiple depot capacitated vehicle routing problem with partial coverage. In this problem, there is a set of depots, each with a fleet of vehicles. The goal here is to maximize the total profit (served demand), subject to limited budget, travel time, and capacity by making a set of subtours for each vehicle departing from a depot and returning to the same depot. Furthermore, each node can serve other nodes that are not indirectly on the route. Two new genetic algorithms, GA-MDAS and GA-MDSW, are designed to solve the problem in a reasonable amount of time. In GA-MDAS, routes are constructed based on the assignment method, while in GA-MDSW, routes are constructed based on the sweep method. Both GAs are coded in Scala. Moreover, to measure the efficiency of the GA approaches, the problem is solved as an MILP. For most instances, GA-MDSW provides better solutions than those of GA-MDAS. Results are presented and analyzed.

6.1 Introduction

The multiple depot vehicle routing problem is one variation of the vehicle routing problem that is more suitable for addressing practical problems [1]. The MDVRP is an extension of the VRP in which there are multiple depots where vehicles depart, and they can either return to the same depot from which they start their tour (fixed-destination MDVRP), or they can return to other depots (non-fixed-destination MDVRP). The MDVRP can be extended to different problems based on the objective and constraints. One variant of the MDVRP is a selective MDVRP with profits, which can be applied to a variety of real-world problems, such as humanitarian logistics

and emergency response to devastating disasters [2, 3]. Another application can be delivery of home heating fuel, whereby the urgency of a customer request is considered as a score [4].

This newly introduced multiple depot capacitated vehicle routing problem with partial coverage is defined on an undirected graph $G = (R \cup V^K \cup W^k \cup Z, E)$, where $V^K \cup W^k \cup Z$ is the vertex set, $E = \{(v_i^k, v_j^k): v_i^k, v_j^k \in R \cup V^K \cup W^k \cup Z^k, i < j\}$ is the edge set, and R is the set of depots. In each depot, there is a fleet of vehicles denoted by G_r such that $\bigcup_R G_r = K$. All vehicles must return to the same depot from which they originate, i.e., MDCVRPWPC is a fixed-destination problem. V^K is the set of nodes served directly by vehicle k, W^k is the set of nodes served indirectly by the nodes in set V^K, and set Z is the set of nodes not served. A cost c_{ij}^k and a travel time t_{ij}^k are associated with each edge (v_i^k, v_j^k) for vehicle k. Each vertex v_i has a demand d_i and a capacity cap_i. In this problem, there is a limited budget and a limited time for each vehicle to serve the nodes on its route. A prize is associated with each vertex, which is the total demand of the nodes served directly and indirectly by vehicle k. The aim of MDCVRPWPC is to determine a set of subtours of with maximum directly and indirectly served demand (i.e., profit) from different depots subject to budget and time constraints.

MDCVRPWPC is different from the CVRPWPC presented in Chapter 3, in that there are multiple depots, while in CVRPWPC, there is a single depot. Each depot has a fleet of vehicles to serve the other nodes, and each vehicle starting a tour from a depot must return to the same depot.

The MDVRP is NP-hard [5], and there are several heuristics and metaheuristics in the literature to solve it. Metaheuristics such as simulated annealing algorithms [6, 7] and the variable neighborhood search [8, 9] can be used to solve the capacitated MDVRP. Construction-based algorithms use an improvement after constructing a solution [10, 11]. Tabu search algorithms [12, 13] and genetic algorithms are the other metaheuristics used to solve the MDVRP [14, 15].

Also, there are some hybrid algorithms to solve the MDVRP. Vidal, Crainic, Gendreau, and Prins [16] propose a hybrid GA with iterated local search and dynamic programming to solve the MDVRP with an unconstrained vehicle fleet. Ting and Chen [17] propose a hybrid algorithm that combines multiple ant colony systems and simulated annealing. Wang, Zhang, and Wang [18] combine simulated annealing with a modified variable neighborhood search algorithm, and a clustering algorithm is used to allocate customers in the initial solution construction phase.

Some GA-based algorithms are used to solve a non-fixed-destination MDVRP [19, 20, 21, 23]. A few GAs in the literature solve fixed-destination MDVRPs. Thangiah and Salhi [23] introduce GenClust genetic algorithms in which an adaptive method based on geometric shapes is developed, a clustering approach is used to assign customers to each depot, and then a solution is found for each cluster. Ombuki-Berman and Hanshar [24] introduce a genetic algorithm using a dynamic inter-depot mutation operator, which may reassign the borderline of clusters to solve the MDVRP.

In this chapter, two new genetic algorithms are designed to solve large-scale problems. The first uses a method inspired from the sweep method [25] to construct initial solutions, while the second uses a technique inspired from the assignment method [26] to construct routes.

This chapter contributes to the literature review in several ways. First, a new multiple depot capacitated vehicle routing problem is proposed, whereby some customers might not get served due to limited resources. Moreover, since partial demand of some nodes can be satisfied by nodes on the route, the prize at each node depends on nodes that are visited. Second, new genetic algorithms based on clustering, sweep, and assignment methods are designed to solve the problem. Furthermore, the mutation operator uses a greedy 2-OPT mutation and also incorporates inter-depot mutation, which may change cities visited on the borderlines of clusters. The problem is

solved as MILP for medium-size problems, in order to have a benchmark from which to evaluate the performance of the proposed metaheuristics.

The remainder of this chapter is organized as follows. In section 6.2, a mathematical model is presented to define the problem. In section 6.3, the proposed GAs are explained. Computational results and analysis are found in section 6.4, followed by conclusions in section 6.5.

6.2 Mathematical Model

MDCVRPWPC is a selective vehicle routing problem [27]. In this problem, a set of cities with their demands and multiple depots exist. At each depot, there is a set of vehicles with a certain capacity. This problem is a fixed-destination problem, whereby each vehicle starting from a depot should terminate its trip at the same depot. Due to limited budget and time, all nodes might not be visited. A node's demand can be partially satisfied if it lies within a distance of other nodes on a route, i.e., of nodes served directly. The objective is to find the best subtour with the maximum total profit. Each vehicle can serve each node up to that node's capacity, and each node can serve a portion of other nodes' demand, if their distance is less than a predefined distance limit. Because each node can serve the other nodes, profit that can be gained from visiting a node depends on the nodes visited previously, because other nodes on the route could also serve some of the nodes that are not on the route.

The MDCVRPWPC is formulated as a mathematical model as follows:

Sets

$I, J: sets\ of\ nodes$

$K_r: set\ of\ velicles\ at\ depot\ r$

$K: set\ of\ velicles$

$$\bigcup_{r \in R} K_r = K$$

$R: set\ of\ depots$

Parameters

$i,j \in I$: nodes indices

$k \in K$: vehicle index

$r \in R$: depot index

$tp_i^k: time\ of\ getting\ service\ at\ node\ i\ by\ vehicle\ k$

$T_{tour}^k: maximum\ time\ that\ vehicle\ k\ can\ be\ on\ route$

$t_{ij}^k: time\ of\ traveling\ between\ node\ i\ and\ node\ j\ by\ vehicle\ k$

$D_i: demand\ at\ node\ i$

$c_{ij}^k: cost\ of\ traveling\ from\ node\ i\ to\ node\ j\ by\ traveler\ k$

$L_{ij}: distance\ between\ node\ i\ and\ node\ j$

$DIS_j: maximum\ distance\ which\ people\ will\ travel\ to\ come\ to\ city\ j$

$CAP_i: capacity\ of\ node\ i$

$E_j: number\ of\ people\ who\ travel\ to\ city\ j$

$capv_k: capacity\ of\ vehicle\ k$

$B_k: budget\ assigned\ to\ vehicle\ k$

Variables

$$z_i^k = \begin{cases} 1 & if\ node\ i\ gets\ served\ by\ vehicle\ k \\ 0 & if\ node\ i\ does\ not\ get\ served\ by\ vehicle\ k \end{cases}$$

$$w_i^k = \begin{cases} 1 & if\ node\ i\ gets\ served\ by\ vehicle\ k\ indirectly \\ 0 & if\ node\ i\ does\ not\ get\ served\ by\ vehicle\ k\ indirectly \end{cases}$$

$$v_i = \begin{cases} 1 & if\ node\ i\ \ is\ not\ on\ the\ route\ nor\ gets\ served\ \ indirectly \\ 0 & O.W. \end{cases}$$

$$x_{ij}^k = \begin{cases} 1 & if\ node\ i\ proceed\ to\ node\ j\ by\ vehicle\ k\ immediately \\ 0 & O.W. \end{cases}$$

$$y_{ij}^k = \begin{cases} 1 & if\ node\ j\ is\ assigned\ to\ node\ i\ \ to\ be\ served\ in\ directly \\ 0 & O.W. \end{cases}$$

u_{ik}: $auxilary\ variable$

Objective

$$\text{Max} \sum_{k\in K} \sum_{i\in I} z_i^k d_i + \sum_{k\in K} \sum_{i\in I} \sum_{j\in I} y_{ij}^k e_j \quad (6.1)$$

Constraints

$$d_i \sum_{k\in K} z_i^k + \sum_{k\in K} \sum_{j\in I} y_{ij}^k e_j \le CAP_i \sum_{k\in K} z_i^k \ \ i \in I \quad (6.2)$$

$$\sum_{j\in I, j\neq i} x_{ij}^k = z_i^k \ \ i \in I\ , \forall k \in K \quad (6.3)$$

$$\sum_{i\in I, i\neq j} x_{ij}^k = z_j^k \ \ j \in I\ , \forall k \in K \quad (6.4)$$

$$\sum_{j\in I} x_{rj}^k = 1 \quad r \in R, k \in K_r \quad (6.5)$$

$$\sum_{j\in I} x_{jr}^k = 1 \ \ r \in R, k \in K_r \quad (6.6)$$

$$\sum_{k\in K} z_i^k \ \ + \sum_{k\in K} w_i^k + V_i = 1 \ \ \forall\, i \in I \quad (6.7)$$

$$\sum_{i\in I} y_{ij}^k = w_j^k \quad \forall j \in I, \forall k \in K \quad (6.8)$$

$$\sum_{j\in I} \sum_{i\in I} t_{ij}^k x_{ij}^k + \sum_{i\in I} tp_i^k z_i^k \le T_{tour}^k \quad \forall k \in K \quad (6.9)$$

$$\sum_{j\in I} \sum_{i\in I} c_{ij}^k x_{ij}^k \le B_k \ \ \forall k \in K \quad (6.10)$$

$$\sum_{i\in I} d_i z_i^k + \sum_{i\in I} \sum_{j\in I} y_{ij}^k e_j \le capv_k \ \ \forall k \in K \quad (6.11)$$

$$z_i^k \ge y_{ij}^k \quad \forall i, j \in I, \forall k \in K \quad (6.12)$$

$$L_{ij} y_{ij}^k \le DIS_j \quad \forall\, i, j \in I \quad (6.13)$$

$$u_{ik} - u_{ik} + n x_{ij}^k \le n - 1 \ \ \forall k \in K, \forall\, i \in I \quad (6.14)$$

Equation (6.1), the objective function, maximizes the served demand. Constraint (6.2) ensures that the capacity at a node can serve the demand at that node and all other indirectly served demands from other nodes that are not on a route.. Constraints (6.3) and (6.4) imply that from each

node, a vehicle can travel only to another node, and a vehicle can arrive to a node only from one node. Constraints (6.5) and (6.6) ensure that each vehicle departs from its corresponding depot and returns back to the same depot only once. Constraint (6.7) ensures that each node can have only one of the three possible states: the node is on a route (i.e., served directly), the node is not on a route but is being served indirectly by one of the nodes on the route, or the node is not on a route and is not served indirectly. Constraint (6.8) ensures that each node can be served indirectly by only one of the other nodes that is on a route. Constraints (6.9) and (6.10) ensure that the time and cost of traveling cannot exceed the budget for each vehicle. Constraint (6.11) implies that each vehicle capacity should be greater than the demand of the nodes serving directly and indirectly. Constraint (6.12) implies that if node *j* is receiving service from node *i*, then node *i* must be on the route. Constraint (6.13) limits the distance between the serving node and the served node. Constraint (6.14) is a Miller-Tucker-Zemlin subtour elimination constraint [28] for each route.

This mathematical program includes $3\,[I][K] + [I]^2[K] + [I]^2 + [I]$ variables and $4\,[I][K] + [I]^2[K] + [I]^2 + 2[I] + 2[R][K_r] + 3[K]$ constraints. There are $[I][K]$ integer variables, and the remaining variables are binary. The number of variables and constraints are determined by summing the cardinality of all variables and constraints.

Proposition 1. The complexity of the MDCVRPWPC problem is $\binom{n}{x_1,\ldots,x_m}\frac{(x_1-1)!}{2} \times \ldots \times \frac{(x_m-1)!}{2} \times 2^{n-(x_1+\cdots+x_m)}$, such that $x_1 + \cdots + x_m \leq n$, where *m* is the number of vehicles, and *n* is the number of nodes.

Proof:

The number of visited nodes by a typical vehicle denoted by *m* can be a variable denoted by x_m. The number of different ways that *n* nodes can be assigned to *m* vehicles is $\binom{n}{x_1,\ldots,x_m}$.

Each vehicle can have $\frac{(x_m-1)!}{2}$ Hamiltonian cycles for an asymmetric distance matrix. For the remaining nodes that are not served directly (i.e., $n-(x_1+\cdots+x_m)$), there are two possibilities: they can be either assigned to one of the nodes on the route or not. Then, there are $2^{n-(x_1+\cdots+x_y)}$ cases of assigning nodes. In total, there are $\binom{n}{x_1,\ldots,x_m}\frac{(x_1-1)!}{2}\times\ldots\times\frac{(x_m-1)!}{2}\times 2^{n-(x_1+\cdots+x_m)}$ ways to be evaluated.

Proposition 2. MDCVRPWPC is NP-hard.

Proof:

The TSP is NP-hard [29]. In an asymmetric TSP, there are $\frac{(n-1)!}{2}$ cycles to be evaluated to find the best Hamiltonian cycle. The complexity of the CVRPWPC is $\binom{n}{x_1,\ldots,x_m}\frac{(x_1-1)!}{2}\times\ldots\times\frac{(x_m-1)!}{2}\times 2^{n-(x_1+\cdots+x_m)}$, which is greater than $\frac{(n-1)!}{2}$ cycles. Therefore, CVRPWPC is NP-hard.

Because the problem is NP-hard and difficult to solve largescale problems, for larger instances of this problem, one may need to use heuristic methods to find good quality solutions in a reasonable amount of time. Thus, we propose to solve MDCVRPWPC using a GA-based metaheuristic.

6.3 Genetic Algorithm to Solve MDCVRPWPC

In this algorithm, there is a fleet of vehicles in each depot, and they are required to return to their origin. As nodes are assigned to each depot, first based on a clustering algorithm, we use the same logic we use for CVRPWPC for each depot to construct structured solutions: either the sweep method or the assignment method are used to construct the routes for each depot. The final solution is a combination of all structured solutions. Then a two-point crossover is performed to construct new solutions. To perform the crossover, chromosomes are transformed based on the

vehicle serving each node. After crossover, the fittest chromosomes are selected and then inter-depot and intra-depot mutations are implemented to preserve the diversity of solutions.

The GA-MDCVRPWPC implementation is summarized in Table 6.1, and details of the implementation are presented in sections that follow.

Table 6.1. Genetic Algorithm for MDCVRPWPC

Steps	Pseudo Code of Proposed Genetic Algorithm
1	Set number of generations *N* and population size
2	Set selection probability p_s, crossover probability p_c, mutation probability p_m
3	Generate initial population
4	While (the number of iterations is less than *N*)
5	Evaluate each chromosome according to its fitness function
	5.1 Solve assignment problem to find the fitness for each path
6	Select chromosomes according to their fitness using roulette wheel selection until pool is full
7	Execute crossover operation
	7.1 Check feasibility of each chromosome
	7.2 Modify unfeasible chromosome to make it feasible
8	Execute mutation
9	End

6.3.1 Initialization

The MDVRP solution is represented by *n* vectors, where *n* is the number of depots, and each vector consists of corresponding routes originating and terminating to a depot. For each depot, there is a vector representing a cluster of paths starting and returning to the depots by a fleet of

vehicles. Therefore, if there are *n* depots, *n* vectors will represent a solution. For instance, assuming that there are 25 nodes and two depots and there is a fleet of two vehicles in first depot and a fleet of three vehicles in second depot, a solution can be found as shown in Figure 6.1 Based on this solution, in the first depot, vehicle 1 is visiting nodes 1-4-6-8-7 and vehicle 2 is visiting nodes 12-15-18-2-3-10. In second depot, vehicle 1 is visiting nodes 11-5-9-13-16-17, vehicle 2 is visiting nodes 17-22-14-24-19-21 and vehicle 3 is visiting nodes 19-21.

Depot	1	Vehicle	1	1	1	1	1	2	2	2	2	2	2
		Vector 1	1	4	6	8	7	12	15	18	2	3	10
	2	Vehicle	1	1	1	1	1	1	2	2	2	3	3
		Vector 2	11	5	9	13	16	17	22	14	24	19	21

Figure 6.1. Chromosome representation

To generate routes, a clustering approach is implemented. In this approach, for each depot, nodes are assigned to each cluster based on their geographical position and demand. The size of each cluster is proportional to the capacity of each depot. Two methods are used in each cluster to determine the corresponding routes: the sweep method [25] and a method inspired from the assignment algorithm [26].

To perform the sweep method, the polar angle of each node is determined with respect to the corresponding depot. Since the vehicles of each depot must return to the same depot, a random node out of each cluster is generated and designated to a vehicle belonging to the corresponding depot. The order of visiting nodes for each vehicle is based on ascending order of their polar angles.

In the assignment method, some seeds would be assigned to vehicles first. These seed points are dummy points and are defined in order to calculate the assignment cost of nodes to each vehicle. Each seed is located along the ray that bisects the angle of the corresponding vehicle cone. The distance of each seed from the depot is fixed and is equal to the maximum distance of cities

belonging to the vehicle cone. This method can be applied to the planar case where whole cities are on a plane, and distances between the cities can be measured by using the Euclidean distance. The plane of each cluster should be partitioned into *k* cones, where *k* is the number of vehicles in that depot. Each cone is constructed by merging the cities' cones. Each city cone is defined by drawing rays from the depot that bisect the angle between adjacent cities and the corresponding city. The cone of each vehicle is formed by merging the adjacent cones of cities until the cumulative demand of those cities is less than the vehicle capacity.

Then cities are assigned to vehicles based on the following threshold:

$$CT_{si} = cap_i/[c_{0i} + c_{si} - c_{0s}] \tag{6.15}$$

Assuming a symmetric travel cost metric c_{si} with 0 representing the depot, the assignment cost of node *i* to seed *s* equals $c_{0i} + c_{si} - c_{0s}$ [30], and cap_i is the capacity of city *i*. In both methods, the insertion will be done until the feasibility of the solution is violated, which means that if any of the following conditions occur, then the insertion will be stopped:

- Total cost of traveling between nodes on the route of each vehicle is greater than the assigned budget to that vehicle.
- Total capacity of nodes on the route of each vehicle is greater than the capacity of each vehicle.
- Total time of traveling between nodes on the route of each vehicle is greater than the maximum time that the vehicle can be on the route.

The assignment will be performed until any budget, capacity, or time constraints are violated. The order of nodes is based on their polar angles.

6.3.2 Fitness Evaluation and Selection

In the genetic algorithm, each chromosome is evaluated based on a fitness function to select the chromosomes for crossover and mutation. The fitness is a function of the problem objective which is the maximum number of customers served:

$$f = \exp(obj) \tag{6.16}$$

where f denotes the fitness of chromosome, and obj represents the maximum number of served customers on the tour.

The maximum number of served nodes is determined by assigning nodes that are not on the route to nodes that are on the route. This is an optimization problem where a set of nodes should be assigned to another set of nodes with a certain serving capacity. Furthermore, the distance of nodes assigned to a node on a route cannot be more than a predetermined covering distance.

Since the cities on a route are determined, the MDCVRPWPC mathematical program reduces to the following assignment mathematical program:

Parameters

$$\bar{Z}_\iota = \begin{cases} 1 & if\ node\ i\ is\ on\ any\ given\ route \\ 0 & o.w. \end{cases}$$

Variables

$$h_i = \begin{cases} 1 & if\ node\ i\ is\ on\ any\ route\ but\ served\ indirectly \\ 0 & o.w. \end{cases}$$

$$q_{ij} = \begin{cases} 1 & if\ node\ j\ is\ assigned\ to\ node\ i \\ 0 & o.w. \end{cases}$$

Model

$$\max \sum_{i\in I} D_i \bar{Z}_\iota + \sum_{j\in I} \sum_{i\in I} \mathrm{E}_j q_{ij} \tag{6.17}$$

Subject to

$$\bar{Z}_\iota + h_i + V_i = 1 \qquad \forall\ i \in I \tag{6.18}$$

$$\sum_{i=1}^{n} q_{ij} = W_j \qquad \forall j \in I \qquad (6.19)$$

$$\overline{Z}_\iota \geq q_{ij} \qquad \forall i, j \in I \qquad (6.20)$$

$$\sum_{j \in I} E_j q_{ij} + D_i \overline{Z}_\iota \leq CAP_i \overline{Z}_\iota \qquad \forall i \in I \qquad (6.21)$$

$$L_{ij} q_{ij} \leq DIS_j \qquad \forall\, i, j \in I \qquad (6.22)$$

q_{ij}, h_i: $binary\ variables$

Given a chromosome, we already know which nodes are on the route, i.e., $\overline{Z}_\iota$ is known. Constraint (6.18) ensures that each node takes only one of three possible states: the node is on the route, the node is not on the route but is being served partially by one of the nodes on the route, or the node is neither on the route nor served indirectly. Constraint (6.19) ensures that each node can be served indirectly by only one of the other nodes on the route. Constraint (6.20) implies that if node *i* is served by node *j*, then node *j* must be on the route (otherwise it cannot serve node *i*). Constraint (6.21) limits the number of customers served directly and indirectly at each node by the capacity of that node. Constraint (6.22) limits the distance between the serving node and the served node. As shown, to find the fitness of each chromosome, an optimization problem is solved to find the maximum number of served customers based on the given path. In this problem, there are $2\,[I] + [I]^2$ variables and $3\,[I] + 2[I]^2$ constraints.

After computing the fitness, a roulette wheel selection [31] is implemented to select the parents to crossover.

6.3.3 Crossover

A two-point crossover is implemented on one of the depots at each generation. To implement the crossover at each cluster, as performed for the CVRPWPC, solutions are recoded based on the vehicle number serving the nodes. Labeling the vehicles is based on the closeness of the average polar angles of the nodes they are visiting: the first parent vehicles are labeled based

on the number of vehicles, and the second parent vehicles are labeled based on the closeness of the average polar angle of the nodes they are visiting to the average polar angle of the nodes that the first parent vehicles are visiting. Therefore, vehicles with same the label will serve nodes in the same area. Because the structured solutions are constructed for each cluster individually and merged, this type of labeling would help to reorder nodes in the neighborhood cones.

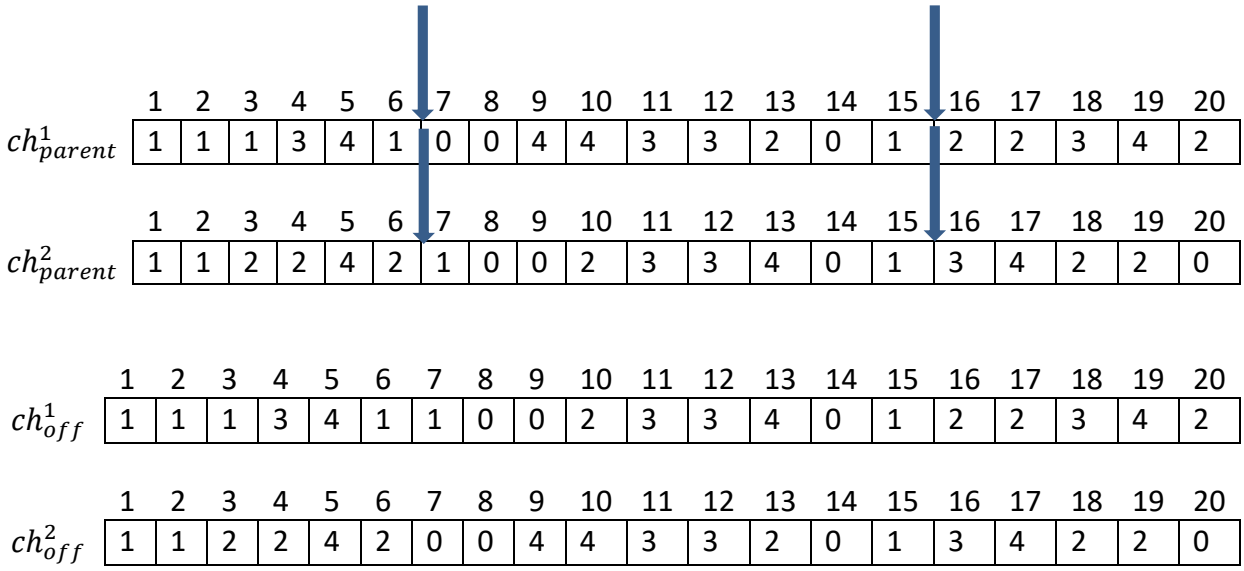


Figure 6.2. Crossover procedure example

After creating new chromosomes, there might be some infeasibility in terms of violating the time limit, capacity, and budget constraints. In this case, some cities are dropped randomly from the current route to achieve feasibility. Because each node can receive service from a vehicle only once or it will not receive service at all in this new chromosome representation, no city will be repeated on the same route.

6.3.4 Mutation

To preserve diversity, an intra-depot mutation and an inter-depot mutation [24] are performed. In the intra-depot mutation, a random depot is selected, and a greedy 2-OPT mutation is performed on the selected chromosomes of that depot. In the greedy 2-OPT mutation, a greedy

2-OPT optimization is performed on each route to save budget in terms of time and cost of traveling. Thereafter, within the released budget, an insertion mutation is implemented to insert nodes out of the not-served nodes based on their polar angles. Furthermore, due to imposing the initial clustering and assigning nodes to the depot, an inter-depot mutation is implemented to swap some nodes between clusters and improve the quality of the solution consequently.

6.3.4.1 Intra-Depot Mutation

Applying an intra-depot mutation will increase the diversity of solutions in each cluster. Since the objective is maximizing the collected profit, a greedy 2-OPT mutation is implemented to improve the solution. Before implementing insertion mutation, the greedy 2-OPT optimization [32] is implemented, aiming to improve the solution in terms of saving cost and time of traveling. Then performing the insertion mutation provides a higher chance of having a feasible solution. The greedy 2-OPT mutation is summarized in Table 6.2.

Table 6.2. 2-OPT Greedy Algorithm

Step	Pseudo Codes of Greedy 2-OPT Mutation Algorithm
1	Perform 2-OPT greedy algorithm for each route of selected chromosome.
	1.1 Exchange node i and j, $i < j$. Compare resulting z with current z^*. If $z > z^*$, and j<n then repeat 1.1; otherwise, go to step 1.2.
	1.2 If $z < z^*$, $j = j + 1$ and go to step 1.1. If $z > z^*$ and $j = n$, set $i = i + 1$, $j = j + 1$ and go to step 1.1; otherwise, the current solution is the best solution and terminate the procedure.
2	Insert node in the optimized solution.
	2.1 Calculate polar angle of nodes not served and average polar angle of nodes on each route.
	2.2 For each route, add node that has closest polar angle to average polar angle of nodes on route.
	2.3 While the solution is feasible, go to step 2.1.
3	End

6.3.4.2 Inter-Depot Mutation

Since the initial clustering increases the chance of sticking at the local optima, and in order to diversify solutions, inter-depot mutation is performed at every ten generations. In inter-depot mutation, a customer from a depot will be swapped to another depot, if it is feasible.

6.4 Computational Results and Analysis

We use the clustering algorithm to assign nodes to the depots based on their geographical distribution and considering the capacity of vehicles assigned to each depot and node demand. Thereafter, in each depot, two different algorithms (GA-MDSW and GA-MDAS) are used to construct the structured solutions. In GA-MDSW, the routes are constructed based on the sweep method, while in GA-MDAS, routes are constructed based on the assignment method. Crossover and inter-depot mutation are randomly done for one depot in each generation. To preserve diversity, every ten generations, intra-depot mutation is performed. This procedure was explained in section 6.3.

6.4.1 Experimental Design

Random instances of a problem are generated using Geomidpoint.com. In this software, by giving the latitude and longitude of the depot, and the number of nodes in the problem, cities are randomly generated around the depot within a specified distance. The distance between cities is calculated using the Euclidean distance measurement. .

The demand of each node is distributed uniformly. Demand and capacity of each node are equal and constant. If a node is on a route, then all of its demand will be served. If a node is served by another node indirectly, then half of its demand will be served.

We want to evaluate the efficiency of proposed algorithms for randomly dispersed instances as well as instances generated by a clustering algorithm. In the first case, since the

assignment of nodes to depots is based on a clustering algorithm, we also examine the effect of implementing different algorithms on the final solution of GAs.

The proposed GAs (i.e., GA-MDSW and GA-MDAS) are coded in Scala programming language [33]. Moreover, each instance is solved as a mixed-integer linear program in the GAMS [34] using the CPLEX [35] solver to provide a benchmark with which to compare the results of both GAs to optima, and each instance is run on a computer with 8GB RAM, Core™ i7 CPU, and 465 GB hard disk space.

Ten randomly dispersed instances with different sizes are generated. A predetermined demand of 10 and capacity of 15 are assigned to each node. The budget amount is set such that the computational time of solving the problem with CPLEX is large. In some cases, the upper limit of the CPLEX solution is captured where CPLEX fails to solve the problem. Moreover, each instance is solved considering two different numbers of vehicles: two and three.

We also construct three different distributed instances where two of them have two depots, and one of them has three depots. The nodes of each depot are well fitted to a separate cluster. GA-MDSW and GA-MDAS are run for each instance.

Each experiment has been solved using two different algorithms, GA-MDSW, GA-MDAS, and the optimal solution is found by solving the MDCVRPWPC MILP. Results for each algorithm are recorded and shown. To evaluate the efficiency of proposed GAs, each solution is compared to the optimal solution (or the upper bound provided by CPLEX, if CPLEX fails to find the optimal solution and is out of memory) by calculating the relative error as follows:

$$RE = \frac{optimal\ profit\ \ or\ upper\ bound - best\ profit}{optimal\ profit\ upper\ bound} \tag{6.24}$$

For each instance, the GAs are run for five times, and the average and best profit are recorded. Furthermore, to illustrate the performance of GAs for different distributions, the best

profit results are depicted in different graphs. The efficiency of GA-MDSW is better comparing to GA-MDAS, although both algorithms provide good solutions. Due to partitioning the nodes into clusters, their distribution might not be dispersed fairly enough around the depot, which results in poor initial solutions for GA-MDAS. In well-clustered instances where nodes are randomly dispersed around the depot, both GAs provide good solutions. However, in the long run, GA-MDSW converges to a solution closer to optima.

6.4.2 Fine-Tuning Genetic Algorithms

Several GA parameters should be fine-tuned:

- Size of the population (*sp*) implies how many chromosomes are in population in any generation.
- Selection probability (p_s) indicates how often crossover is performed.
- Mutation probability (p_m) presents how often parts of chromosome are mutated.
- Elitism probability (p_e) defines the rate of saving the best solutions of previous generation in new generation.

The probabilities are set such that the sum of p_s, p_m, and p_e equal one. The number of computational cycles (N_{max}) is set such that the final solution does not change significantly and CPU time is practical.

A different set of parameters is considered, the GAs are run for a given data set for five times, and the best combination resulting in a highest profit is selected. The set of control parameters are shown in Table 6.3.

Table 6.3. GA Control Parameters Set

Control Parameters	**Values**					
Size of Population (sp)	100,50,30	100,50,30	100,50,30	100,50,30	100,50,30	100,50,30
Selection Probability (p_s)	0.7	0.65	0.6	0.55	0.50	0.50
Mutation Probability (p_m)	0.2	0.25	0.3	0.35	0.4	0.45
Elitism Probability (p_e)	0.1	0.1	0.1	0.1	0.1	0.05

GA parameters are set for different sizes of MDCVRPWPC as shown in Tables 6.4 and 6.5. For larger problems, a lower population size and a greater number of generations provide better results. The crossover and selection probabilities do not vary significantly based on the size of the problem. To preserve the diversity of solutions in GA-MDAS, the mutation rate is slightly greater than the mutation rate in GA-MDSW. Because GA-MDAS constructs the routes cone by cone, the diversity of its solutions is less than the diversity of GA-MDSW solutions. In addition, inter-depot mutation is implemented every ten generations.

Table 6.4. Parameters Used to Fine-Tune GA-MDSW

Scale of TSP	N_{max}	sp	p_s	$p_m^{Inter-dep}/p_m^{intra-dep}$	p_e
$n \leq 50$	30	100	0.6	0.3	0.1
$50 < n \leq 90$	40	100	0.55	0.35	0.1
$90 < n \leq 200$	60	30	0.55	0.40	0.05

Table 6.5. Parameters Used to Fine-Tune GA-MDAS

Scale of TSP	N_{max}	sp	p_s	$p_m^{Inter-dep}/p_m^{intra-dep}$	p_e
$n \leq 50$	30	100	0.55	0.35	0.1
$50 < n \leq 90$	40	100	0.55	0.35	0.1
$90 < n \leq 200$	60	30	0.50	0.45	0.05

6.4.3 Numerical Results

We investigate the following: (a) the effect of budget amount on the optimal solution, (b) the effect of different clustering algorithms on the final results of both GAs, (c) the evaluation of final results of proposed algorithms for well-fitted clusters, and (d) the impact of problem size on GA-MDSW and GA-MDAS performance for different sets of instances.

6.4.3.1 Effect of Budget Amount on Optimal Solution for 32-Node Instance with Two Depots and Fleet of Two Vehicles at Each Depot

For a 32-node instance, we investigate the impact of the budget amount on the optimal solution and CPU time needed by CPLEX to determine this solution. In one of the first our executions, we note that the amount of budget has an effect on the execution time. Our goal is to study the effect of different budget amounts on the execution time for the exact solution. First, we calculate the minimum budget required to visit all nodes as a multiple depot vehicle routing problem, which also gives the maximum amount of profit (i.e., served demand, 724.7). Then, we limit the total budget for the sensitivity analysis of 10% to 90% of this amount in 10% increments. The total profit corresponding to each case is listed in Table 6.6. We also present the optimal routes for the case with 40% budget in Figure 6.3. Table 6.6 shows that the most difficult instance of the problem is when the budget is around 50% of the required budget to visit all nodes. All other instances seem to have shorter CPU execution time, i.e., easier instances to solve.

The effect of budget amount on the final solution is also studied. Increasing the amount of budget results in gaining more profit as a result of larger feasible regions. Furthermore, this results in more directly and indirectly served nodes. After assigning 50% of the minimum budget to visit all nodes directly, the number of unserved nodes drops to zero (Table 6.5). For the amounts more than the minimum tour cost to visit all nodes, the budget increase does not have any more impact on the final result, since the maximum profit is reached at that point. The routes of four vehicles

are depicted in Figure 6.3. Assigning a 40% budget to all vehicles results in visiting ten nodes directly and ten nodes indirectly.

Table 6.6. Profit and Execution Time for Exact Method under Different Budget Amounts (724.7)

Budget	Percent of Minimum Cost	Profit	Number of Directly Served Nodes	Number of Indirectly Served Nodes	Number of Not Served Nodes	CPLEX Time (sec)
72.47	10	0	0	0	32	1.585
144.94	20	0	0	0	32	1.769
217.41	30	75	5	5	22	15.246
289.88	40	150	10	10	17	29,018.562
362.35	50	210	14	14	4	261,864.62
434.82	60	250	18	14	0	240,869.09
507.29	70	280	24	8	0	15,466.026
579.76	80	300	28	4	0	1,527.033
652.23	90	310	30	2	0	1,725.593
724.7	100	320	32	0	0	2,902.536

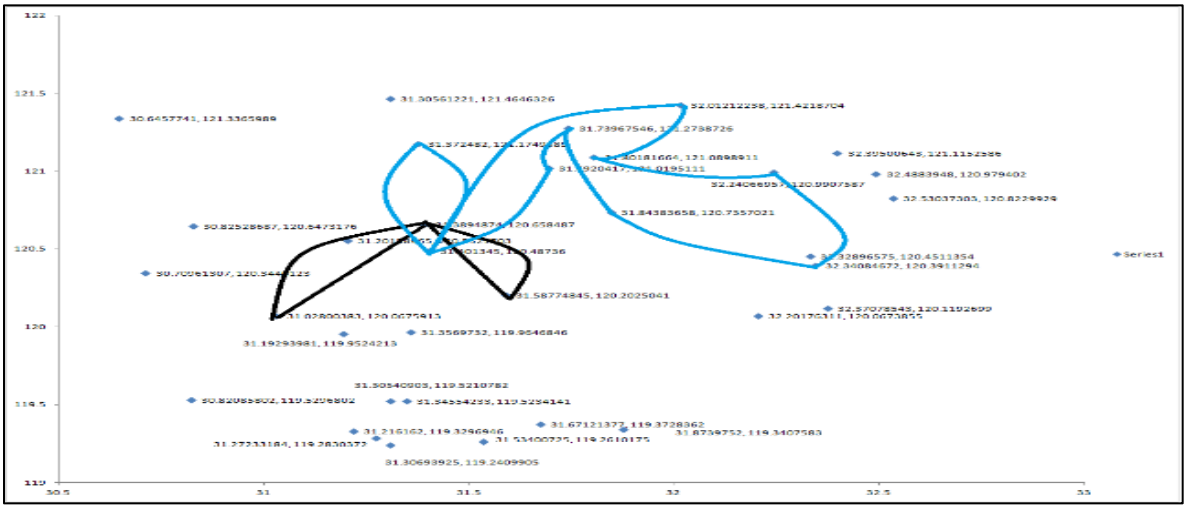
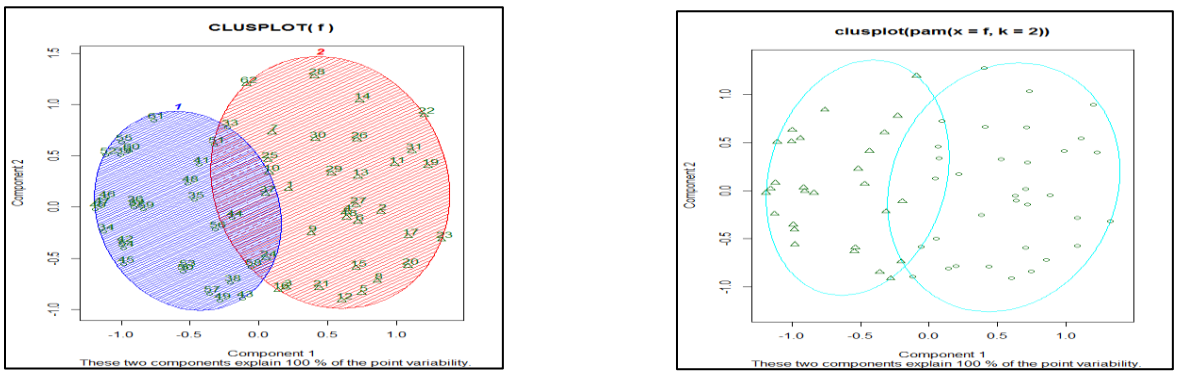


Figure 6.3. Map of routes for instance with 40% minimum budget

6.4.3.2 Do Clustering Methods Have Any Impact on Performance of Proposed Solution Methods?

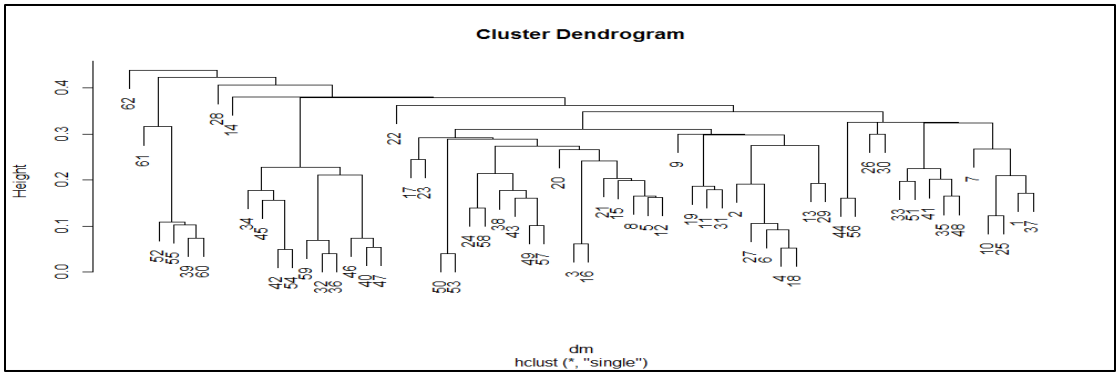
Clustering problems arise in many different applications, such as data mining, pattern recognition, and pattern classification. Cluster analysis is the process of classifying objects into subsets that have meaning in the context of a particular problem. In this chapter, the impact of three different clustering methods—hierarchical clustering, k-means clustering, and K-medoids clustering—on the solution quality of the metaheuristic solution are analyzed.

Because the objective is to assign nodes to each depot based on node location, node demand, and number of vehicles in each depot, these three factors are considered as we use the three different clustering algorithms. In k-means and k-medoids, the number of clusters must be defined. The number of clusters is fixed and set equal to the number of depots (Figure 6.4). Both depots are close to each other and lie within the intersection of clusters. Because demands are distributed evenly among nodes, performing k-means and k-medoids provides equal-size clusters. Since each depot has the same number of vehicles, the results of clustering is satisfying. If the demand and number of vehicles are not even, then a heuristic must be applied after obtaining the k-means and k-medoids results, in order to distribute the nodes among clusters based on their demand and the capacity of vehicles serving each cluster of nodes.



(a) K-means (b) K-medoids

Figure 6.4. Results of different clustering algorithm for RA-60 instance



(c) Hierarchical

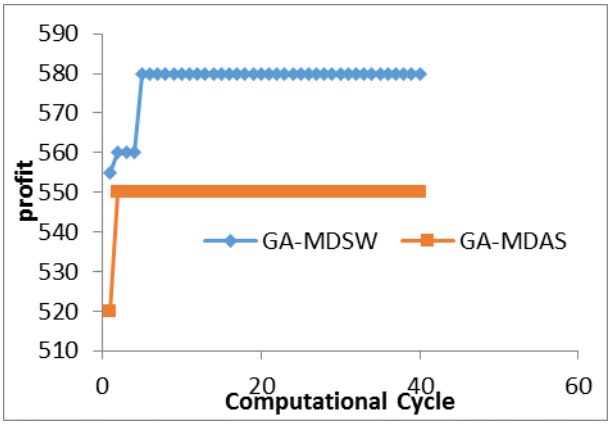
Figure 6.4. (continued)

As shown in Table 6.7, the results of k-means and k-medoids clustering are considerably better than the results of hierarchical clustering. K-medoids results in better solutions for GA-MDSW, while k-means works better for GA-MDAS. However, since the performance of GA-MDAS depends highly on the quality of defining cones, the k-means algorithm seems more promising. In the k-means algorithm the centroid of each cluster is not necessarily one of the nodes, while in the k-medoids algorithm, the centroid of each cluster should be one of the nodes, which can be one reason for better performance of the k-means algorithm. However, the clustering method does not greatly affect the quality of the solution because the clusters are not determined far from the ideal clusters. The CPLEX time to solve this problem is 15,406.63 seconds, while the GA-MDSW takes about 70 seconds to solve the problem.

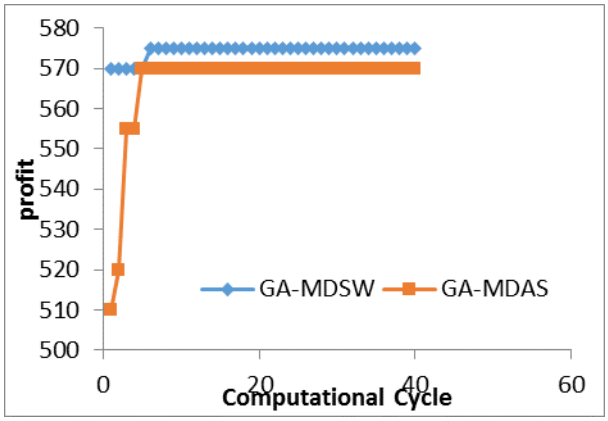
Table 6.7. Impact of Clustering methods on Solution Quality for 60-Node Problem (Optimal Solution = 600)

Clustering Methods	**Average Profit**		**Best Profit**		**RE**	
	GA-MDSW	**GA-MDAS**	**GA-MDSW**	**GA-MDAS**	**GA-MDSW**	**GA-MDAS**
K-medoids	577.875	549.25	580	550	0.03	0.08
K-means	574.375	566.5	575	570	0.04	0.05
Hierarchical	444	417	450	420	0.25	0.3

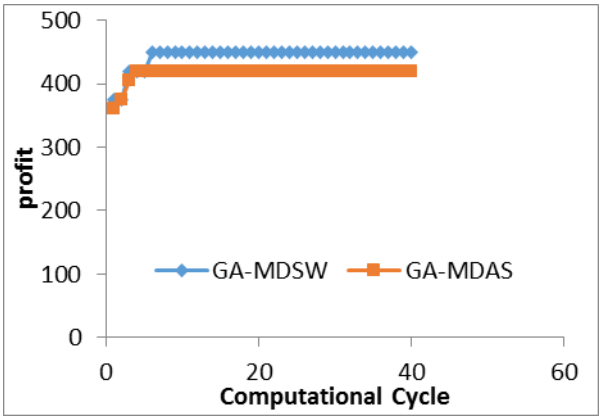
The convergence trend of GAs for different clustering algorithms is shown in Figure 6.5. After splitting the nodes to clusters, distribution of nodes around each depot is not uniform. Consequently, the cones are not well defined for each cluster, and the quality of initial solutions are not good for GA-MDAS. Because GA-MDAS does not have the diversity of GA-MDSW in its initial solutions, the solutions do not improve after a number of computational cycles (Figure 6.5) As a result, for each clustering algorithm, the convergence trend of GA-MDSW to the optimal solution is faster than GA-MDAS.



(a) K-means



(b) K-medoids

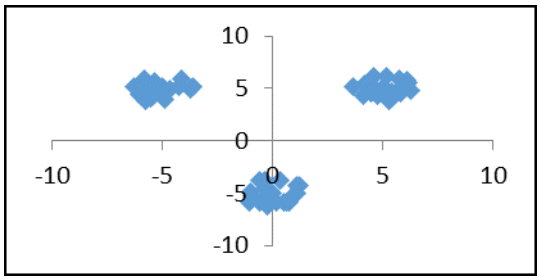


(c) Hierarchical

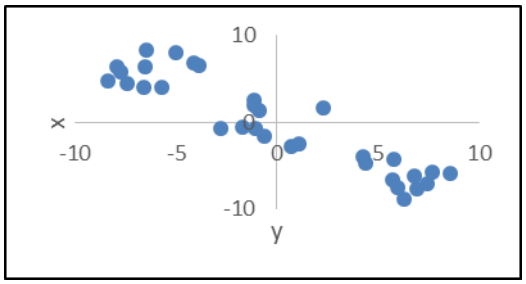
Figure 6.5. Convergence trend of different clustering algorithms

6.4.3.3 GA Evaluation for Different Distributed Instances

To evaluate the performance of the proposed GA methods, three different distributions of nodes are generated, whereby all clusters are well separated and nodes are fairly dispersed around each depot. Each depot is located in the center of each cluster. In one instance, M2D2V, there are two depots, and at each depot there is a set of two vehicles. In the other two instances, there are three depots, each having two vehicles (Figure 6.6). Both GA-MDSW and GA-MDAS approaches are run for five times to check the impact of node distribution variations on the average profit and the best profit (Table 6.8). To calculate the relative error from the optimal solution, the MILP is solved using GAMS/ CPLEX solver.

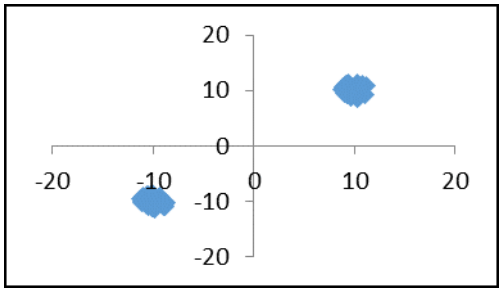


(a) M3D2V-u/d distribution graph



(b) M3D2V1-u/d distribution graph

Figure 6.6. Different distributed instances based on k-means clustering algorithm (continued)



(c) M2D2V-u/d distribution graph

Figure 6.6. Different distributed instances based on k-means clustering algorithm

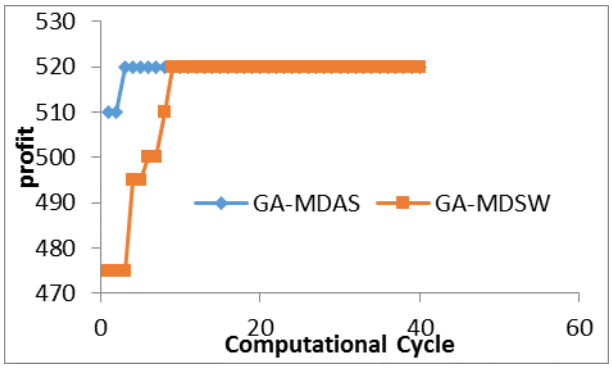
Table 6.8. Results of GA Approaches for Well Separated Clustered Instances

Name	Average Profit		Best Profit		Optimal Profit	RE		CPLEX Time (sec)
	GA-MDSW	GA-MDAS	GA-MDSW	GA-MDAS		GA-MDSW	GA-MDAS	
M2D2V	514.125	519.5	520	520	520	0	0	75,450.66*
M3D2V	593.125	583.125	595	585	600	0.008	0.025	11,088.775
M3D2V1	539.75	535	540	535	540	0	0.009	13,574.583

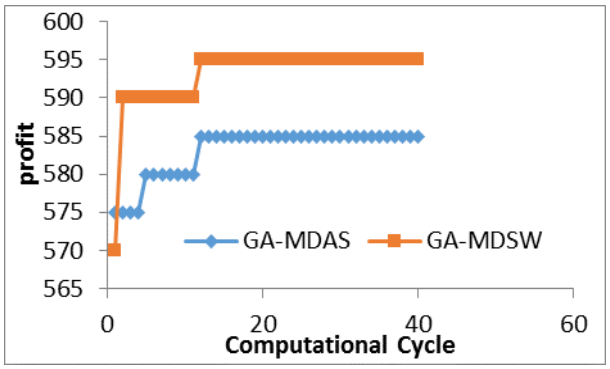
* Indicates CPLEX out of memory

The average profit and best profit for each instance are calculated for both GAs and shown in Table 6.8. Both GAs perform very well where nodes are well clustered. The relative error of each algorithm is less than 0.02. However, the solution of GA-MDSW is closer to optima comparing to results of GA-MDAS. In each cluster, the nodes are randomly dispersed around depot. Therefore, using GA-MDSW can result in a better solution over the same number of computational cycles, because in the initialization phase, routes are constructed from all nodes, while in GA-MDAS, routes are constructed cone by cone, and the diversity of its initial solutions is less than that of GA-MDSW. CPLEX CPU time to solve each problem is much higher than the CPU time required by GA-MDSW and GA-MDAS. The GA-MDSW takes 77.099 seconds to solve M2D2V, 75.200 seconds to solve M3D2V, and 78.310 seconds to solve M3D2V1 (i.e., less than 1% of CPLEX solution time).

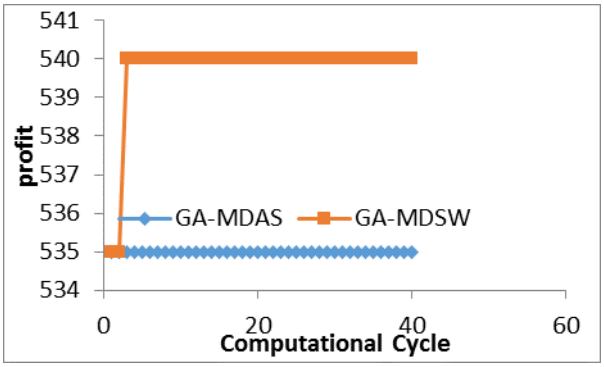
Profit changes as a function of the number of iterations for both GA approaches are depicted in Figures 6.7(a) to 6.7(c). It is clear that the profit of GA-MDAS converges faster and is higher than the GA-MDAS approach. In addition, in the case where clusters can be identified optimally, the GA approaches find a very close solution to optima.



(a) M2D2V



(b) M3D2V



(c) M3D2V1

Figure 6.7. Convergence trend of well-clustered instances

6.4.3.4 GA Evaluation for Random Distributed Nodes

To evaluate the performance of GA-MDSW and GA-MDAS on randomly distributed problems, nine different-size problems are generated. For each instance, two different scenarios are considered. In the first case, there are two depots having two vehicles per depot. In the second case, there are two depots, and each depot has a fleet of three vehicles. The budget amount for each instance is set such that it is difficult for CPLEX to find the optimal. Two algorithms, GA-MDSW and GA-MDAS, and CPLEX are used to solve these instances. The results are presented in Table 6.9. Furthermore, the convergence trend of each instance is shown in Figures 6.8 and 6.9.

Table 6.9. Results of GA Approaches for Different Size of Instances (2 depots and 2 vehicles)

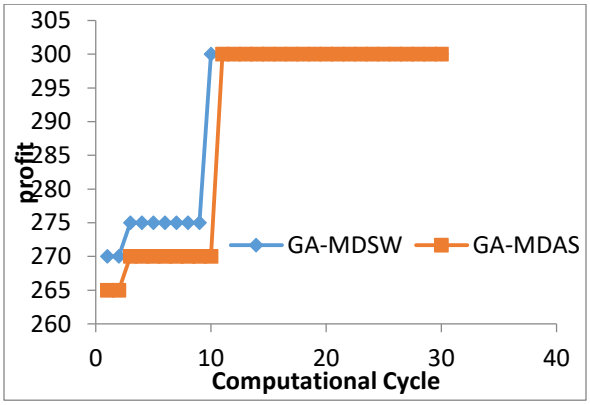
Name	Average Profit		Best Profit		Optimal Profit	RE		CPLEX Time (sec)
	GA-MDSW	GA-MDAS	GA-MDSW	GA-MDAS		GA-MDSW	GA-MDAS	
RA-32	292.167	289.67	300	300	300	0	0	1,527.033
RA-50	451.17	444	455	445	460	0.01	0.03	461.42
RA-60	577.875	549.25	580	550	600	0.03	0.08	15,406.63
RA-70	640	639	640	640	640	0	0	1041.832
RA-80	704.75	697.625	715	700	720	0.007	0.03	954.608
RA-90	797.375	777.875	805	785	820	0.02	0.04	62,028.43*
RA-100	888.5	881.1	890	890	940	0.05	0.05	13,699.014
RA-120	919.2	868.8	930	870	1000	0.07	0.1	178,029.603
RA-140	1103	1062.1	1105	1065	1120	0.01	0.05	406,266.385*

* Indicates CPLEX out of memory

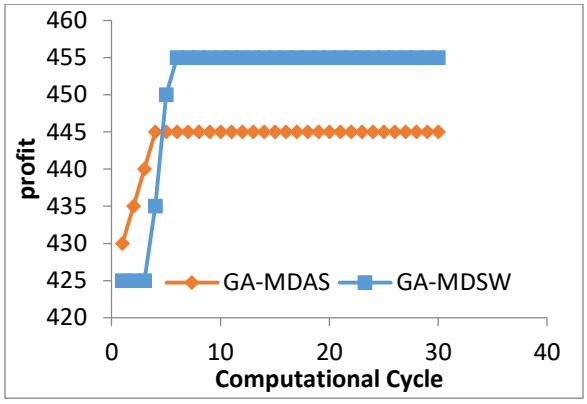
Table 6.9 shows the average profit and best profit obtained from running each GA with two depots and a set of two vehicles at each depot for each instance. The CPLEX time to determine the optimal solution is much higher than the time needed for the proposed GAs to solve the problem. For example, RA-140, RA-90, and RA-32 take 123.320, 43.372 and 9.141 seconds, respectively. Both GAs perform well, as the relative error is less than 0.08 for each instance. For most cases, the optimality gap of GA-MDSW is less than the relative error of GA-MDAS. GA-MDSW constructs the routes out of whole nodes, while GA-MDAS constructs routes at each cone, which decreases the diversity of solutions. Consequently, GA-MDSW results in better solutions over the same number of computational cycles.

Figures 6.8(a) to 6.8(i) show the convergence trend of each instance. The performance of GA-MDSW is considerably better than that of GA-MDAS. Because the instances are clustered,

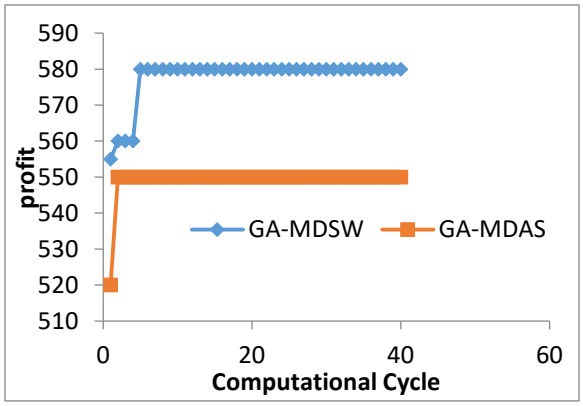
the nodes are not necessarily dispersed uniformly around the depot in each cluster. Consequently, GA-MDAS is not providing very good initial solutions due to poor cone construction.



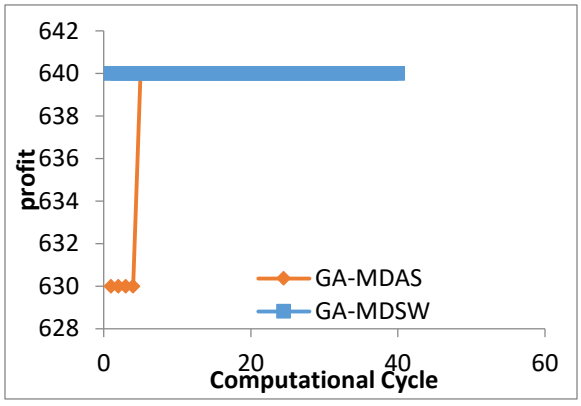
(a) RA-32



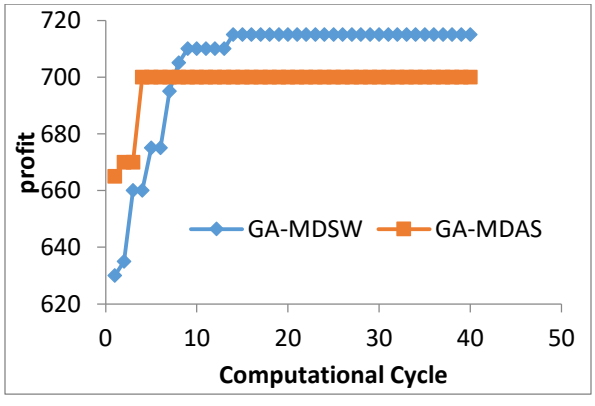
(b) RA-50



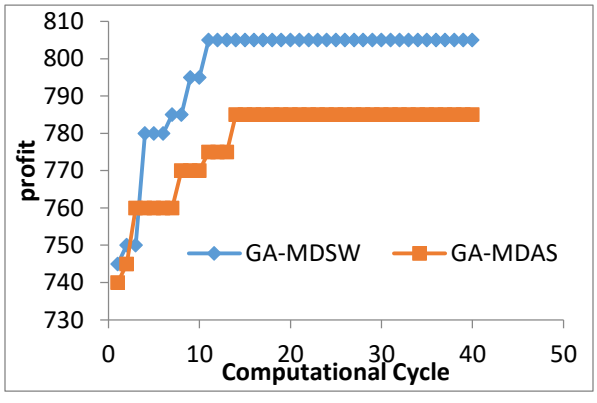
(c) RA-60



(d) RA-70

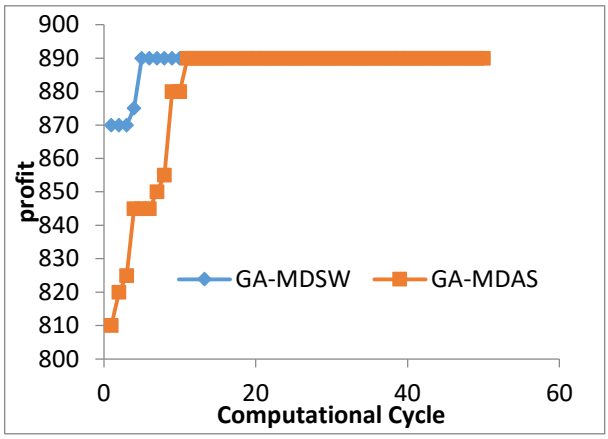


(e) RA-80

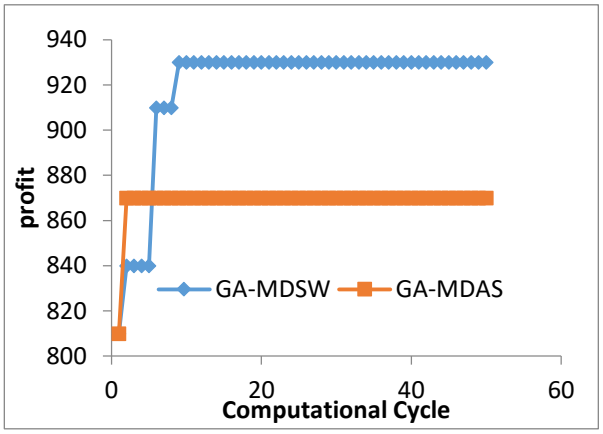


(f) RA-90

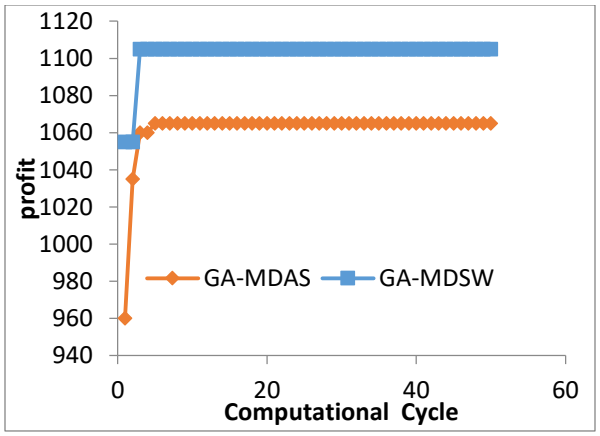
Figure 6.8. Trend of profit changes over different generations for instances with 2 depots and 2 vehicles



(g) RA-100



(h) RA-120



(i) RA-140

Figure 6.8. (continued)

The convergence trend of absolute error for different instances is shown in Figure 6.9. The absolute error (i.e., difference between optimal profit and GA's best profit) of both GAs decreases as the number of computational cycles increases. The absolute error of GA-MDSW is lower than GA-MDAS after a number of computational cycles for each instance. For some instances, the absolute error of GA-MDAS is lower than the absolute error of GA-MDSW in the very first computational cycles. In these instances, the cones are defined correctly due to uniform distribution of nodes around each depot.

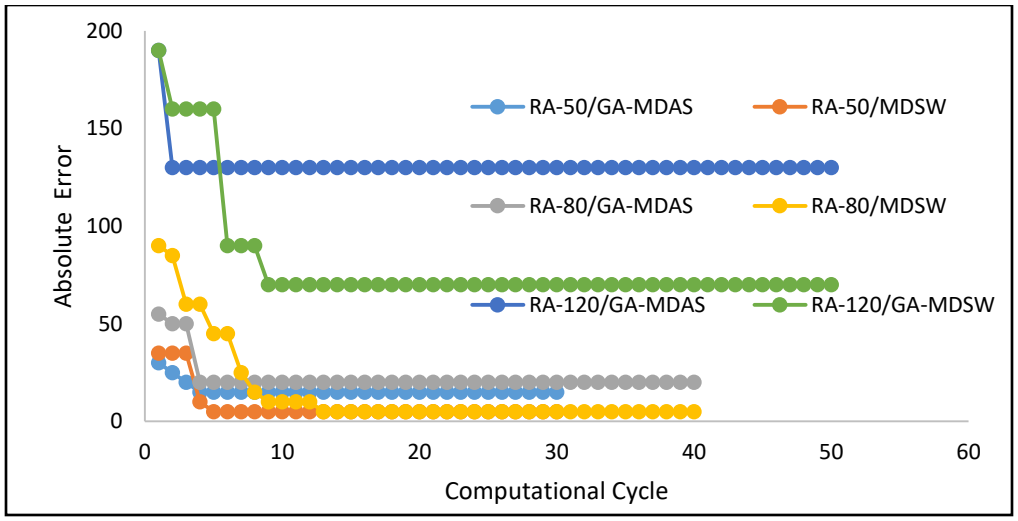


Figure 6.9. Trend of absolute error convergence for different instances with two vehicles

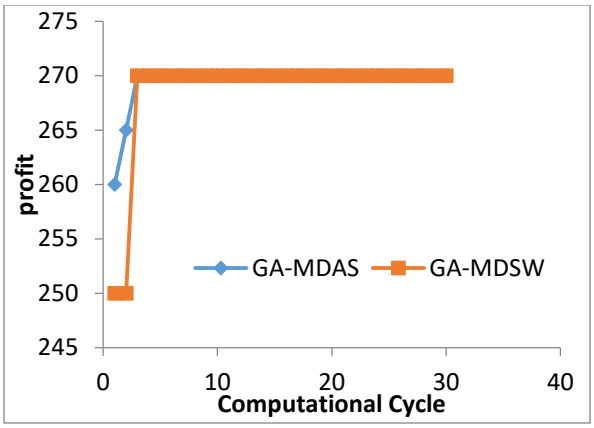
Table 6.10 shows the average profit and best profit obtained from running each GA with two depots and a set of three vehicles at each depot for different instances. Both GAs provide solutions with an absolute error less than 0.05. In larger-size instances, the optimality gap of GA-MDSW is less than the optimality gap of GA-MDAS. The average profit of GA-MDSW is greater than the average profit of GA-MDAS for all instances, due to the faster convergence of GA-MDSW to a better solution in comparison to GA-MDAS. As it can be seen, the computational time is very high for CPLEX, while Scala takes 251.430 seconds to solve large-scale problems (i.e., RA-140), 99.835 seconds to solve medium-size problems (i.e., RA-90), and 18.665 seconds to solve small-size problems (i.e., RA-32).

Table 6.10. Results of GA Approaches for Different-Size Instances (2 Depots and 3 Vehicles)

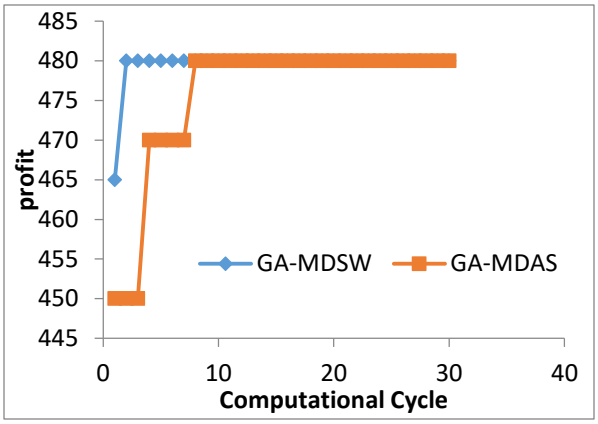
Name	Average Profit		Best Profit		Optimal Profit	RE (%)		CPLEX Time (sec)
	GA-MDSW	GA-MDAS	GA-MDSW	GA-MDAS		GA-MDSW	GA-MDAS	
RA-32	269.5	268.67	270	270	270	0	0	1,825.12
RA-50	479.5	475.67	480	480	480	0	0	20,377.26*
RA-60	568.67	538.33	570	540	570	0	0.05	1,267.771
RA-70	664.75	658.25	665	660	690	0.036	0.04	56,599.418
RA-80	727.25	718.375	730	720	750	0.026	0.04	166,184.805
RA-90	784	774.25	785	775	870	0.09	0.1	79,136.603*
RA-100	880.3	864.7	885	870	900	0.016	0.03	4,462.922
RA-120	986.1	987.3	990	990	1020	0.03	0.03	40,925.774*
RA-140	1204.1	1194.4	1205	1195	1260	0.04	0.05	83,408.52*

* Indicates CPLEX out of memory

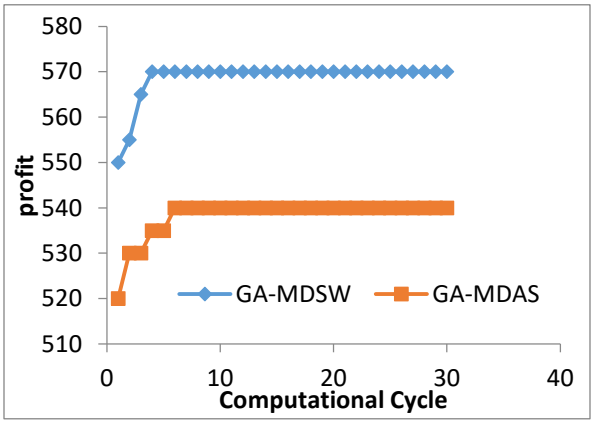
Figures 6.10(a) to 6.10(i) also show that GA-MDSW is providing better solutions for most cases. Although both GAs result in good solutions, the performance of GA-MDSW is better in the long run. For most instances, GA-MDAS does not provide good solutions in the very first computational cycles, because after clustering nodes, the distribution of nodes around the depot is not necessarily uniform. Consequently, cones are not defined very well and can result in poor initial solutions.



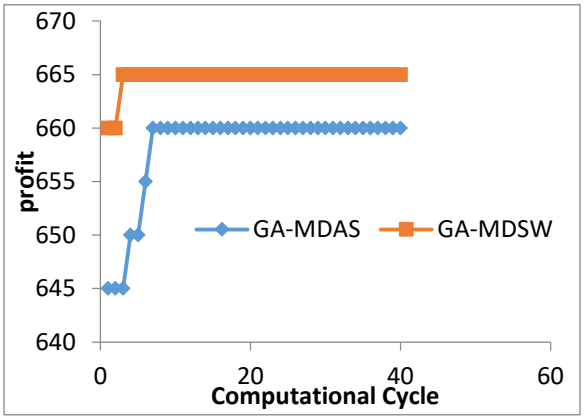
(a) RA-32



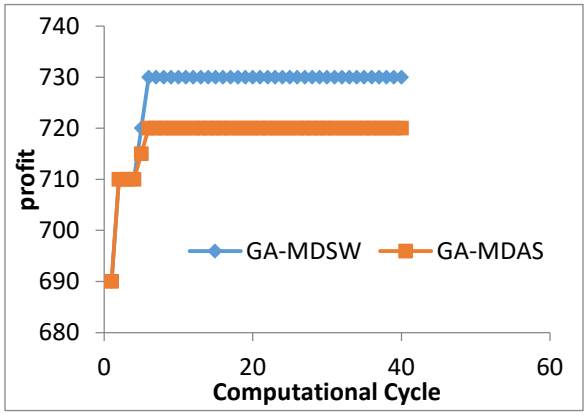
(b) RA-50



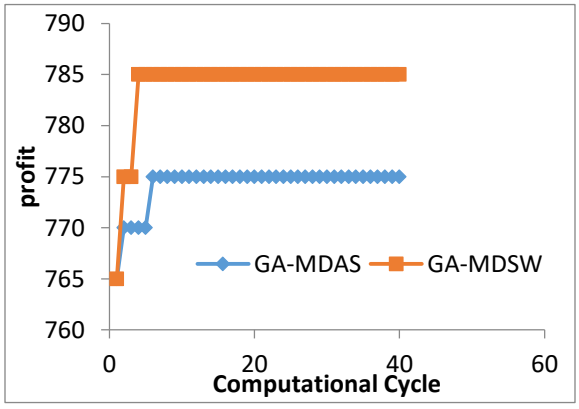
(c) RA-60



(d) RA-70

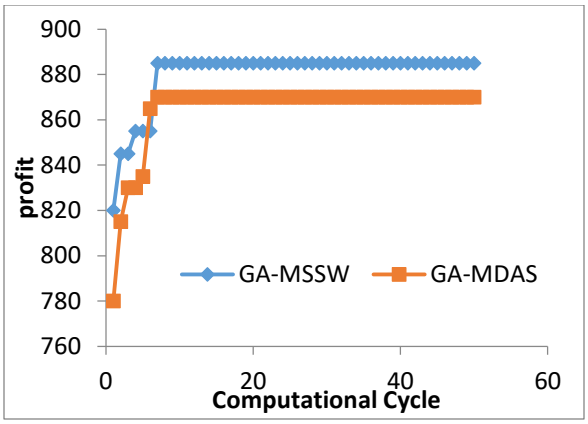


(e) RA-80

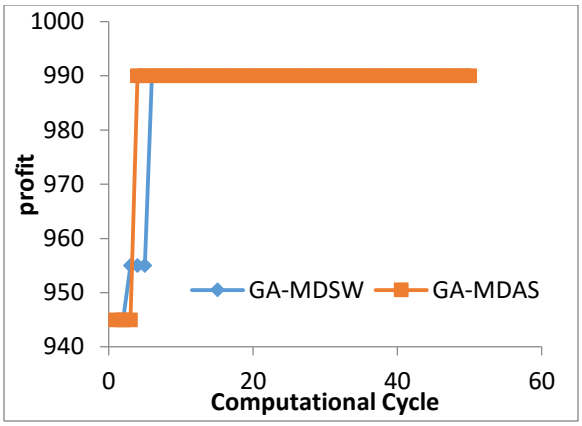


(f) RA-90

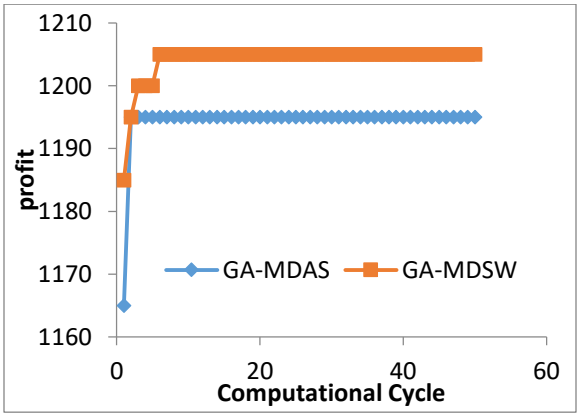
Figure 6.10. Trend in profit changes over different generations for instances with 2 depots and 3 vehicles



(g) RA-100



(h) RA-120



(i) RA-140

Figure 6.10. (continued)

The trend of absolute error changes of GAs for different instances with three vehicles is shown in Figure 6.11. As it can be seen, the absolute error of both GAs decreases over the same number of computational cycles. Note that GA-MDSW has a lower absolute error than GA-MDAS, for all instances.

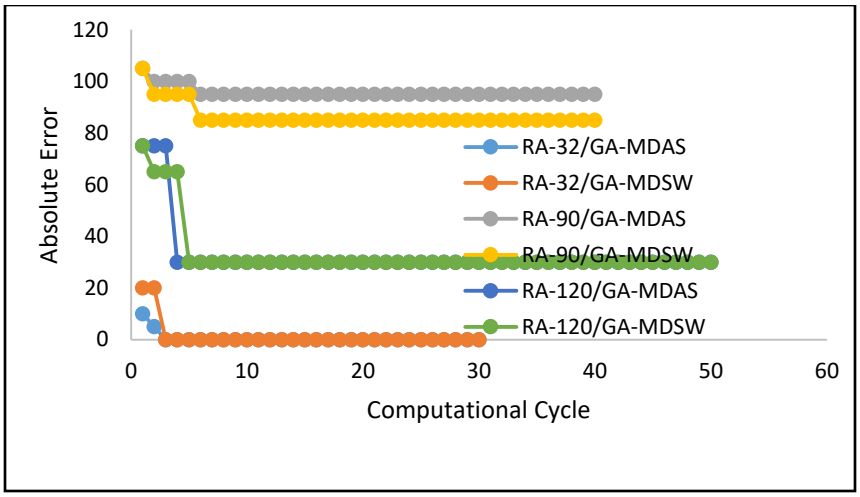


Figure 6.11. Trend in absolute error convergence for different instances with three vehicles

As it was shown in Table 6.9 previously, the CPLEX time to find the optima is very large. We capture the trend of time changes to find the optima for CPLEX and GA-MDSW for one instance (i.e., RA-90 with three vehicles), as shown in Figure 6.12. As can be seen, CPLEX finds the optima after 56,349 seconds, while it takes 5.567 seconds for GA-MDSW to find a solution with a relative error of 3%. The best solution that CPLEX finds in this time is at 0.

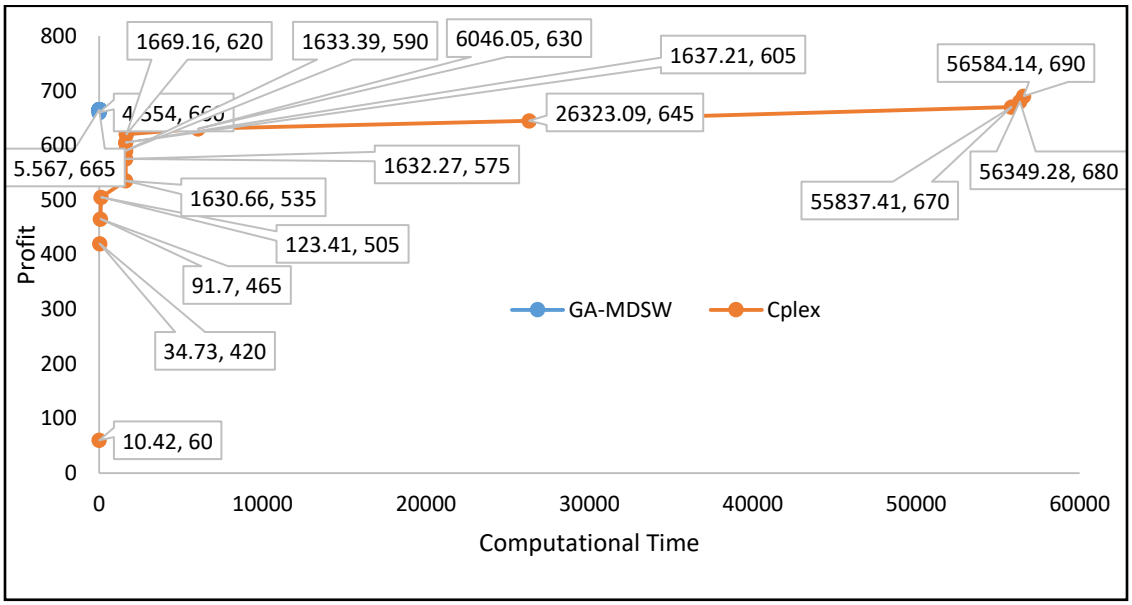


Figure 6.12. Comparison of computation time for CPLEX and GA-SW
(RA-70 with three vehicles)

Some observations can be seen from running GAs for different instances. First, for randomly dispersed instances, the k-means clustering algorithm provides a fairly good separation of nodes. Hence, both GAs—GA-MDSW and GA-MDAS—are performing well, based on this separation: a good clustering results in good initial solutions for GAs, and the convergence of solutions to optima is exhilarated. Depending on the distribution of nodes around the depot, the performance of GA-MDAS might be different. If nodes are not dispersed randomly around each depot, then the GA-MDAS will result in poor initial solutions, and converging to a good solution might become more difficult. GA-MDSW is performing well for all scenarios due to the diversity of the initial solutions population.

Second, where nodes can be fitted very well to clusters and nodes are dispersed randomly around each depot, both GAs provide very good solutions, primarily converging to optima. Although, the performance of GA-MDSW is better compared to GA-MDAS.

Third, among different clustering algorithms, k-means and k-medoids perform better than hierarchical clustering. k-means clustering results in more promising solutions than K-medoids clustering, although, the quality of solutions does not highly depend on the clustering method, as the result of performing inter-depot mutation. Inter-depot mutation tries to switch the searching solution to new areas by swapping some of the nodes between depots. It is still important to determine clusters well enough to start with good-structured solutions.

Finally, inter-depot mutation makes a great contribution toward improving solutions. In intra-depot mutation, each solution is improved based on the greedy 2-OPT algorithm, and then nodes are added to each route based on the closeness of each node's polar angle to the average polar angle of nodes on a given route.

6.5 Conclusions

A new multiple depot vehicle routing problem, MDCVRPWPC, is introduced in this chapter. Due to its complexity, the MILP cannot find the solution for large-scale problems in a reasonable amount of time. Moreover, the MILP execution time of an instance is variable due to the budget and tour duration amounts. If they are set to a very large or very small amount, then the execution time would be much lower, compared to the case where the budget and time are constrained at a medium range. In the first case, most cities can be on the route, whereas in the second case, only a few cities would be on the route.

Because this problem is NP-hard, two genetic algorithms—GA-MDSW and GA-MDSA—are designed. GA-MDSW always provides a closer solution to optima than GA-MDAS, regardless of node distribution. The diversity of the initial pool of solutions in GA-MDSW results in probing more search areas to find the optima. Although GA-MDAS can provide very good initial solutions, providing that nodes are fairly dispersed around each depot, the convergence of GA-MDSW to a closer solution to optima is better than GA-MDAS in the long run.

The performance of both GAs is not highly dependent on using the clustering algorithm, as long as the clusters are not poorly defined. The k-means algorithm provides a fair separation of nodes to clusters, and the results of running both GAs are considerably good compared to the case where clusters are poorly diagnosed.

The parameters for GAs play an important role in their performance. Generally, with larger parameters, the solutions are better. For instance, with a larger population size and iteration number, a better solution can be obtained. Furthermore, due to having lower diversification of solutions of the initial population in GA-MDAS, selection and mutation parameters are greater than their corresponding parameters in GA-MDSW.

This research could be extended to the case where there is a time window for each node in which a visit can occur. Moreover, some pick up as well as delivery could be considered at each node. In this case, the available capacity of each vehicle could vary based on the order of visits.

6.6 References

[1] Zuhori, S. T., Peya, Z. J., & Mahmud, F. (2012). A novel three-phase approach for solving multi-depot vehicle routing problem with stochastic demand. *Algorithms Research, 1*(4), 15-19.

[2] Sheu, J. B. (2007). An emergency logistics distribution approach for quick response to urgent relief demand in disasters. *Transportation Research Part E: Logistics and Transportation Review*, *43*(6), 687-709.

[3] Van Hentenryck, P., Bent, R., & Coffrin, C. (2010, June). Strategic planning for disaster recovery with stochastic last mile distribution. In *Integration of artificial intelligence (AI) and operations research (OR) techniques in constraint programming*. Lecture Notes in Computer Science Series (vol. 6140, pp. 318-333). Berlin Heidelberg: Springer.

[4] Golden, B. L., Levy, L., & Vohra, R. (1987). The orienteering problem. *Naval Research Logistics*, *34*(3), 307-318.

[5] Garey, M. R., & Johnson, D. (1979). *Computer and intractability: A guide to the theory of NP-completeness*, 1st ed. W. H Freeman.

[6] Wu, T. H., Low, C., & Bai, J. W. (2002). Heuristic solutions to multi-depot location routing problems. *Computers & Operations Research*, *29*(10), 1393–1415.

[7] Lim, A., & Zhu, W. (2006). A fast and effective insertion algorithm for multi-depot vehicle routing problem and a new simulated annealing approach. In M. Ali & R. Dapoigny (Eds.), *Advances in artificial intelligence*. Lecture Notes in Computer Science (vol. 4031, pp. 282–291).

[8] Polacek, M., Hartl, R. F., Doerner, K., & Reimann, M. (2005). A variable neighborhood search for the multi depot vehicle routing problem with time windows. *Journal of Heuristics*, *10*(6), 613–627.

[9] Polacek, M., Benkner, S., Doerner, K. F., & Hartl, R. F. (2008). A cooperative and adaptive variable neighborhood search for the multi depot vehicle routing problem with time windows. *Business Research*, *1*(2), 207–218.

[10] Clarke, G., & Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research, 46*, 93–100.

[11] Tillman, F.A. (1969). The multiple terminal delivery problem with probabilistic demands. *Transportation Science*, *3*, 192-204.

[12] Lim, A., & Wang, F. (2005). Multi-depot vehicle routing problem: A one-stage approach. *IEEE Transactions on Automation Science and Engineering*, *2*(4), 397–402.

[13] Maischberger M., & Cordeau J. -F. (2011). Solving variants of the vehicle routing problem with a simple parallel iterated Tabu Search. In *Network optimization*. Lecture notes in computer science, Vol. 6701 (pp. 395400).

[14] Bae, S.-T., Hwang, H. S., Cho, G.-S., & Goan, M.-J. (2007). Integrated GA-VRP solver for multi-depot system. *Computers & Industrial Engineering*, *53*(2), 233–240.

[15] Vidal, T., Crainic, T. G., Gendreau, M., Lahrichi, N., & Rei, W. (2012). A hybrid genetic algorithm for multi depot and periodic vehicle routing problems. *Operations Research*, *60*(3), 611-624.

[16] Vidal, T., Crainic, T. G., Gendreau, M., & Prins, C. (2014). Implicit depot assignments and rotations in vehicle routing heuristics. *European Journal of Operational Research*, *237*(1), 15–28.

[17] Ting, C. J., & Chen, C. H. (2009). Combination of multiple ant colony system and simulated annealing for the multi depot vehicle routing problem with time windows. *Transportation Research Record: Journal of the Transportation Research Board*, *2089*, 85–92.

[18] Wang, Z., Zhang, J., & Wang, X. (2011). A modified variable neighborhood search algorithm for the multi depot vehicle routing problem with time windows. *Chinese Journal of Management Science*, *19*(2), 99– 109.

[19] Filipec, M., Skrlec, D., & Krajcar, S. (1997, October). Darwin meets computers: New approach to multiple depot capacitated vehicle routing problem. In *Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation, 1997 IEEE International Conference on* (Vol. 1, pp. 421-426). IEEE.

[20] Filipec, M., Skrlec, D., Krajcar, S.: Genetic algorithm approach for multiple depot capacitated vehicle routing problem solving with heuristic improvements. *International Journal of Model ling & Simulation 20*, 320–328 (2000)

[21] Skok, M., Skrlec, D., & Krajcar, S. (2000). The genetic algorithm method for multiple depot capacitated vehicle routing problem solving. In *Knowledge-Based Intelligent Engineering Systems and Allied Technologies, 2000. Proceedings. Fourth International Conference on* (Vol. 2, pp. 520-526). IEEE.

[22] Skok, M., Skrlec, D., & Krajcar, S. (2000, June). The non-fixed destination multiple depot capacitated vehicle routing problem and genetic algorithms. In *Information Technology Interfaces, 2000. ITI 2000. Proceedings of the 22nd International Conference on* (pp. 403-408). IEEE.

[23] Thangiah, S. R., & Salhi, S. (2001). Genetic clustering: An adaptive heuristic for the Multi depot vehicle routing problem. *Applied Artificial Intelligence*, *15*(4), 361–383.

[24] Ombuki-Berman, B., & Hanshar, F. T. (2009). Using genetic algorithms for multi-depot vehicle routing. In *Bio-inspired algorithms for the vehicle routing problem* (pp. 77-99). Berlin Heidelberg: Springer.

[25] Gillett, B. E., & Miller, L. R. (1974). A heuristic algorithm for the vehicle-dispatch problem. *Operations Research*, *22*(2), 340-349.

[26] Fisher, M. L., & Jaikumar, R. (1981). A generalized assignment heuristic for vehicle routing. *Networks*, *11*(2), 109-124.

[27] Defryn, C., Sörensen, K., & Cornelissens, T. (2016). The selective vehicle routing problem in a collaborative environment. *European Journal of Operational Research*, *250*(2), 400-411.

[28] Miller, C. E., Tucker, A. W., & Zemlin, R. A. (1960). Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, *7*(4), 326-329.

[29] Hachicha, M., John Hodgson, M., Laporte, G., & Semet, F. (2000). Heuristics for the multi-vehicle covering tour problem. *Computers & Operations Research*,*27*(1), 29-42.

[30] Baker, B. M., & Ayechew, M. A. (2003). A genetic algorithm for the vehicle routing problem. *Computers & Operations Research*, *30*(5), 787-800.

[31] Srinivas, M., & Patnaik, L. M. (1994). Genetic algorithms: A survey. *IEEE Journal on Computer*, *27*(6), 11–26.

[32] Byung-In, K., Jae-Ik, S., & Min, Z. (1998). Comparison of TSP Algorithms–Project for Models in Facilities Planning and Materials Handling.

[33] Odersky, M& al. (2004). *An overview of the Scala programming language*. EPFL Lausanne, Switzerland

[34] Brooke, A., Kendrick, D., Meeraus, A., Raman, R., & America, U. (1998). The general algebraic modeling system. *GAMS Development Corporation*.

[35] CPLEX, I. I. (2009). V12. 1: User's Manual for CPLEX. *International Business Machines Corporation*, *46*(53), 157.

CHAPTER 7

CONCLUSIONS AND FUTURE RESEARCH

7.1 Conclusions

This dissertation focuses on introducing a new class of selective traveling salesman and vehicle routing problems. For each problem, a mathematical model is proposed, and each problem is solved as a mixed-integer linear program. The objective of each problem is to maximize the served demand. The results of exact methods are collected to create a benchmark to evaluate metaheuristics designed to solve large-scale problems. Because the TSP and VRP are NP-hard, new hybrid genetic algorithms are designed for each specific problem. The proposed HGAs consider the polar angle of the nodes and incorporate clustering algorithms to determine and improve solutions, which can be obtained by regular GAs. The impact of several factors (e.g., how nodes are dispersed in the plane, clusters, time limit, budget, etc.) is evaluated. The proposed solution frameworks will provide decision-makers or tour planners with efficient planning tools to schedule their tours with respect to budget and time constraints. Furthermore, the proposed frameworks can be utilized to solve large-size routing problems with limited resources. The following sections provide a short summary and conclusions for each problem presented in this dissertation.

7.1.1 Conclusion for Chapter 3

Chapter 3 proposes a new variant of the traveling salesman problem called the traveling salesman problem with partial coverage. The objective of this problem to maximize the served demand, given budget and travel time constraints. A partial demand of the nodes that are not on the route can be maintained by nodes on the route. Each node has a limited capacity to serve the other nodes, and there is a limited distance to serve other nodes that are not on the route. A hybrid

genetic algorithm is developed to solve the TSPWPC. In this approach, each route is captured as a chromosome, and a one-point crossover procedure is performed on the initial randomly generated solutions pool. To measure the fitness of each solution, an assignment optimization problem is solved. Moreover, instead of mutation, a local optimization is implemented to improve the solution: based on a clustering algorithm, the nodes are reorganized in a given tour. Thereafter, out of nodes that are not on the route, those that have the closest polar angle to the average polar angle of nodes on the route are inserted into the route. To evaluate the efficiency of the proposed genetic algorithm, another GA is designed, whereby an insertion mutation is implemented. Both GAs are coded in C. Furthermore, each problem is solved as an MILP to evaluate their efficiency. In this exact method, dynamic cuts are added to the problem in order to eliminate the subtours. Using these dynamic cuts decreases the computational time.

Results show that the GA with local optimization provides good solutions for the TSPWPC. If the nodes are clustered, preprocessing with clustering optimization helps the GA-cluster method to find good solutions: clustering optimization can be done with respect to the number of clusters and also with respect to which clustering method is utilized. Moreover, providing that the number of clusters is optimized, the convergence rate increases, and a closer solution to optima is found. Also, the k-means clustering algorithm provides a better separation of nodes, and the results of GAs using k-means clustering are more promising in comparison to other clustering algorithms.

It is also observed that GA parameters play an important role in their performance. As the population size increases, the mutation rate also slightly increases to conserve the diversity of solutions.

7.1.2 Conclusion for Chapter 4

Chapter 4 presents a new class of vehicle routing problem, the capacitated vehicle routing problem with partial coverage. In this problem, there is a fleet of vehicles departing from a single depot having a known capacity. The objective here is maximizing the served demand. A partial demand of nodes not on the route can be maintained by the nodes that are on the route. Each node has a limited capacity and can serve other nodes providing that their distance is less than a predetermined distance. Each vehicle has a limited budget and travel time. Two new genetic algorithms are developed to solve the problem in a reasonable computational time. In this approach, two different methods are used to construct the initial solutions. The first method is inspired from the sweep method, and the other is inspired from the assignment method. In the sweep method, nodes are assigned to vehicles randomly and sorted based on their polar angles. In the assignment method, the plane is partitioned into cones according to the number of vehicles, and nodes are assigned to each vehicle based on a tradeoff between their potential prize and the cost of assignment. Each solution is captured as a chromosome, and a two-point crossover is implemented on the transformed chromosomes. The goal of this crossover method is to improve the chance of serving nodes in the same neighborhood by the same vehicle. An insertion mutation is implemented after improving solutions based on swapping some nodes in the given route. To evaluate the efficiency of GAs, each instance is solved as an MILP, and the results of GAs are compared to results of the exact method.

Results show that both GAs are providing good solutions in a reasonable amount of time. Although the GA-AS gives better initial solutions, providing that nodes are fairly dispersed around the depot, the GA-SW provides a closer solution to optima in the long run. If nodes are not dispersed randomly around the depot, then the GA-AS initializes poor solutions and it can find a

local optima, while GA-SW provides good solutions, regardless of the distribution of nodes around the depot. It is also observed that it is important to set GA parameters correctly, in order to accelerate the convergence rate of GAs. To conserve the diversity of the solution in the GA-AS, selection and mutation rates are slightly greater than the selection and mutation rates of the GA-SW.

7.1.3 Conclusion for Chapter 5

Chapter 5 focuses on introducing a variant to the multiple traveling salesman problem, called the multiple traveling salesman problem with partial coverage. In this problem, there is a set of travelers to serve a set of nodes. Each traveler cannot be on a route more than a limited period of time. A fixed cost is associated with each traveler, and there is a limited budget for the entire traveling cost and assignment cost of travelers. Due to having a fixed cost, the number of travelers is not fixed. The objective here is to maximize the served demand. Each node has a limited capacity, and it can serve a partial demand of other nodes not on the route, providing that their distance is less than a predetermined distance. Two new genetic algorithms are proposed to solve the problem in a reasonable amount of time. The algorithms differ in their mutation procedure. In one algorithm, an in-route and cross-route mutation are implemented, while in the other algorithm, a greedy 2-OPT mutation and a cross-route mutation are implemented. To construct the initial solution, due to not having a fixed number of travelers, a method inspired from the sweep method is implemented. The crossover procedure is similar to the one used in the CVRPWPC. To evaluate the efficiency of the proposed GAs, each problem is solved as an MILP, and results of the GAs are compared to the results of the exact method.

Results show that the GA-2-OPT is providing a closer solution to optima than the GA-IR. For cases where the feasible region is tight, the GA-2-OPT is performing better than the GA-IR.

As a result of performing greedy 2-OPT optimization on each solution, the chance of improving the solution increases, and more nodes can be added to a given route. Setting the parameters is important to achieving a good performance of the GAs.

7.1.4 Conclusion for Chapter 6

Chapter 6 presents a new class of the multiple depot vehicle routing problem called the multiple depot capacitated vehicle routing problem with partial coverage. In this problem, there is a set of depots, and at each depot, there is a fleet of vehicles. Each vehicle has a limited capacity, budget, and traveling time. The objective here is to maximize the served demand. Each node has a limited capacity and can serve other nodes not on the route according to its capacity, providing that their distance is less than a predetermined distance. Two new genetic algorithms are developed to solve this problem. The GAs are different in the initialization step. One of them is inspired from the sweep method, and the other is inspired from the assignment method, similar to the CVRPWPC. In these algorithms, nodes are assigned to depots based on a clustering algorithm, and the interaction of solutions is considered. A crossover similar to the one implemented in the CVRPWPC is performed on the solutions of one depot at each generation. Two mutation procedures, inter-depot and intra-depot mutation, are implemented on the solutions. The first one tries to change the borderlines of the clusters by swapping some nodes among depots. This mutation is implemented every ten generations. In the second one, a greedy 2-OPT mutation is performed on the solutions of one depot at each generation, and an insertion mutation is implemented on those improved solutions thereafter. To evaluate the efficiency of the GAs, each problem is solved as an MILP, and the GA results are compared to those of the exact method.

Results show that the GA-MDSW is providing a closer solution to optima than the GA-MDAS. The quality of GA-MDAS solutions depends on the distribution of nodes around each

depot. Providing that nodes are fairly dispersed around each depot, the GA-MDAS can provide very good initial solutions and can converge to a very close solution to optima. GA performance does not strongly depend on the clustering algorithm, as long as nodes are well separated due to implementing the inter-depot mutation. If the clusters are defined poorly, then converging to a good solution can be very difficult. Setting the GA parameters plays an important role in their performance. The mutation rate of GA-MDAS is slightly greater than that of GA-MDAS, in order to conserve diversity.

7.1.5 Summary of Contributions of This Dissertation

In this dissertation, a new class of selective traveling salesman and vehicle routing problems is introduced. The objective of these problems is to maximize the collected prize, subject to limited resources. The contributions of this dissertation can be summarized as follows:

- A new class of selective traveling salesman and vehicle routing problems, which are a combination of prize collecting and covering salesman problems, is introduced.
- Exact methods and metaheuristics are developed to solve each model presented.
- Using the solution frameworks developed for these selective TSPs and VRPs, the decision-maker has the opportunity to implement efficient planning tools to schedule vehicle tours with respect to budget and time constraints.
- Several performance measures are used to analyze and explore the proposed metaheuristics dynamics, in order to greatly evaluate the quality of solutions.

The metaheuristics proposed in this dissertation may be used to solve a large-size routing problem subject to different conditions. However, section 7.2 explores the conditions and possible extensions that will define future research.

7.2 Future Research

This dissertation introduces a new class of selective TSP and VRP with the objective of maximizing the served demand. A direct extension of the proposed mathematical models could be applied to problems where there are time window constraints on served nodes. The uncertainty of demand could be considered to address problems with the presence of limited resources to capture information. Different objectives could be added to the problem, and a multi-objective routing problem could be defined. Moreover, the packing pattern of items could be taken into consideration, and a combination of the routing problem and the loading problem could be addressed.

7.2.1 Time Windows Consideration for Each Introduced Problem

In this dissertation, it is assumed that there is no restriction on the time that a node can be visited. The vehicle routing problem could be extended to include time windows. In this problem, in addition to the capacity constraint, each vehicle must visit each node within its time window. There can be one or multiple time windows for each node. Different heuristics and metaheuristics would be used to solve the VRPTW. And different construction heuristics could be considered for constructing the initial solutions. Some of these are savings-based heuristics [1], whereby first, single customers are considered as a single route, and then, based on the maximal savings, the routes are merged. Other heuristics could be the nearest-neighborhood insertion-based method [2].

To continue this research, a time window constraint could be added to each problem addressed in this dissertation. (i.e., each node could be served within its time limit). A metaheuristic similar to the metaheuristics developed could be expanded to solve these new problems. Then, time window constraints must be considered to construct the initial solutions.

7.2.2 Considering Demand Uncertainty at Each Node

In this dissertation, it is assumed that demand at each node is known and certain. However, in many real-world problems, uncertainty relative to demand should be taken into consideration. Two types of uncertainties regarding optimization under uncertainty are the following: In the first, information about an uncertain parameter is updated over time, while in the second, there is no information about the uncertain parameter, and there are no resources to capture the data. An example is the case where a disaster occurs and quick decisions must be made. There are considerable literature reviews about the vehicle routing problem under uncertainty. In some of them, resources exist and approaches are developed accordingly [3, 4, 5], while others consider both cases of not having information and also having information at the beginning of the day [6].

To continue this research, a study could be developed to determine the effect of having uncertain demand for problems addressed in this dissertation. Both cases of having access to information and not having available information could be considered. In these types of problems, the objective could be maximizing the reliability of performing as expected, or it could be a robust optimization, which is more risk-averse.

7.2.3 Multiple-Objective Vehicle Routing Problems with Partial Coverage

Three types of the vehicle routing problem are based on the objective. The objective can be maximizing profit, minimizing cost, or a combination of both. In most studies, one of these objectives is considered the objective, while the other is limited as a constraint. There are many techniques to solving the multi-objective vehicle routing problem, and they can be categorized into three different groups: scalar method, Pareto method, and other methods different from the previous methods. Most works using the scalar method aggregate the objectives and transform the problem to a single-objective problem [7, 8], while others use different approaches, such as the

ε-constraint approach [9]. Many works use the Pareto approach to tackle the problem as well [10, 11].

To continue this research, some cost-related constraints could be treated as objectives. Because there are conflicting objectives, the problem will be considered as a multi-objective vehicle routing. Different strategies such as goal programming and genetic algorithms could be used to solve the problem.

7.2.4 TSP with Partial Coverage and Three-Dimensional Loading Constraints

The loading problem is an NP-hard problem. Loading a vehicle is important, because using the capacity of a vehicle optimally reduces the number of vehicles needed to be on the route, and also the order of goods delivered to different locations will have an impact on the loading procedure. Some studies involve combinatorial optimization of the VRP and two-dimensional packing [12, 13]. The first paper that worked on the vehicle routing problem and three-dimensional loading was by Gendreau et al. [14] who considered a variety of loading constraints such as fragility, support, and last-in first-out (LIFO) constraints.

To continue this research, the packing problem could be considered as well. This would make the problem more complex because the nodes that are chosen to be on the route might differ in terms of type of item that can be received, and this will have effect on the packing approach.

7.3 References

[1] Clarke, G. U., & Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, *12*(4), 568-581.

[2] Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, *35*(2), 254-265.

[3] Chow, J. Y., & Regan, A. C. (2011). Network-based real option models. *Transportation Research Part B: Methodological*, *45*(4), 682-695.

[4] Powell, W. B., Simao, H. P., & Bouzaiene-Ayari, B. (2012). Approximate dynamic programming in transportation and logistics: a unified framework. *EURO Journal on Transportation and Logistics*, *1*(3), 237-284.

[5] Gan, L. P., & Recker, W. (2013). Stochastic preplanned household activity pattern problem with uncertain activity participation (SHAPP). *Transportation Science*, *47*(3), 439-454.

[6] Van Hentenryck, P., Bent, R., & Coffrin, C. (2010, June). Strategic planning for disaster recovery with stochastic last mile distribution. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming* (pp. 318-333). Berlin Heidelberg: Springer.

[7] Park, Y. B., & Koelling, C. P. (1986). A solution of vehicle routing problems in a multiple objective environment. *Engineering Costs and Production Economics*, *10*(2), 121-132.

[8] Park, Y. B., & Koelling, C. P. (1989). An interactive computerized algorithm for multicriteria vehicle routing problems. *Computers & Industrial Engineering*, *16*(4), 477-490.

[9] Pacheco, J., & Martí, R. (2006). Tabu search for a multi-objective routing problem. *Journal of the Operational Research Society*, *57*(1), 29-37.

[10] Rahoual, M., & Djoukhdjoukh, W. (2003). Métaheuristique hybride Pareto pour le probleme de tournées de véhicules avec fenêtres horaires. In *Evolution artificielle* (Vol. 2003, pp. 380-395).

[11] Rahoual, M., Kitoun, B., Mabed, M. H., Bachelet, V., & Benameur, F. (2001, July). Multicriteria genetic algorithms for the vehicle routing problem with time windows. In *4th Metaheuristics International Conference* (pp. 527-532).

[12] Iori, M., Salazar-González, J. J., & Vigo, D. (2007). An exact approach for the vehicle routing problem with two-dimensional loading constraints. *Transportation Science*, *41*(2), 253-264.

[13] Leung, S. C., Zhang, Z., Zhang, D., Hua, X., & Lim, M. K. (2013). A meta-heuristic algorithm for heterogeneous fleet vehicle routing problems with two-dimensional loading constraints. *European Journal of Operational Research*, *225*(2), 199-210.

[14] Gendreau, M., Iori, M., Laporte, G., & Martello, S. (2006). A tabu search algorithm for a routing and container loading problem. *Transportation Science*, *40*(3), 342-350.