# ABSTRACT

BAHMANINEZHAD, AZADEH. Exact and Heuristic Approaches for Solving the Multi-Vehicle Advance Request Dial-a-Ride Problem. (Under the direction of Dr. John Baugh).

The dial-a-ride problem is a subclass of vehicle routing problems with pick-up and delivery requirements, a problem that has been shown to be NP-hard. In the dial-a-ride problem, customers make pick-up and/or drop-off requests for transportation that is provided by a transportation center. Customers may make an outbound (drop-off) request for transportation from their home to the hospital, for instance, and an inbound (pick-up) request for transportation to return home. The transportation center satisfies the requests through the use of shared-ride vehicles that depart from a single depot. In general terms, the goal of the dial-a-ride problem is to produce routes that satisfy customer requests with minimum cost and inconvenience while considering side constraints such as limited capacities for vehicles, time windows for picking up and dropping off customers, and specified limits on the duration of each route.

In this dissertation, the characteristics of a practical dial-a-ride service are identified, and a formal definition of the problem is given. Several different solution methods are developed. A mixed integer linear programming (MILP) model is formulated so that solvers such as Gurobi and CPLEX can be used to obtain exact solutions on small problem instances. For solving larger single vehicle dial-a-ride problems, algorithms are developed that improve on two popular and computationally efficient approaches: a space-time nearest neighbor heuristic and an insertion heuristic. Consistent with the MILP formulation, service policies and objectives adopted in the new algorithms are consistent with the practical dial-a-ride characteristics previously identified. Although the results they produce are not guaranteed to be optimal, a number of experiments are performed to evaluate the quality of the solutions and the computational performance of the algorithms. Results show that the

improvements made lead to better solutions, which can be obtained at a modest computational cost since the new algorithms retain the computational complexity of their original counterparts. For solving larger multi-vehicle dial-a-ride problems, a cluster-first, route-second approach is developed that makes use of the foregoing algorithms. For assigning passengers to vehicles (clustering), a genetic algorithm is used, and for scheduling the service order and times within a cluster (routing), the improved space-time nearest neighbor and insertion heuristic algorithms are used, both separately and in combination. A variety of experiments are performed to evaluate the quality and computational performance of the routers in the context of this multi-vehicle optimization framework.

# Exact and Heuristic Approaches for Solving the Multi-Vehicle Advance Request Dial-a-Ride Problem

by
Azadeh bahmaninezhad

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Operations Research

Raleigh, North Carolina
2019

APPROVED BY:

| | |
|---|---|
| Dr. Michael Kay | Dr. Yahya Fathi |
| Dr. Thomas Reiland | Dr. John Baugh<br>Chair of Advisory Committee |

# DEDICATION

To my parents, my brother, and my two sisters. Without their support and love, none of this would be possible.

# BIOGRAPHY

Azadeh Bahmaninezhad has received her B.S. degree in Applied Mathematics from Shahid Chamran Univeristy, Ahvaz, Iran in 2009 and her M.S. degree in Applied Mathematics from Amirkabir University of Technology, Tehran, Iran in 2013.

Azadeh joined the Operations Research Graduate Program at North Carolina State University in August 2015. Her research interests include network optimization, supply chain optimization, and more specifically, designing different algorithms for dial-a-ride problems and vehicle routing problems.

# ACKNOWLEDGEMENTS

I would like to express my sincerest gratitude to my advisor Dr. John Baugh for his supervision, advice, patience, and his continued support and encouragement. There is no doubt that I would not be able to complete this work without his support, and guidance. He has always been available for me with his guidance and wisdom which helped me learn and grow in many ways. I am truly grateful to have him as my mentor.

I also would like to thank my committee members, Dr. Yahya Fathi and Dr. Michael Kay and Dr. Thomas Reiland who offered guidance and support. In particular, I am grateful to Dr. Yahya Fathi for his continuous support during my graduate study.

Last but not least, I would like to thank my friend, Hossein Tohidi, for his unconditional support, encouragement and help.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

CHAPTER

# 1

# INTRODUCTION

As a problem category, vehicle routing with pick-up and delivery refers to problems in which goods and/or people are transported between pick-up and delivery locations. It includes several sub-problems, as described by Parragh et al. (2008) and depicted in Figure 1.1. The unpaired vehicle routing problem with pick-up and delivery refers to problems in which an item picked-up from a location can be dropped off at any delivery location. Whereas in the paired vehicle routing problem with pick-up and delivery, all pick-up and delivery locations are paired: an item picked-up from a location is dropped off at a specified delivery location.

Unpaired vehicle routing problems with pick-up and delivery include two sub-problems: PDTSP, the pick-up and delivery traveling salesman problem, and PDVRP, the pick-up and

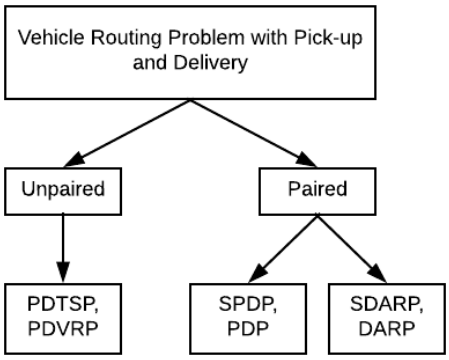


Figure 1.1: The class of vehicle routing problems with pick-up and delivery

delivery vehicle routing problem.

Paired vehicle routing problems with pick-up and delivery include two sub-categories. The first consists of two sub-problems: SPDP, the single vehicle pick-up and delivery problem, and PDP, the pick-up and delivery problem. The second consists of two sub-problems: SDARP, the single vehicle dial-a-ride problem, and DARP, the dial-a-ride problem. SPDP and PDP deal with the transportation of goods, whereas SDARP and DARP deal with the transportation of people.

In this dissertation, the focus is on the dial-a-ride problem (DARP), both single and multi-vehicle, as described in the following sections.

## 1.1 Problem Statement and Motivation

The dial-a-ride problem is a subclass of vehicle routing problems with pick-up and delivery, where customers make pick-up and drop-off requests that are serviced by a transportation center. The application of the problem mostly arises in door-to-door transportation for elderly or disabled people. They usually make an outbound (drop-off) request for trans-

portation from their home to a destination, such as a treatment center or grocery store, and an inbound (pick-up) request for transportation to return home.

The transportation center satisfies these requests using a fleet of vehicles that start from a common depot. Each vehicle provides a *shared-ride* service to the customers, meaning there might be more than one customer in the vehicle at the same time. In general terms, the goal is to produce routes that satisfy customer requests while minimizing transportation costs and customer inconvenience. Side constraints may include the following, for instance: each vehicle has a limited capacity, there is a time window for pick-up and drop-off for each customer, and the duration of each route cannot exceed a specified limit.

There are two types of dial-a-ride problem that may be encountered, static and dynamic. In the static dial-a-ride problem, requests and all related information are known prior to scheduling. However, in the dynamic dial-a-ride problem, the information becomes available gradually over time. In practice, transportation centers have generally found the dynamic problem to be too costly to implement because it limits scheduling options, and instead they tend to adopt a static policy of collecting requests ahead of time, usually a day or more in advance. Given its widespread applicability, the static dial-a-ride problem is the focus of this study.

Stakeholders of dial-a-ride services include transportation centers and their customers, whose interests are only partially aligned and are competing in some respects. Routes that minimize costs, or perhaps as a surrogate, distance traveled, are needed to make the service economically feasible. On the other hand, routes that minimize customer inconvenience are more desirable from the customers' perspective, as one would imagine, where inconvenience might be measured in terms of departure from specified time windows and excess ride times. Taken together, such concerns mean that designing a practical dial-a-ride service is a challenging task.

The issue of stakeholder interests is addressed in two ways in the literature. Some

researchers consider an objective function that minimizes operation cost subject to constraints that maintain a good quality of service for customers. Others formulate the problem with multiple objective functions: one to minimize operating cost and another to minimize factors causing customer inconvenience.

In this study, the dial-a-ride problem is formulated with multiple objectives for minimizing operating cost as well as customer inconvenience. It addresses customer satisfaction factors in new ways that may be considered more practical from the perspective of the customer, overcoming some of the limitations of prior approaches, as discussed in the following chapters. The goal of the research is to design mathematical models and heuristic algorithms that more carefully take into account these customer satisfaction factors.

## 1.2 Basic Terminology

In this section, some basic terms used in the dissertation are defined.

- **EPT**, or early pick-up time: the earliest time a customer can be picked up without causing a time window violation.
- **LPT,** or late pick-up time: the latest time a customer can be picked up without causing a time window violation.
- **EDT,** or early drop-off time: the earliest time a customer can be dropped off without causing a time window violation.
- **LDT,** or late drop-off time: the latest time a customer can be dropped off without causing a time window violation.
- **Hard Time Window:** Each customer must be picked-up in the (*EPT*, *LPT*) interval and dropped-off in the (*EDT*, *LDT*) interval.

- **Soft Time Window:** Each customer can be picked-up or dropped-off any time (no restriction).

- **Excess Ride Time:** Ride time for a customer is minimal when the vehicle travels directly from the customer's origin to the requested destination without serving other customers. The difference between actual ride time and the minimum ride time of a customer is called excess ride time.

- **Time Window Violation:** Each customer has predetermined time windows for both pick-up from the origin and drop-off to the destination. A customer being picked-up earlier than EPT or later than LPT, or dropped-off earlier than EDT or later than LDT, constitutes a time window violation.

- **Waiting Time:** When a vehicle arrives earlier than the EPT and must wait to pick up the customer, or arrives earlier than the EDT and (unnecessarily) waits to drop off the customer, the time that the vehicle idles is the waiting time.

- **Total Travel Time:** The duration of time starting from when the vehicle leaves the depot until it arrives again at the depot, minus any waiting time.

## 1.3 Contributions

The contributions of this research can be summarized as follows:

#### Problem Redefinition

The various types of dial-a-ride problems generally take both customer inconvenience and operating cost in consideration. However, they may incorporate these factors differently in their models and formulations. Route length and travel time affect the operating cost. On

the other hand, excess ride time, departure from specified time windows, and waiting time affect the customer satisfaction and convenience.

This dissertation addresses customer satisfaction in new ways, particularly with respect to time windows and waiting times. To increase the customer satisfaction, the following policies are adopted:

- Customers are not picked up before their EPT. Therefore, if a vehicle arrives earlier than the EPT it must wait and allow the customer to board at the EPT.
- If a vehicle arrives earlier than the EDT, the customer is allowed to depart from the vehicle immediately on arrival (as one would expect).
- Vehicles may arrive later than a customer's LPT or LDT, but for outbound (drop-off) requests, arriving later than the LDT is heavily penalized.
- Time window violations are classified and maintained as four separate categories. Each is given a weight commensurate with its importance, and a weighted combination is minimized as part of the objective function.
- Apart from waiting times, vehicles do not idle as they pick up and drop off customers.

A more complete explanation of these policies is provided in subsequent chapters.

**Solution Approaches**

In this dissertation, exact, heuristic, and meta-heuristic approaches for the dial-a-ride problem are developed. A brief description of each method is given below.

*Exact solution approach*

- A mixed integer programming model has been developed for a version of the dial-a-ride problem that addresses customer satisfaction concerns. The model can be used

to solve both the single and multi-vehicle dial-a-ride problems. Since the dial-a-ride problem is known to be NP-hard, only small problem instances can be solved to optimality using this exact approach.

*Heuristic solution approaches for the single vehicle dial-a-ride problem*

- The space-time nearest neighbor heuristic algorithm proposed by Baugh et al. (1998) is a popular algorithm. A detailed description and assessment are given, along with a critique that suggests directions for improvement in the algorithm.
- To generate better and more practical solutions, improvements have been made to the space-time nearest neighbor heuristic algorithm.
- An insertion heuristic algorithm for the dial-a-ride problem is described, and improvements have been made to the insertion heuristic algorithm.

*Meta-heuristic solution approaches for multi-vehicle dial-a-ride problem*

- A cluster-first, route-second approach is defined to solve the multi-vehicle dial-a-ride problem. A genetic algorithm is developed for the clustering, and the various heuristic algorithms introduced for solving the single vehicle dial-a-ride problem can be used for the routing.

## 1.4 Structure of the Dissertation

The remainder of the dissertation is organized as follows.

**Chapter 2:** A brief literature review on solution approaches for the single and multi-vehicle dial-a-ride problems is provided.

**Chapter 3:** The dial-a-ride problem is defined formally, and a new mathematical programming formulation of the problem is presented.

**Chapter 4:** Descriptions of the space-time nearest neighbor heuristic algorithm and an insertion heuristic algorithm are given, and improvements to the algorithms for solving the single vehicle dial-a-ride problem are presented.

**Chapter 5:** Experimental results on the single vehicle dial-a-ride problem are presented.

**Chapter 6:** A genetic algorithm for solving the multi-vehicle dial-a-ride problem is described.

**Chapter 7:** Experimental results on the multi-vehicle dial-a-ride problem are presented.

**Chapter 8:** Conclusions and future work are given.

CHAPTER

# 2

# LITERATURE REVIEW

Over the past decades the dial-a-ride problem has been studied extensively, and optimization methods for the problem have received considerable attention. This chapter reviews the literature on the dial-a-ride problem and the solution methods proposed. It begins with the single vehicle dial-a-ride problem and follows with the multi-vehicle dial-a-ride problem.

## 2.1 Single Vehicle Dial-a-Ride Problem

The single vehicle dial-a-ride problem is the simpler version of the multi-vehicle dial-a-ride problem. Most of the algorithms developed for the multi-vehicle dial-a-ride problems can

be used to solve single vehicle dial-a-ride problems. Though simpler than the multi-vehicle dial-a-ride problem, it is still NP-hard, and only small problem instances can be solved to optimality. In the following sections, exact, heuristic, and meta-heuristic approaches for the problem are reviewed.

### 2.1.1 Exact Solution Approaches

One of the early studies on exact solution approaches for the dial-a-ride problem is that of Psaraftis (1980). He proposes a dynamic programming approach for the dial-a-ride problem in which the objective minimizes the weighted combination of total service time and customer dissatisfaction (waiting time and ride time). He does not consider time windows for the customers in this paper. Customers make immediate requests, meaning they would like to receive the dial-a-ride service as soon as possible. As with other exact approaches to dial-a-ride problem, this one is unable to solve problems of a practical size. The complexity of the algorithm is $O(n^2 3^n)$, where $n$ is the number of requests. An instance with 9 requests is the largest instance solved in this paper. Subsequently, Psaraftis (1983b) modifies the algorithm to solve a dial-a-ride problem with hard time windows for the customers. The complexity of the modified algorithm remains the same, limiting its use to small problems.

The study by Desrosiers et al. (1986) reformulates the single vehicle dial-a-ride problem as an integer programming model. They propose a forward dynamic programming algorithm to solve a single vehicle dial-a-ride problem with time windows and vehicle capacities, and with an objective function that minimizes total distance. The algorithm uses a two-dimensional labeling scheme (time and cost of the partial solution for each state). It also eliminates infeasible and dominated states. An instance with 40 requests is the largest solved in this paper.

### 2.1.2 Heuristic and Meta-Heuristic Solution Approaches

The study by Häme (2011) proposes an adaptive insertion algorithm for a single vehicle dial-a-ride problem in which the objective function minimizes the linear combination of route duration and total customer ride times and waiting times. Hard time windows are employed in this model. He proposes an advanced insertion heuristic algorithm that can find optimal solutions with the assumption that the vehicle starts at time 0. This exact approach is only practical for solving small problems since it relies on a complete enumeration; therefore, he proposes a heuristic extension of the algorithm. Tests on randomly generated data show that it produces locally optimal solutions in a reasonable amount of time.

In practice, the number of requests received for a dial-a-ride service is not small enough to be satisfied by a single vehicle. Therefore, the single vehicle dial-a-ride problem has received less attention than the multi-vehicle dial-a-ride problem in the literature. One solution approach for the multi-vehicle dial-a-ride problem is a cluster-first, route-second strategy. In this method, all the customers are clustered in several groups, and each cluster is solved separately as a single vehicle dial-a-ride problem. Representative papers are reviewed below.

The studies by Sexton and Bodin (1985a,b) propose a heuristic approach to solve a single vehicle dial-a-ride problem with one-sided time windows in which the objective function minimizes excess ride time and time window violation for each customer. Sexton and Bodin (1985a) present a mathematical formulation of the problem and describe the Benders' decomposition application for this problem through alternative routing and scheduling. The paper proposes an optimal algorithm for the scheduling subproblem. Sexton and Bodin (1985b) focus on the routing subproblem. They design a space-time heuristic algorithm to find the initial route and an improvement algorithm to improve the sequences in the route.

The paper by Baugh et al. (1998) proposes a space-time nearest neighbor heuristic to

solve the single vehicle problem as a part of a cluster-first, route-second strategy that uses simulated annealing for clustering. The same space-time nearest neighbor algorithm is used by Jorgensen et al. (2007), but with a genetic algorithm for clustering. Both papers are reviewed in the following section.

## 2.2 Multi-Vehicle Dial-a-Ride Problem

The multi-vehicle dial-a-ride problem is studied more extensively than the single vehicle dial-a-ride problem. The reason is that the number of requests received by transit agencies is not small enough to serve all of them with just one vehicle. For example, the Winston Salem Authority (WSTA) had a daily ridership of more than 300 customers decades ago, as reported by Baugh et al. (1998), and larger metropolitan areas have significantly more.

In this section, exact, heuristic, and meta-heuristic solution approaches for the multi-vehicle dial-a-ride problem are reviewed.

### 2.2.1 Exact Solution Approaches

Few exact approaches have been developed for the multi-vehicle dial-a-ride problem, given its intractability and the typical size of practical problems. Where they have, only a single objective function and basic constraints are considered. Representative approaches based on branch-and-cut, branch-and-price, and branch-price-and-cut algorithms are reviewed below.

Cordeau (2006) proposes a branch-and-cut algorithm for a multi-vehicle dial-a-ride problem with an objective function that minimizes the total travel time. The problem is formulated as a mixed integer linear programming model, and hard time windows are used. The algorithm uses new valid inequalities for the dial-a-ride problem as well as known valid

inequalities for the traveling salesman, vehicle routing, and pick-up and delivery problems. At first, the algorithm solves the LP relaxation of the problem; if the solution is integer, the optimal solution is obtained and the algorithm terminates. However, if the solution is not integer, in the branching process it uses a separation heuristic to introduce the violated constraints. By conducting experiments, the author claims the algorithm can reduce run time and the number of nodes explored in the tree. Although the algorithm cannot be used to solve problems of practical sizes, it is nevertheless valuable for evaluating solutions obtained by heuristic and meta-heuristic methods, and to optimize small individual routes in dial-a-ride problems.

Liu et al. (2015) also propose a branch-and-cut algorithm for a multi-vehicle dial-a-ride problem with an objective function that minimizes the total travel time. Two mixed integer programming models are introduced for the problem, and hard time windows are again used. To solve the problem using a branch-and-cut algorithm, eight families of valid inequality are defined. Experimental results show that the valid inequalities can improve the lower bound in the LP relaxation of the model, and the branch-and-cut algorithm can solve instances of up to 22 requests.

### 2.2.2 Heuristic and Meta-Heuristic Solution Approaches

For solving problems of a larger size, both heuristic and meta-heuristic approaches have been developed.

One of the early heuristic approaches for the multi-vehicle dial-a-ride problem is that of Jaw et al. (1986). The authors present a greedy insertion heuristic with an objective of minimizing ride times, time window violations, and operating costs (consumption of available vehicle resources). The algorithm sorts customers by their pick-up times, and then assigns them to separate vehicles, one by one, until all the vehicles have just one

assigned customer. The remaining customers are assigned to vehicles, in order, using a cheapest insertion criterion. There is a possibility that one or more customers may not be assigned to a vehicle, since there are hard constraints that cannot be violated. Such customers are therefore rejected. The algorithm is simple and fast, and serves as a basis for other insertion algorithms that have been developed to solve single and multi-vehicle dial-a-ride problems.

Meta-heuristic approaches that have been developed for the dial-a-ride problem include tabu search, simulated annealing, and genetic algorithms.

Cordeau and Laporte (2003) present an approach using tabu search with an objective function that minimizes the routing cost. In their model the duration of each route and the maximum ride time of each customer are limited, and the time windows are hard. In the implementation of tabu search, the constraints are relaxed and intermediate infeasible solutions are allowed. However, the violations are minimized by use of a penalized objective function. The authors use a continuous diversification strategy to avoid local optima, and intra-route exchanges are performed every $k$ iterations to optimize the routes. The algorithm is tested on randomly generated datasets as well as six real datasets from the Danish transporter. Compared with exact solution approaches, the algorithm can solve large dial-a-ride problems within a reasonable amount of time.

The study reported by Baugh et al. (1998) uses a cluster-first, route-second approach to solve a multi-vehicle dial-a-ride problem with an objective function that minimizes the total number of vehicles used, total distance traveled by vehicles, and inconvenience to the customers (total time window violation). In their model, soft time windows are used. Simulated annealing is used for clustering, and a space-time nearest neighbor heuristic is used for routing. The authors test their model on a real dataset from the Winston Salem Transit Authority with more than 300 requests. Experimental results show that the model can produce near optimal solutions in a reasonable time.

Jorgensen et al. (2007) use a cluster-first, route-second approach to solve a multi-vehicle dial-a-ride problem with an objective function that minimizes total transportation cost and inconvenience to the customers (total excess ride time, total waiting time in the vehicles, and total time window violation). In their model, soft time windows are used. A genetic algorithm is used for clustering and the space-time nearest neighbor heuristic developed by Baugh et al. (1998) is used for routing.

Another study by Cubillos et al. (2009) also uses a cluster-first, route-second approach. The authors develop a genetic algorithm different from the genetic algorithm presented by Jorgensen et al. (2007) to cluster the customers and an insertion heuristic to construct the routes. Cubillos et al. (2009) compare their results with the results of Jorgensen et al. (2007) and Cordeau and Laporte (2003), and claim their approach is faster.

CHAPTER

# 3

# MIXED INTEGER LINEAR PROGRAMMING MODEL

## 3.1 Introduction

The dial-a-ride problem is NP-hard; therefore, medium and large instances of the problem cannot be solved by exact approaches. However, exact approaches *can* be used to evaluate the quality of solutions obtained by heuristic algorithms or meta-heuristic algorithms on small problem instances. Also, exact approaches may be useful once good clusters have been found using cluster-first route-second algorithms to optimize individual routes.

The dial-a-ride problem can be formulated as a mixed integer linear programming model, with several approaches reported in the literature Cordeau and Laporte (2007), Paquette et al. (2013), Cordeau (2006), Jorgensen et al. (2007). In most cases, however, the formulations are used to characterize the precise nature of the problem being addressed through other means, e.g., by heuristic or meta-heuristic algorithms, instead of directly providing a way to solve problems. In cases where they are used to solve actual problem instances, the mathematical programming formulations produce routes that sometimes appear to be unrealistic.

Paquette et al. Paquette et al. (2013), and Cordeau Cordeau (2006) formulate mixed integer programming models in which the objective function minimizes total travel time. In their formulation, there are constraints on hard time windows, maximum route duration, and maximum ride time for each customer. One way in which the formulations are impractical is in their use of hard time windows. Although customers would prefer service which does not violate them, in some cases no solution can be found, even when routes exist that have very small or negligible time window violations. Such solutions, if they can be found, serve to provide a measure of the quality of the solution when used within cluster-first route-second algorithms, i.e., for determining how good an assignment of customers to a vehicle really is. Therefore, it is often more practical to use soft time windows, and to have two objective functions, one that minimizes total travel time (as a surrogate for service cost), and another that minimizes total time window violations.

Also impractical is the way in which ride times are handled. For all requests, a common and fixed upper limit is placed on ride times, even though the direct travel time from pick-up location to drop-off location varies considerably. Because customers generally prefer more direct, taxi-like service from point $A$ to point $B$, it is therefore desirable to minimize the travel time in excess of that to make the routes more attractive.

Cordeau Cordeau (2006) proposes a branch and cut algorithm to solve this mathematical

model. He introduces some valid inequalities to make the algorithm more efficient. Valid inequalities help reduce the optimality gap; however, the approach cannot be used to solve medium size instances. Paquette et al.Paquette et al. (2013), on the other hand, develop a tabu search heuristic to solve the problem, but they do not report any results from the mathematical programming model.

Jorgensen et al. Jorgensen et al. (2007) also present a mixed integer programming model for the dial-a-ride problem. They define a multi-criteria objective function and use the weighted sum method to produce a single objective function. There are seven terms in the objective function, which minimizes total travel time, total excess ride time, total waiting time for customers in the vehicle, total work time, total time window violation, maximum excess ride time, and excess work time. An advantage of this model is that it considers several objectives; however, it has some disadvantages. The first downside with this model is that it puts the same weight on all different types of time window violation. Also, it considers the same maximum travel time for all the routes. The authors do not report any results for the mathematical model. A cluster-first route-second approach using a genetic algorithm and space-time nearest neighbor heuristic algorithm is used to solve the problem.

In the following section, a mixed integer programming model for the dial-a-ride problem is proposed. The limitations associated with other formulations are addressed so that more practical solutions are produced.

## 3.2 The Proposed MILP Model

Let $G = (V, A)$ be a directed graph. The vertex set $V$ is defined as $\{\{0, 2n+1\}, P, D\}$ where 0 and $2n+1$ both refer to the depot, $P = \{1, 2, ..., n\}$ is the set of pick-up (origin) vertices, and $D = \{n+1, n+2, ..., 2n\}$ is the set of drop-off (destination) vertices. Requests are a set of $(i, n+i)$ pairs where $i \in P$ and $n+i \in D$. There are two types of requests: outbound and

inbound. Outbound requests are those who are going to a doctor appointment or hospital. So, it is very important to arrive to their destination within the time window. On the other hand, inbound requests are those who are going back home from a doctor appointment or hospital. The arc set in the graph $G$ is defined as $A = \{(i, j) : i = 0, j \in P$, or $i, j \in P \cup D, i \neq j$ and $i \neq n + j$, or $i \in D, j = 2n + 1\}$.

To formulate the problem as a mixed integer programming model, the following notation is used to define various sets, parameters, and variables.

**Sets**

$V$: Set of all vertices, with $|V| = 2n + 2$

$P$: Set of origin vertices, with $|P| = n$

$D$: Set of destination vertices, with $|D| = n$

$D'$: Set of destination vertices for outbound requests

$K$: Set of vehicles

**Parameters**

$t_{i,j}$ : The time of traversing arc $(i, j)$

$[e_i, l_i]$ : The time window for vertex $i$

$q_i$ : Load at vertex $i$

$d_i$ : The service time at vertex $i$

$Q_k$ : The capacity of vehicle $k$

$w_1$ : The weight on total traveling time

$w_2$ : The weight on total excess ride time

$w_3$ : The weight on total waiting time

$w_4$ : The weight on total lateness in all vertices except drop-off vertices for outbound requests

$w_5$ : The weight on total lateness in drop-off vertices for outbound requests

$w_6$ : The weight on total earliness for drop-off vertices

**Variables**

$x_{i,j,k}$ : Is equal to 1 if arc $(i, j)$ is traversed by vehicle $k$; otherwise, it is equal to 0

$u_{i,k}$ : The time at which the vehicle $k$ starts servicing at vertex $i$

$w_{i,k}$ : The load of vehicle $k$ after leaving vertex $i$

$a_{i,k}$ : The time at which the vehicle $k$ arrives to vertex $i$

Then, the dial-a-ride problem can be formulated as follows:

$$\begin{aligned}
\min \quad & w_1 \sum_{k\in K}\sum_{i\in V}\sum_{j\in V} t_{i,j}x_{i,j,k} + w_2 \sum_{k\in K}\sum_{i\in P}((u_{i+n,k}-u_{i,k})-t_{i,i+n}) \\
& + w_3 \sum_{k\in K}\sum_{i\in P\cup D}(u_{i,k}-a_{i,k}) + w_4 \sum_{k\in K}\sum_{i\in V\setminus D'}\max\{0,(u_{i,k}-l_i)\} \\
& + w_5 \sum_{k\in K}\sum_{i\in D'}\max\{0,(u_{i,k}-l_i)\} + w_6 \sum_{k\in K}\sum_{i\in D}\max\{0,(e_i-u_{i,k})\} \qquad (3.1)
\end{aligned}$$

s.t.

$$\sum_{k\in K}\sum_{j\in V} x_{i,j,k} = 1 \qquad \forall i \in P, \qquad (3.2)$$

$$\sum_{i\in P\cup\{2n+1\}} x_{0,i,k} = \sum_{i\in D\cup\{0\}} x_{i,2n+1,k} = 1 \qquad \forall k \in K, \qquad (3.3)$$

$$\sum_{j\in V} x_{i,j,k} - \sum_{j\in V} x_{n+i,j,k} = 0 \qquad \forall i \in P, \forall k \in K, \qquad (3.4)$$

$$\sum_{j\in V} x_{j,i,k} - \sum_{j\in V} x_{i,j,k} = 0 \qquad \forall i \in P\cup D, \forall k \in K, \qquad (3.5)$$

$$u_{j,k} \geq (u_{i,k}+d_i+t_{i,j})x_{i,j,k} \qquad \forall i,j \in V, \forall k \in K, \qquad (3.6)$$

$$a_{j,k} = \sum_{i\in V}(u_{i,k}+d_i+t_{i,j})x_{i,j,k} \qquad \forall j \in P\cup D, \forall k \in K, \qquad (3.7)$$

$$w_{j,k} \geq (w_{i,k}+q_j)x_{i,j,k} \qquad \forall i,j \in V, \forall k \in K, \qquad (3.8)$$

$$u_{i,k} \geq e_i \qquad \forall i \in P, \forall k \in K, \qquad (3.9)$$

$$u_{i+n,k} - u_{i,k} \geq t_{i,i+n} \qquad \forall i \in P, \forall k \in K, \qquad (3.10)$$

$$\max\{0,q_j\} \leq w_{i,k} \leq \min\{Q_k, Q_k+q_i\} \qquad \forall i \in V, \forall k \in K, \qquad (3.11)$$

$$x_{i,j,k} = 0 \; or \; 1 \qquad \forall i \in V, \forall k \in K. \qquad (3.12)$$

In this formulation, the objective function (3.1) is a multi objective function. It consists of the objectives of minimizing the total transportation cost, and the customer inconvenience.

The total transportation cost is estimated to be proportional to the total time used when transporting the customers by all the vehicles. Customer inconvenience is estimated to be proportional to the total excess ride time for the customers and the total waiting time for the customers in the vehicles and total lateness for the customers in the vehicles.

In order to handle this multi-criteria objective function, each part of the objective function is multiplied by a weight. These weights are denoted $w_1, w_2, ..., w_6$. The values of the weights are then used to decide the relative weight of each criteria in the overall problem.

The constraints (3.2) describe the requirement that each vertex should be served once and by just one vehicle. Constraints (3.3) make sure that each vehicle should start and end the route at the depot. Constraints (3.4) indicate that both the pick-up vertex and drop-off vertex of a request should be served by just one vehicle. Constraints (3.5) make sure that if the vehicle enters a vertex other than the depot, it should leave that vertex. Constraints (3.6) and (3.7) and (3.8) indicate that if the arc $(i, j)$ is traversed by vehicle $k$, the start of service at vertex $j$ cannot be before the finish of service at vertex $i$ plus the travel time between these two vertices, the arrival time to a vertex $j$ would be equal to the finish time at vertex $i$ plus the traveling time between these two vertices, and the load of the vehicle after leaving vertex $j$ should be consistent with the load at vertex $j$, respectively. Constraints (3.9) make sure that customers cannot be picked up before the customers' early pick-up time. Constraints (3.10) make sure that for each request, service at a drop-off vertex cannot start before the finish of service at the pick-up vertex plus the direct travel time between these two vertices. Constraints (3.11) indicate that the load of vehicle after leaving a vertex should be more than the load at that vertex, and it cannot exceed the capacity of vehicle. Finally, Constraints (3.12) address the binary variable $x$.

### 3.2.1 Linearization

The mathematical model described above is an integer programming model; however, it is not a mixed integer linear programming model because there are nonlinear terms in the objective function and constraints. In this section, all nonlinear terms are converted to linear ones. Therefore, a mixed integer linear programming model is obtained.

The fourth and fifth terms in the objective function are nonlinear. In order to make these two terms linear, an auxiliary variable described as follows is added to the model:

$s_{ik}$ : lateness at vertex $i$ by vehicle $k$

The fourth and fifth terms in the objective function now become:

$$w_4 \sum_{k \in K} \sum_{i \in V \setminus D'} S_{i,k} + w_5 \sum_{k \in K} \sum_{D'} S_{i,k}$$

together with the following constraints:

$$u_{i,k} - s_{i,k} \leq l_i \qquad \forall i \in V, \forall k \in K$$

The sixth term in the objective function is not linear. To make this term linear, another auxiliary variable is added:

$v_{ik}$ : earliness at vertex $i$ by vehicle $k$

Therefore, the sixth term in the objective function now becomes:

$$w6 \sum_{k \in K} \sum_{i \in D} v_{i,k}$$

together with the following constraints:

$$u_{i,k} + v_{i,k} \geq e_i \qquad \forall i \in D, \forall k \in K$$

Constraint 3.6 is not linear. An equivalent linear constraint is the following:

$$(u_{i,k} + d_i + t_{i,j} - u_{j,k}) \leq M_1(1 - x_{i,j,k}) \qquad \forall i, j \in V, \forall k \in K$$

Constraint 3.7 is not linear. Two linear constraints equivalent to this one are the following:

$$a_{i,j,k} \leq M_2 x_{i,j,k} \qquad \forall i \in V, \forall j \in P \cup D, \forall k \in K$$
$$a_{i,j,k} \leq u_{i,k} + d_i + t_{i,j} \qquad \forall i \in V, \forall j \in P \cup D, \forall k \in K$$

Constraint 3.8 is not linear. An equivalent linear constraint is the following:

$$(w_{i,k} + q_j - w_{j,k}) \leq M_3(1 - x_{i,j,k}) \qquad \forall i, j \in V, \forall k \in K$$

Constraint 3.11 is not linear. four linear constraints equivalent to this one are the following:

$$w_{i,k} \geq 0 \qquad \forall i \in V, \forall k \in K$$
$$w_{i,k} \geq q_i \qquad \forall i \in V, \forall k \in K$$
$$w_{i,k} \leq Q_k + q_i \qquad \forall i \in V, \forall k \in K$$
$$w_{i,k} \leq Q_k \qquad \forall i \in V, \forall k \in K$$

In these constraints, $M_1$, $M_2$, and $M_3$ are arbitrarily large numbers. Now, the problem can be reformulated as the following mixed integer linear programming problem:

$$\min \quad w_1 \sum_{k\in K}\sum_{i\in V}\sum_{j\in V} t_{i,j}\,x_{i,j,k} + w_2 \sum_{k\in K}\sum_{i\in P}((u_{i+n,k}-u_{i,k})-t_{i,i+n})$$
$$+ w_3 \sum_{k\in K}\sum_{i\in P\cup D}(u_{i,k}-a_{i,k}) + w_4 \sum_{k\in K}\sum_{i\in V\setminus D'} s_{i,k}$$
$$+ w_5 \sum_{k\in K}\sum_{i\in D'} s_{i,k} + w_6 \sum_{k\in K}\sum_{i\in D} v_{i,k} \tag{3.13}$$

s.t.
$$\sum_{k\in K}\sum_{j\in V} x_{i,j,k} = 1 \qquad \forall i \in P, \tag{3.14}$$
$$\sum_{i\in P\cup\{2n+1\}} x_{0,i,k} = \sum_{i\in D\cup\{0\}} x_{i,2n+1,k} = 1 \qquad \forall k \in K, \tag{3.15}$$
$$\sum_{j\in V} x_{i,j,k} - \sum_{j\in V} x_{n+i,j,k} = 0 \qquad \forall i \in P, \forall k \in K, \tag{3.16}$$
$$\sum_{j\in V} x_{j,i,k} - \sum_{j\in V} x_{i,j,k} = 0 \qquad \forall i \in P\cup D, \forall k \in K, \tag{3.17}$$
$$(u_{i,k} + t_{i,j} - u_{j,k}) \le M_1(1-x_{i,j,k}) \qquad \forall i,j \in V, \forall k \in K, \tag{3.18}$$
$$a_{i,j,k} \le M_2 x_{i,j,k} \qquad \forall i \in V, \forall j \in P\cup D, \forall k \in K, \tag{3.19}$$
$$a_{i,j,k} \le b_{i,k} + t_{i,j} \qquad \forall i \in V, \forall j \in P\cup D, \forall k \in K, \tag{3.20}$$
$$(w_{i,k} + q_j - w_{j,k}) \le M_3(1-x_{i,j,k}) \qquad \forall i,j \in V, \forall k \in K, \tag{3.21}$$
$$u_{i,k} \ge e_i \qquad \forall i \in P, \forall k \in K, \tag{3.22}$$
$$u_{i+n,k} - u_{i,k} \ge t_{i,i+n} \qquad \forall i \in P, \forall k \in K, \tag{3.23}$$
$$u_{i,k} - s_{i,k} \le l_i \qquad \forall i \in V, \forall k \in K, \tag{3.24}$$
$$u_{i,k} + v_{i,k} \ge e_i \qquad \forall i \in D, \forall k \in K, \tag{3.25}$$
$$w_{i,k} \ge 0 \qquad \forall i \in V, \forall k \in K, \tag{3.26}$$
$$w_{i,k} \ge q_i \qquad \forall i \in V, \forall k \in K, \tag{3.27}$$
$$w_{i,k} \le Q_k + q_i \qquad \forall i \in V, \forall k \in K, \tag{3.28}$$
$$w_{i,k} \le Q_k \qquad \forall i \in V, \forall k \in K, \tag{3.29}$$
$$x_{i,j,k} = 0 \; or \; 1 \qquad \forall i \in V, \forall k \in K. \tag{3.30}$$

CHAPTER

4

# HEURISTIC APPROACHES FOR THE SINGLE VEHICLE DIAL-A-RIDE PROBLEM

## 4.1 Introduction

The single vehicle dial-a-ride problem is an important subproblem of multi-vehicle dial-a-ride problem. The goal of the single vehicle dial-a-ride problem is to determine an optimal route for a set of customers in respect to a specific objective function. This problem is

still an NP-hard problem and practical size instances of this problem cannot be solved to optimality using exact approaches. Different exact approaches are designed and used for this problem (for example see Psaraftis (1983b); Desrosiers et al. (1986)). However, these approaches can be used only for small size instances of single vehicle dial-a-ride problem. Heuristic or meta-heuristic approaches are more popular and practical for medium size and large size instances of this problem (for example see Baugh et al. (1998); Häme (2011); Psaraftis (1983a)).

In this chapter, two heuristic algorithms, a space-time nearest neighbor heuristic algorithm and an insertion heuristic algorithm, are described. These two algorithms are fast and they can produce good quality solutions for the single vehicle dial-a-ride problem in a reasonable amount of time.

The space-time nearest neighbor heuristic algorithm and the insertion heuristic algorithm are modified in a way to incorporate the new definition of the dial-a-ride problem (new service policies and new objectives). Also, improvements have been made on these two algorithms that will be discussed carefully in the following sections. Finally, these improved algorithms are combined with a route improvement heuristic algorithm (two-opt).

## 4.2 Space Time Nearest Neighbor Heuristic

A space-time nearest neighbor (STNN) heuristic algorithm was first introduced by Baugh et al. (1998) for solving the single vehicle dial-a-ride problem. The STNN heuristic algorithm is a constructive heuristic in the sense that it starts with an empty solution and then, at each iteration, extends the solution until a complete route is obtained.

The space-time nearest neighbor heuristic algorithm is a fast algorithm for generating routes and scheduling pick-ups and drop-offs in the dial-a-ride problem. The algorithm generates good solutions, and has been used in other case studies, e.g., by Jorgensen et al.

(2007). Baugh et al. (1998) provide pseudo code for the algorithm and Jorgensen et al. (2007) give a brief description of it, but neither account offers a clear picture of how the algorithm works.

The original space-time nearest neighbor (STNN) heuristic algorithm introduced by Baugh et al. (1998) and used by Jorgensen et al. (2007) is described below in detail. A flow chart of the algorithm is given in Figure 4.1. The steps in the flow chart are described below.

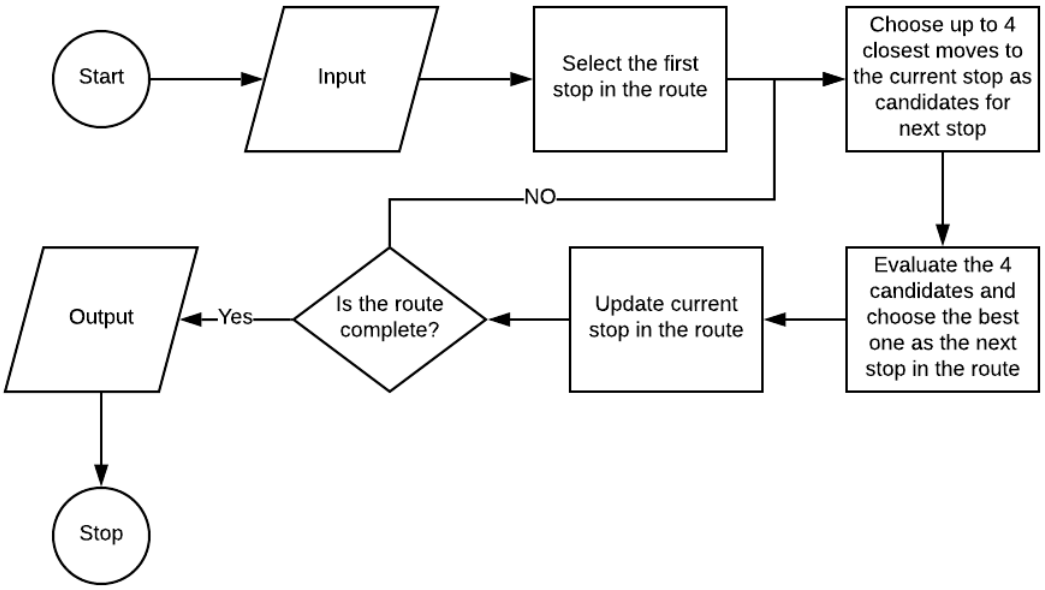


Figure 4.1: A flow chart describing space-time nearest neighbor heuristic algorithm.

**Step 0**

The algorithm starts with an empty route. That means the vehicle is empty and none of the origin and destination locations appear in the route.

**Step 1**

Input for the algorithm:

- The location for depot
- The origin and destination locations for all requests
- The pick-up and drop-off time windows for all requests
- The capacity of the vehicle

**Step 2**

The first stop in the route is selected as follows. The origin of the request with the smallest EPT is added as the first stop in the route. If there are multiple such requests, one of them is chosen randomly. The vehicle is scheduled to leave the depot so that it arrives at the EPT of the first stop.

**Step 3**

To find the closest moves from the current stop, the procedure 1 is used. In this procedure, $a$ is the current stop, and $t$ is the departure time from the current stop, and *exclude* is a list containing the stops cannot be added to the subroute.

The *Closest* procedure checks all possible moves from the current stop, and finds the one with minimum move cost.

Possible moves from the current stop include all the origin and destination locations not added to the route in previous steps. However, infeasible moves are not considered:

- The destination location for a request cannot be considered as a possible move if that request has not been picked up yet (the origin location of that request has not been added to the route).

**Algorithm 1** The closest function.

```
1:  procedure CLOSEST(a, t, exclude)
2:      min cost = inf
3:      min node = None
4:      for c ∈ V do
5:          if c.destination ∉ exclude then
6:              new cost = Move_Cost(a, c.destination, t, W)
7:              if new cost < min cost then
8:                  min cost = new cost
9:                  min node = c.destination
10:     if |V| ≥ Cap then
11:         return min node
12:     for c ∉ V do
13:         if c.origin ∉ exclude then
14:             new cost = Move_cost(a, c.origin, t, W)
15:             if new cost < min cost then
16:                 min cost = new cost
17:                 min node = c.origin
18:     return min node
```

If the current stop is the destination location of a request, and all other requests have been visited (their origin and destination locations have been added to the route), the only possible next move is the depot.

Baugh et al. (1998) calculates the moving cost using the procedure 2. If the current stop is $a$ and a possible next stop is $b$, the moving cost is the weighted sum of the travel time from the current stop to the next stop and the time window violation at the next stop. So, $W = (w_1, w_2)$ in which $w_1$ and $w_2$ are weights for the travel time term and the time window violation term, respectively. Therefore, the output of *Move_Cost* procedure is as the following formula:

$$cost\ (a,b) = w_1 * \text{travel time from } \ a \ \text{ to } \ b + w_2 * \text{ time window violation at } \ b$$

**Algorithm 2** The move cost function.

```
1:  procedure MOVE_COST(a, b, t, W)
2:      t_time = distance(a, b)
3:      e_time = t + t_time
4:      if t > a.ltw then
5:          ltime = e_time
6:      else
7:          ltime = a.ltw + t_time
8:      TW_Viol = 0
9:      if ltime < b.etw then
10:         TW_Viol = b.etw − e_time
11:     if e_time > b.ltw then
12:         TW_Viol = e_time − b.etw
13:     return W ∗ (t_time, TW_Viol)
```

In the *Closest* procedure, Baugh et al. (1998) use the $W = (20, -1)$ weights for travel time and time window violation. The reason to put $w_2 = -1$ in the call to *Move_Cost* is to make late nodes appear relatively more attractive to visit.

**Step 4**

All four candidates selected in the previous step are evaluated and one of them is chosen as the next stop in the route. The flow chart in Figure 4.2 ,and procedure 3 describes this step. For each candidate, three subsequent moves are found, one after another, by looking for the next lowest cost move to produce a possible sub-route for the candidate. The candidate whose sub-route has the lowest cost is selected as the next stop.

In the *Subroute_Cost* procedure, Baugh et al. (1998) use the $W = (20, 1)$ weights for travel time and time window violation.

**Step 5**

The current stop is updated to be the location chosen in step 5.

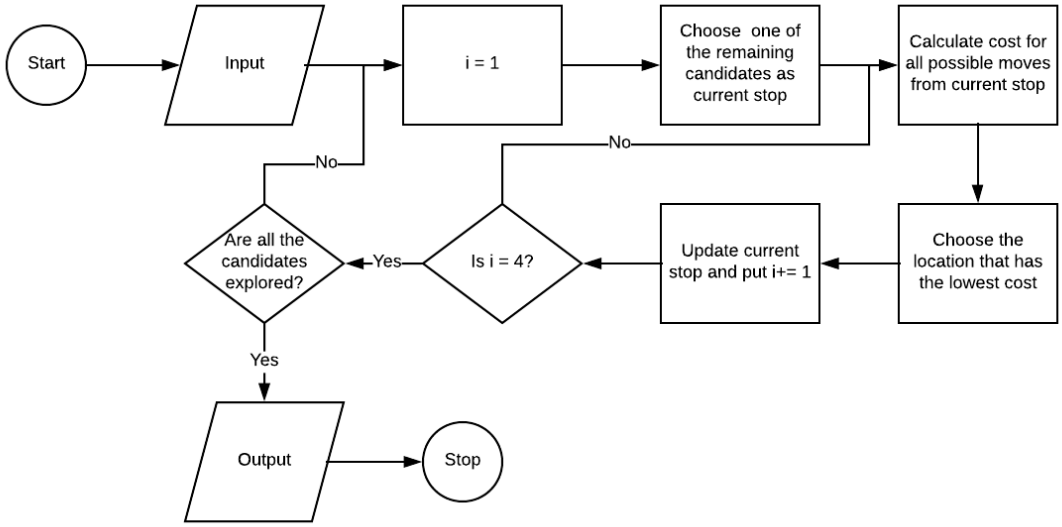


Figure 4.2: Evaluation step in space-time nearest neighbor heuristic flow chart

**Algorithm 3** The subroute cost .

```
1:  procedure SUBROUTE_COST(a, b, t)
2:      subroute = []
3:      tot_cost = 0
4:      for i ∈ {1,2,3,4} do
5:          tot_cost+ = Move_Cost(a, b, t, W)
6:          return W ∗ (t_time, TW_Viol)
7:          subroute ∪ b
8:          a = b
9:          t = departure time at b
10:         b = Closest(a, t)
11:         if b = None then
12:             break
13:     return tot_cost
```

**Step 6**

Check to see if all requests (both origin and destination locations) have been added to the route or not. If they have, go to step 8. Otherwise return to step 3.

**Step 7**

Output the route generated with pick-up and drop-off times for the set of requests.

### 4.2.1 Search Space of the STNN Algorithm

Assume the total number of customers for a single vehicle dial-a-ride problem is $n$. The maximum number of subroutes and routes explored by the space-time nearest neighbor heuristic algorithm is given by $S(n)$:

$$S(n) = 1 \quad n = 1$$

$$S(n) = 4(2n-1) \quad n > 1$$

When $n = 1$, it is obvious that $S(n) = 1$. When $n > 1$, the algorithm explores up to 4 subroutes in each iteration (steps 3 and 4 in the flowchart of the algorithm). The number of available stops (nodes) are $2n$ (2 nodes are associated with each customer, one for the origin and one for the depot). In each iteration of the algorithm, just one stop is added to the route, so it takes exactly $2n-1$ iterations to terminate the algorithm. It is not $2n$ iterations, because the first stop is always the origin location of the customer with smallest EPT.

This function shows that, for example, if the total number of customers is 10, the STNN heuristic algorithm explores at most 76 subroutes and routes.

## 4.3 Insertion Heuristic

Similar to the STNN heuristic algorithm, the insertion (INS) heuristic algorithm is also a constructive heuristic. Generally, the insertion heuristic algorithm starts with a subtour, and it extends this tour by inserting the remaining nodes one after the other until all nodes have been inserted.

The insertion heuristic algorithm is widely used for different types of the traveling salesman problem and vehicle routing problem, and it can produce good quality results with a reasonable running time for the problems (for example Gendreau et al. (1992, 1998); Salhi and Nagy (1999); Campbell and Savelsbergh (2004)). Many studies also used the insertion heuristic algorithm for dial-a-ride problem (see for example Jaw et al. (1986); Madsen et al. (1995); Diana and Dessouky (2004); Wong and Bell (2006); Häme (2011)).

The insertion heuristic algorithm can be implemented in several different ways. The following features affect the implementation of this algorithm:

- How to generate the initial subtour
- How to choose the next node to be inserted
- Where to insert the next node

The second and third features cause the major distinctions in the different types of the insertion heuristic algorithm. The different types of the insertion heuristic algorithm are described below.

**Nearest Insertion**: The nearest insertion algorithm repeatedly chooses the un-routed vertex with the minimum cost to its nearest neighbor among the routed vertices, and inserts it in between the two consecutive routed vertices for which such an insertion causes the minimum increase in total cost.

**Farthest Insertion**: The farthest insertion algorithm repeatedly chooses the un-routed vertex with the maximum cost to its nearest neighbor among the routed vertices, and inserts it in between the two consecutive routed vertices for which such an insertion causes the minimum increase in total cost.

**Cheapest Insertion**: The cheapest insertion algorithm repeatedly chooses the un-routed vertex whose insertion causes the minimum increase in total cost, and inserts it in between the two consecutive routed vertices for which such an insertion causes the minimum increase in total cost.

**Random Insertion**: The random insertion algorithm repeatedly chooses a random un-routed vertex, and inserts it in between the two consecutive routed vertices for which such an insertion causes the minimum increase in total cost.

In this section, a cheapest insertion heuristic is developed in the manner of Häme (2011), who introduces an algorithm for problems with hard and narrow time windows. The following definitions are his:

**Definition**: A service sequence is an ordered list of origin vertices and destination vertices. For instance, if the origin vertex of customer $i$ is denoted by $i^{\uparrow}$ and the destination vertex of the same customer denoted by $i^{\downarrow}$, the service sequence $(i^{\uparrow}, j^{\uparrow}, j^{\downarrow}, i^{\downarrow})$ indicates the customers $i$ and $j$ are picked-up, and then the customers $j$ and $i$ are dropped-off, respectively.

**Definition**: Assume $s = (p_1, p_2, ..., p_m)$ is a service sequence, where $p_j \in \{1, 2, ..., 2N\} \forall\, j \in \{1, ..., m\}$. Consider two natural numbers $h$ and $k$ such that $h \in \{1, 2, ..., m+1\}$ and $k \in \{h+1, ..., m+2\}$. The insertion function $I(s, i, h, k)$ that inserts the $i$th customer into the service sequence $s$ is defined as the following service sequence:

$$I(s, i, h, k) = (r_1, ..., r_{m+2}) = (p-1, p_2, ..., p_{h-1}, i^{\uparrow}, p_h, ..., p_{k-2}, i^{\downarrow}, p_{k-1}, ..., p_m)$$

The above definitions help in understanding the advanced insertion algorithm, which is presented below as Algorithm 4. At first, the set of feasible service sequences $S_i$ is determined recursively for each customer $i \in \{1,...,N\}$ by inserting the origin and destination vertices of customer $i$ into each feasible service sequence with respect to customers $1,...,i-1$. In this way the algorithm produces the set $S_N$ of all feasible routes with respect to customers $1,...,N$. Then the solution to the problem is obtained by choosing the sequence $s \in S_N$ with minimal cost $C(s)$. In Häme (2011), the cost function involves a weighted sum of route duration, total waiting time, and total riding time of customers.

**Algorithm 4** The advanced insertion algorithms.

```
1:  procedure INSERTION(N)
2:      S_0 = {∅}
3:      for i ∈ {1,2,...,N} do
4:          S_i = ∅
5:          for s ∈ S_{i-1} do
6:              for h ∈ {1,...|s|+1} and k ∈ {h+1,...,|s|+2} do
7:                  r = I(s,i,h,k)
8:                  if r is feasible then
9:                      S_i = S_i ∪ {r}
10:     min cost = ∞
11:     min route = ∅
12:     for s ∈ S_N do
13:         Calculate cost C(s)
14:         if C(s) < min cost then
15:             min cost = C(s)
16:             min route = s
17:     return min cost, min route
```

In the advanced insertion algorithm described in Algorithm 4, the precedence constraints are satisfied automatically; therefore, the solution obtained by this algorithm is always feasible if soft time windows are considered. Moreover, if a fixed start time is assumed, this algorithm finds the optimal solution.

The complexity of the advanced insertion algorithm described in Algorithm 4 grows exponentially with the size of the problem. Therefore, this algorithm is unable to produce exact solutions for medium or large size instances of the dial-a-ride problem. Importantly, solutions produced by the algorithm are optimal only if a fixed departure time from the depot is assumed.

In Algorithm 4, $S_i$ is a set that contains all possible service sequences for each customer $i \in \{1,2,...,N\}$, and the algorithm generates all possible service sequences for all the customers. At the beginning, the algorithm starts with $S_0 = \{\}$. Then, for $i = 1$ there is only one feasible service sequence that is $S_1 = \{(1^{\uparrow}, 1^{\downarrow})\}$. For $i = 2$, there are six feasible service sequences $S_2 = \{(1^{\uparrow}, 1^{\downarrow}, 2^{\uparrow}, 2^{\downarrow}), (2^{\uparrow}, 2^{\downarrow}, 1^{\uparrow}, 1^{\downarrow}), (1^{\uparrow}, , 2^{\uparrow}1^{\downarrow}, 2^{\downarrow}), (1^{\uparrow}, , 2^{\uparrow}2^{\downarrow}, 1^{\downarrow}), (2^{\uparrow}, , 1^{\uparrow}1^{\downarrow}, 2^{\downarrow}), (2^{\uparrow}, , 1^{\uparrow}2^{\downarrow}, 1^{\downarrow})\}$. For $i = 3$ there are 90 feasible service sequences. Therefore, it performs like a complete enumeration, and it is the reason for the impracticality of this algorithm for medium size or large size instances of the dial-a-ride problem. Instead of considering all feasible service polices, in each iteration the insertion with minimum cost is chosen. Consider $SP_i$ to represent the cheapest insertion for each customer $i$. With this definition, $SP_0 = ()$. For $i = 1$, there is only one feasible service policy and that would be the cheapest, so $SP_1 = (1^{\uparrow}, 1^{\downarrow})$. For $i = 2$, the different insertions of $2^{\uparrow}$ and $2^{\downarrow}$ into $SP_1$ are evaluated and the service sequence with the cheapest cost would be chosen. For example, if the service sequence $(1^{\uparrow}, 1^{\downarrow}, 2^{\uparrow}, 2^{\downarrow})$ has the cheapest cost, then $SP_2 = (1^{\uparrow}, 1^{\downarrow}, 2^{\uparrow}, 2^{\downarrow})$.

Generally, the cheapest feasible service sequences $SP_i$ is determined for each customer $i\{1,...,N\}$ by inserting the origin and destination vertices of customer $i$ into the cheapest feasible service sequence with respect to customers $1,...,i-1$ and the cheapest service sequence would be chosen. In this way, finally, the algorithm produces the cheapest service sequence $SP_N$ with respect to customers $1,...,N-1$. The solution for the problem would be $SP_N$. With this modification, the algorithm could be used to solve medium size and large size instances of the dial-a-ride problem with reasonable running time. The cheapest

insertion heuristic algorithm is described in Algorithm 5.

**Algorithm 5** The cheapest insertion heuristic algorithms.

```
1:  procedure INSERTION(N)
2:      SP = {0 : []}
3:      for i ∈ {1, 2, ..., N} do
4:          s = SP[i − 1]
5:          min cost = ∞, min route = ∅
6:          for h ∈ {1, ...|s| + 1} and k ∈ {h + 1, ..., |s| + 2} do
7:              r = I(s, i, h, k)
8:              if r is feasible then
9:                  c = Cost(r)
10:                 if c ≤ min cost then
11:                     min cost = c, min route = r
12:         SP[i] = min route
13:     return min cost, min route
```

### 4.3.1 Search Space of the Cheapest Insertion Algorithm

Assume the total number of customers for a single vehicle dial-a-ride problem is $n$. The maximum number of subroutes and routes explored by the insertion heuristic algorithm is given by $S(n)$:

$$\begin{aligned} S(n) &= 1 \quad n = 1 \\ S(n) &= n(2n-1) + s(n-1) \quad n > 1 \end{aligned} \tag{4.1}$$

Assume the vehicle has enough capacity. The calculation of the maximum number of subroutes and routes explored by the insertion heuristic algorithm is described below.

When $n = 1$, it is obvious that $S(1) = 1$. The subroute would be $(1^{\uparrow}, 1^{\downarrow})$. When $n = 2$, algorithm 5 first inserts the first customer in the empty route, then it inserts the second customer in the cheapest subroute with respect to the first customer (there is only one available subroute, that is the cheapest). Therefore, the number of soubroutes would be $S(1)$

plus the number of ways to insert the second customer in the $(1^{\uparrow}, 1^{\downarrow})$ subroute. There are $2*3=6$ possible ways to insert the second customer in $(1^{\uparrow}, 1^{\downarrow})$. In the subroute $(1^{\uparrow}, 1^{\downarrow})$, there are three possible spots to insert the origin and destination of the second customer (before $1^{\uparrow}$, between $1^{\uparrow}$ and $1^{\downarrow}$, and after $1^{\downarrow}$). If the origin of the second customer is inserted in the first spot (before $1^{\uparrow}$), there are 3 spots available to insert the destination of this customer (between $2^{\uparrow}$ and $1^{\uparrow}$, between $1^{\uparrow}$ and $1^{\downarrow}$, and after $1^{\downarrow}$). If the origin of the second customer is inserted in the second spot (between $1^{\uparrow}$ and $1^{\downarrow}$), there are 2 spots available to insert the destination of this customer (between $2^{\uparrow}$ and $1^{\downarrow}$, and after $1^{\downarrow}$). If the origin of the second customer is inserted in the third spot (after $1^{\downarrow}$), there is only 1 spot available to insert the destination of this customer (after $2^{\downarrow}$). Therefore, there are $3+2+1=6$ ways to insert the second customer in the $(1^{\uparrow}, 1^{\downarrow})$ subroute. These 6 ways are:

$(1^{\uparrow}, 1^{\downarrow}, 2^{\uparrow}, 2^{\downarrow}), (2^{\uparrow}, 2^{\downarrow}, 1^{\uparrow}, 1^{\downarrow}), (1^{\uparrow}, , 2^{\uparrow}1^{\downarrow}, 2^{\downarrow}), (1^{\uparrow}, , 2^{\uparrow}2^{\downarrow}, 1^{\downarrow}), (2^{\uparrow}, , 1^{\uparrow}1^{\downarrow}, 2^{\downarrow}), (2^{\uparrow}, , 1^{\uparrow}2^{\downarrow}, 1^{\downarrow})$

So, when $n=2$, the algorithm explores $6+1=7$ subroutes. The same idea can be used for larger instances. So, when there are $n>1$ customers available, the number of soubroutes would be $S(n-1)$ plus the number of ways to insert the $n^{th}$ customer in the cheapest subroute with respect to the $n-1^{th}$ customer. Using the previous idea, there are $2n-1$ spots available to insert the origin and destination of the $n^{th}$ customer in the cheapest subroute with respect to the $n-1^{th}$ customer. If the origin of the $n^{th}$ customer is inserted in the first spot, there are $2n-1$ spots available to insert the destination of this customer. If the origin of the customer is inserted in the second spot, there are $2n-2$ spots available to insert the destination of this customer. Finally, if the origin of the customer is inserted in the last spot, there are $2n-2n+1$ spots available to insert the destination of this customer. Therefore, the number of ways to insert the $n^{th}$ customer in the cheapest subroute with respect to the $n-1^{th}$ customer can be calculated as follows:

$$(2n-1)+(2n-2)+(2n-3)+\ldots+(2n-2n+1)=$$

$$(2n-1)(2n)-(1+2+\ldots+2n-1)=$$

$$(4n^2-2n)-((2n-1)2n/2)=$$

$$4n^2-2n-2n^2+n=$$

$$2n^2-n=$$

$$n(2n-1)$$

Therefore, when there are $n>1$ customers available, the number of explored routes and subroutes would be $S(n-1)$ plus $n(2n-1)$.

The closed form solution is as follows:

$$S(n)=(2/3)n^3+(1/2)n^2-(1/6)n \quad n\geq 1 \tag{4.2}$$

Induction is used to show that Eq. 4.2 is equivalent to Eq. 4.1. When $n=1$, the value of $S(1)$ is 1 for both 4.2 and 4.1. When $n=2$, the value of $S(2)$ is 7 in both 4.2 and 4.1. Assume 4.2 and 4.1 are equivalent for a general $n>1$. Therefore, the following equation holds:

$$S(n)=n(2n-1)+s(n-1)=(2/3)n^3+(1/2)n^2-(1/6)n \tag{4.3}$$

Now, it will be shown that it holds also for $n+1$. By 4.2, the following equation holds:

$$\begin{aligned} S(n+1)=&(2/3)(n+1)^3+(1/2)(n+1)^2-(1/6)(n+1)= \\ &(2/3)n^3+2n^2+2n+(2/3)+(1/2)n^2+n+(1/2)-(1/6)n-(1/6)= \\ &(2/3)n^3+(5/2)n^2+(17/6)n+1 \end{aligned} \tag{4.4}$$

By 4.1 and the assumption of the induction 4.3, the following equation holds:

$$\begin{aligned} S(n+1) =& (n+1)(2n+1) + S(n) = \\ & (n+1)(2n+1) + (2/3)n^3 + (1/2)n^2 - (1/6)n = \\ & 2n^2 + 3n + 1 + (2/3)n^3 + (1/2)n^2 - (1/6)n = \\ & (2/3)n^3 + (5/2)n^2 + (17/6)n + 1 \end{aligned} \tag{4.5}$$

It can be seen that Eq. 4.4 and Eq. 4.5 are equivalent. Therefore Eq. 4.2 and Eq. 4.1 are equivalent, and Eq. 4.2 is the closed form solution of Eq. 4.1.

This function shows that, for example, if the total number of customers is 10, the INS heuristic algorithm explores at most 715 subroutes and routes. However, the STNN heuristic algorithm explores at most 76 subroutes and routes for an instance with size of 10. Therefore, the search space of INS heuristic algorithm is larger than the space of the STNN heuristic algorthm.

## 4.4 Improvements to the Heuristic Algorithms

The purpose of this section is to describe modifications and improvements made to the space-time nearest neighbor heuristic algorithm and insertion heuristic algorithm described in previous sections. These two algorithms are modified to solve the single vehicle dial-a-ride problem with new definitions. The motivation for improving these algorithms is that both algorithms often work well, in that they are able to quickly generate routes with low costs. A close examination of the routes, however, reveals occasional defects that would seem to limit their usefulness in practice. Modifications and areas for improvement to the algorithms and their expression are described below.

### 4.4.1 Service Policies

#### Eliminating Drop-off Waiting Time

In some prior studies, if a vehicle arrives early to a customer's destination (earlier than the EDT), the customer stays in the vehicle until the EDT, then the vehicle drops off the customer and continues (see for example Baugh et al. (1998); Häme (2011)). Though such a policy may be adopted to make time window violations appear smaller in a routing algorithm, implementing it in practice is clearly unrealistic.

In actuality, the customer would depart the vehicle when it arrives at the destination, irrespective of the time window. Dropping off a customer early is a time window violation that reduces customer satisfaction, of course, and such scenarios should be explicitly modeled and minimized. Although the drop-off policy in the original implementation produces no time window violations, it does create waiting time in the vehicle for all the customers in the vehicle, and is also undesirable.

#### Redefining Time Window Violation

One of the modifications made to the original STNN heuristic algorithm is redefining time window violation. In the original STNN algorithm, time window violations are calculated in a way that seems incommensurate with the level of dissatisfaction that customers actually experience. Since the objective function attempts to minimize time window violations, as it should, a better definition of what that means would produce better solutions.

When a vehicle arrives early or late at a location, the amount of time outside of the time window should be considered a time window violation.

Assume a vehicle at stop $a$ at time $t_a$ can move to stop $b$, and the time window for $a$ is $[t_1, t_2]$ and the time window for $b$ is $[t_3, t_4]$. Algorithm 6 calculates the arrival time at stop $b$ and stores it in variable $t_b$, and then uses the time window of $b$ alone to determine

how far outside of that, if at all, the value of $t_b$ falls. This is the calculation performed in the improved algorithm. The original algorithm, on the other hand, uses the time window violation calculation shown in Algorithm 7. This calculation considers the time windows for both $a$ and $b$, and includes logic that results in a final value that is lower than a more straightforward interpretation of the actual time window violation.

**Algorithm 6** Modified Time Window Violation Calculation.

```
1: procedure TW_VIOLATION(a, b, t_a, [t_3, t_4])
2:     tw_viol = 0
3:     t_b = t_a + travel_time(a, b)
4:     if t_b < t_3 then
5:         tw_viol = t_3 − t_b
6:     if t_b > t_4 then
7:         tw_viol = t_b − t_4
8:     return tw_viol
```

**Algorithm 7** Original Time Window Violation Calculation.

```
1: procedure TW_VIOLATION(a, b, t, [t_1, t_2], [t_3, t_4])
2:     tw_viol ← 0
3:     e_time = t + travel_time(a, b)
4:     if t > t_2 then
5:         ltime = e_time
6:     if t ≤ t_2 then
7:         ltime = t_2 + travel_time(a, b)
8:     if ltime < t_3 then
9:         tw_viol = t_3 − e_time
10:    if e_time > t_4 then
11:        tw_viol = etime − t_4
12:    return tw_viol
```

The definition of the time window violation is different in different implementations of

the insertion heuristic algorithm. In the improved cheapest insertion heuristic algorithm, the definition discussed above is used.

**Choosing Arrival Time for the First Stop**

Another issue with the insertion heuristic algorithm and space-time nearest neighbor heuristic algorithm is that a fixed start time is considered. For example, the start time in Häme (2011) is considered to be 0, and Baugh et al. (1998) have the vehicle depart the depot so that it arrives at the first customer's EPT. Empirically, such a policy often results in an increase in total time window violation and total ride time. Therefore, better routes could be obtained by considering other arrival times for the first stop.

In both the original and improved cheapest insertion heuristic algorithms, the first stop is chosen by the algorithm. However, in both the original and improved space-time nearest neighbor heuristic algorithms, the first stop in a route is the one with the earliest EPT. Instead of scheduling the vehicle so that it arrives at the very beginning of the time window, however, the improved algorithm performs a search that allows it to arrive at other points of time within the time window.

In the improved cheapest insertion heuristic algorithm and improved space-time nearest neighbor heuristic algorithm, the start time would be calculated using the following approach:

1. Put ST = The smallest EPT among all the customers.
2. Put the start time at first stop as ST.
3. Obtain the route and schedule for pick-ups and drop-offs by solving the cheapest insertion heuristic algorithm (or the space-time nearest neighbor heuristic algorithm).
4. Put ST = ST + $\epsilon$, and update the start time at the first stop based on step 2. Obtain

the new schedule for the current route (the route obtained in previous step), and calculate the new total cost for the current route.

5. Repeat step 4 until the total cost of the current route is improving (stop when it gets worse).

6. Repeat steps 2-5 until no improvement is obtained.

### 4.4.2 Objectives

In the original STNN heuristic algorithm, two objective functions are considered: minimizing total travel time, and minimizing total time window violation. One disadvantage with this objective function is that it treats different types of time window violations the same. However, for customers, it is more important to arrive at an appointment on time than to arrive home on time. Therefore, to produce practical schedules, different weights for different types of time window violation should be considered. Moreover, there are other terms that could be added to the objective function, such as minimizing excess ride time, that would also improve customer satisfaction with the routes and schedules that are produced.

#### Time Window Violation

Each customer has predetermined time windows for both origin and destination points. Because of asymmetries in how customers are affected by different types of time window violations, however, it is important to distinguish between the different types and to weight them appropriately.

The following types of time window violations are defined so that more practical schedules with higher levels of customer satisfaction can be produced:

1. A vehicle arrives early to pick-up a customer
2. A vehicle arrives early to drop-off a customer
3. A vehicle arrives late to drop-off a customer on an outbound request
4. A vehicle arrives late to pick-up a customer, or to drop-off a customer on an inbound request

In case 1, the vehicle should wait until the EPT, and then it can pick-up the customer (this type may called waiting time) since the customer may not otherwise be ready. There may also be other customers in the vehicle waiting, and that amount of time should be minimized to improve customer satisfaction. In cases 2,3, and 4, the vehicle should drop-off the customer immediately at arrival time.

It should be obvious that a time window violation of type 3 is more important than the other types, since it may mean the customer is late for an appointment. Weights chosen in the objective function should reflect these differences in levels of dissatisfaction that a customer may experience.

**Excess Ride Time**

The time spent in the vehicle is clearly important to customers, who have some sense of how long it takes to get from point $A$ to point $B$. Ride time for a customer is minimal when the vehicle goes directly from the customer's pick-up location to the requested drop-off location without servicing other customers. Although taxi-like service such as this is more costly, an effort should be made to keep time on the vehicle in excess of the minimum from being disproportionately large.

Consider the following terms:

$T_T$ : Total travel time

$TWV_1$ : Total time window violation type 1

$TWV_2$ : Total time window violation type 2

$TWV_3$ : Total time window violation type 3

$TWV_4$ : Total time window violation type 4

$R_T$ : Total excess ride time

$w_1$ : Weight on total travel time

$w_2$ : Weight on total time window violation type 1

$w_3$ : Weight on total time window violation type 2

$w_4$ : Weight on total time window violation type 3

$w_5$ : Weight on total time window violation type 4

$w_6$ : Weight on total excess ride time

In its final form, the objective function in the improved algorithm can be expressed as follows:

$$Obj = w_1 * T_T + w_2 * TWV_1 + w_3 * TWV_2 + w_4 * TWV_3 + w_5 * TWV_4 + w_6 * R_T$$

Where weights $w_i$ are chosen to reflect the relative importance associated with the various terms. It includes six objectives so that the following are minimized: total travel time, total excess ride time, total waiting time, and total time window violation, and where time window violation is further separated into three terms to account for differences in relative importance.

In different implementations of the insertion heuristic algorithm, different objective functions are used. In the improved cheapest insertion heuristic algorithm the objectives described above are used.

## 4.5 Route Improvement Heuristic

One of the main problems with both the STNN heuristic algorithm and advanced insertion heuristic algorithm is that in each iteration of the algorithm, the order of previously added vertices to the route would not change. In STNN heuristic algorithm not only the order will not change, but also new vertices can only be added at the end of the existing subroute. In other words, the new vertices cannot be added in between of previously added vertices. On the other hand, in insertion heuristic algorithm, new vertices could be added in between of previously added vertices. That is why the search space of the insertion heuristic algorithm is larger than the STNN heuristic algorithm.

The *two-opt* algorithm is a local search improvement heuristic algorithm, which means that the algorithm starts with a route and improves it iteratively. The algorithm is fast and it is used widely in the literature to improve routes in the traveling salesman problem, vehicle routing problem, and similar problems (see for example Golden et al. (1980); Li et al. (2005); Yu et al. (2009)). Intuitively,the main idea behind the algorithm is to take a route that crosses over itself and reorder it so that it does not. Specifically, this local search improvement method tests all possible pairwise exchanges of vertices to see if an improvement in the objective function can be obtained.

The two-opt heuristic algorithm is described in Algorithm 8. This improvement algorithm is used to improve the routes obtained by the improved STNN heuristic algorithm and improved cheapest insertion heuristic algorithm. The two-opt algorithm is used to improve the final routes obtained by these two algorithms.

**Algorithm 8** Two-Opt heuristic algorithm.

```
1:  procedure TWO-OPT(route)
2:      best = route
3:      improved = True
4:      while improved do
5:          improved = False
6:          for i ∈ {1, 2, ..., |best| − 1} do
7:              for j ∈ {i, i + 1, ..., |best|} do
8:                  new-route = route[: i] + route[i : j][:: −1] + route[j :]
9:                  if new-route is feasible and Cost(new-route) < Cost(best) then
10:                     best = new-route
11:                     improved = True
12:         route = best
13:     return best
```

CHAPTER

# 5

# EXPERIMENTS WITH THE SINGLE VEHICLE DIAL-A-RIDE PROBLEM

This chapter presents the results of computational experiments performed with exact and heuristic approaches for solving the single vehicle dial-a-ride problem. The six heuristic algorithms are as follows:

**STNN**: The original space-time nearest neighbor heuristic algorithm using the same objective functions and service policies introduced by Baugh et al. (1998).

**ISTNN**: An improved version of the STNN algorithm that includes modified objective functions and service policies, as described in the previous chapter.

**ISTNN+2OPT**: A version of the ISTNN algorithm that includes, at the end, a local search using the two-opt algorithm.

**INS**: The cheapest insertion heuristic algorithm presented in Algorithm 5 with objective functions and service policies comparable to those used in the STNN algorithm.

**IINS**: An improved version of the INS algorithm that includes the modified objective functions and service policies used in the ISTNN algorithm.

**IINS+2OPT**: A version of the IINS algorithm that includes, at the end, a local search using the two-opt algorithm.

The objective of these experiments is to evaluate the effectiveness of heuristic approaches for solving the problem, and to compare the solutions obtained by different approaches. To this end, two collections of instances are used, both of which draw from operational datasets provided by the Winston Salem Transit Authority (WSTA). The first collection, named collection A, contains twenty four cases of various sizes (small, medium, and large), which are used to compare the heuristic algorithms with each other. The second collection, named collection B, contains fifteen cases, all small, which are used to compare the heuristic algorithms with the exact approach.

All algorithms are implemented in Python. The MILP model is also implemented in Python and solved using the Gurobi solver. All experiments are performed on a 3.4 GHz Core i7 computer with 16 GB memory.

## 5.1 STNN Heuristic Algorithm and Variants

In this section, the STNN heuristic algorithm is compared with the ISTNN and ISTNN+2OPT algorithms. Collection A cases are solved by these algorithms and results are reported in Tables 5.1 and 5.2.

Table 5.1: Comparison of total cost (TC) and total run time (RT) for collection A cases in STNN, ISTNN, and ISTNN+2OPT heuristic algorithms.

| | | **STNN** | | **ISTNN** | | **ISTNN+2OPT** | |
|---|---|---|---|---|---|---|---|
| **Case** | **Size** | **TC** | **RT** | **TC** | **RT** | **TC** | **RT** |
| 1 | 2 | 610.56 | 0.0010 | 538.27 | 0.0010 | 431.14 | 0.0010 |
| 2 | 3 | 144.36 | 0.0010 | 144.18 | 0.0020 | 142.36 | 0.0030 |
| 3 | 4 | 1258.75 | 0.0020 | 1149.17 | 0.0030 | 1026.08 | 0.0060 |
| 4 | 6 | 1033.78 | 0.0040 | 889.84 | 0.0110 | 887.73 | 0.0160 |
| 5 | 6 | 1241.88 | 0.0030 | 1299.22 | 0.0090 | 1142.88 | 0.0190 |
| 6 | 7 | 371.80 | 0.0040 | 241.15 | 0.0160 | 241.15 | 0.0270 |
| 7 | 7 | 1027.66 | 0.0040 | 864.23 | 0.0110 | 723.82 | 0.0200 |
| 8 | 8 | 310.06 | 0.0090 | 301.29 | 0.0180 | 236.96 | 0.0340 |
| 9 | 9 | 1096.35 | 0.0080 | 1119.86 | 0.0170 | 801.26 | 0.0460 |
| 10 | 10 | 1348.84 | 0.0110 | 1395.43 | 0.0230 | 1055.46 | 0.0800 |
| 11 | 11 | 1000.18 | 0.0160 | 719.29 | 0.0468 | 611.06 | 0.1570 |
| 12 | 11 | 1520.37 | 0.0120 | 1115.43 | 0.0260 | 957.15 | 0.0690 |
| 13 | 11 | 1219.50 | 0.0140 | 849.65 | 0.0430 | 772.15 | 0.1010 |
| 14 | 11 | 733.20 | 0.0120 | 846.80 | 0.0280 | 483.86 | 0.0710 |
| 15 | 12 | 1549.83 | 0.0140 | 1221.58 | 0.0460 | 1155.30 | 0.0730 |
| 16 | 12 | 1262.33 | 0.0140 | 559.73 | 0.0410 | 557.10 | 0.0624 |
| 17 | 12 | 622.92 | 0.0156 | 492.56 | 0.0312 | 343.12 | 0.0624 |
| 18 | 13 | 1012.65 | 0.0160 | 1294.21 | 0.0500 | 1292.21 | 0.0940 |
| 19 | 13 | 909.56 | 0.0210 | 737.78 | 0.0610 | 658.71 | 0.1890 |
| 20 | 13 | 817.00 | 0.0160 | 883.15 | 0.0312 | 578.30 | 0.0936 |
| 21 | 14 | 2020.70 | 0.0240 | 1421.50 | 0.0430 | 1286.22 | 0.1220 |
| 22 | 14 | 1505.90 | 0.0190 | 1012.28 | 0.0600 | 899.34 | 0.1660 |
| 23 | 16 | 1079.39 | 0.0230 | 1016.28 | 0.0770 | 995.21 | 0.2210 |
| 24 | 17 | 1801.13 | 0.0280 | 1479.74 | 0.0840 | 1318.29 | 0.2360 |
| **Avg.** | | 1062.45 | 0.0122 | 899.69 | 0.0325 | 774.87 | 0.0821 |

Table 5.2: Comparison of total travel time (TT), total time window violation (TWV), and total excess ride time (ERT) for collection A cases in STNN, ISTNN, and ISTNN+2OPT heuristic algorithms.

| | | **STNN** | | | **ISTNN** | | | **ISTNN+2OPT** | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Case** | **Size** | **TT** | **TWV** | **ERT** | **TT** | **TWV** | **ERT** | **TT** | **TWV** | **ERT** |
| 1 | 2 | 21.89 | 88.40 | 84.31 | 20.48 | 82.48 | 46.16 | 20.69 | 7.65 | 9.67 |
| 2 | 3 | 7.12 | 0.74 | 1.27 | 7.12 | 1.29 | 0.53 | 7.12 | 0.00 | 0.00 |
| 3 | 4 | 38.14 | 228.26 | 267.73 | 45.71 | 227.41 | 7.57 | 45.71 | 104.32 | 7.57 |
| 4 | 6 | 40.35 | 179.50 | 20.82 | 40.35 | 82.75 | 0.01 | 40.19 | 83.62 | 0.38 |
| 5 | 6 | 27.90 | 626.41 | 52.26 | 24.76 | 712.30 | 85.59 | 24.76 | 638.23 | 2.39 |
| 6 | 7 | 12.06 | 32.99 | 97.67 | 12.06 | 0.00 | 0.02 | 12.06 | 0.00 | 0.02 |
| 7 | 7 | 34.79 | 241.82 | 67.62 | 27.54 | 306.22 | 7.26 | 27.54 | 165.81 | 7.26 |
| 8 | 8 | 11.90 | 26.69 | 45.43 | 11.90 | 38.84 | 24.51 | 11.85 | 0.00 | 0.01 |
| 9 | 9 | 29.82 | 221.25 | 261.15 | 31.03 | 390.12 | 109.10 | 29.41 | 213.06 | 0.01 |
| 10 | 10 | 49.92 | 154.45 | 195.97 | 49.83 | 244.28 | 154.57 | 43.84 | 104.86 | 73.79 |
| 11 | 11 | 37.23 | 49.00 | 206.55 | 31.09 | 5.63 | 91.77 | 26.50 | 35.33 | 45.79 |
| 12 | 11 | 34.57 | 331.06 | 454.59 | 32.48 | 448.94 | 16.86 | 32.48 | 302.27 | 5.21 |
| 13 | 11 | 32.66 | 204.42 | 361.81 | 32.74 | 183.41 | 11.52 | 29.19 | 188.34 | 0.09 |
| 14 | 11 | 27.19 | 39.32 | 150.12 | 24.19 | 350.80 | 12.17 | 24.19 | 0.00 | 0.04 |
| 15 | 12 | 57.06 | 303.52 | 73.30 | 44.12 | 244.47 | 85.93 | 44.12 | 258.28 | 14.59 |
| 16 | 12 | 33.54 | 280.52 | 310.94 | 23.92 | 73.28 | 8.06 | 23.92 | 69.85 | 8.06 |
| 17 | 12 | 21.68 | 59.82 | 129.56 | 19.01 | 49.50 | 62.87 | 16.58 | 7.41 | 4.16 |
| 18 | 13 | 30.97 | 284.76 | 108.43 | 43.54 | 347.66 | 12.67 | 43.37 | 347.28 | 14.51 |
| 19 | 13 | 30.95 | 56.39 | 234.15 | 32.59 | 21.19 | 64.75 | 29.22 | 20.00 | 54.24 |
| 20 | 13 | 25.30 | 161.80 | 149.26 | 25.30 | 359.14 | 18.06 | 23.66 | 101.80 | 3.32 |
| 21 | 14 | 58.50 | 344.68 | 399.67 | 47.12 | 279.16 | 171.21 | 47.13 | 300.04 | 14.75 |
| 22 | 14 | 45.01 | 336.32 | 151.54 | 35.48 | 248.24 | 54.52 | 34.66 | 185.09 | 19.69 |
| 23 | 16 | 43.68 | 129.67 | 76.10 | 38.96 | 205.06 | 32.11 | 38.78 | 204.60 | 14.99 |
| 24 | 17 | 57.56 | 407.47 | 242.40 | 53.03 | 289.23 | 129.96 | 53.87 | 153.00 | 87.92 |
| | **Avg.** | 33.74 | 199.55 | 172.61 | 31.43 | 216.31 | 50.32 | 30.45 | 145.45 | 16.19 |

In Table 5.1, the total cost and run time of the STNN, ISTNN, and ISTNN+2OPT heuristic algorithms are compared. It should be noted that the objective function of the STNN algorithm is different from that of the ISTNN and ISTNN+2OPT algorithms. In order to compare the solutions obtained by these algorithms, each solution is evaluated using the same cost function:

$$TC = w_1 * T_T + w_2 * TWV_1 + w_3 * TWV_2 + w_4 * TWV_3 + w_5 * TWV_4 + w_6 * R_T \tag{5.1}$$

In which:

$T_T$ : Total travel time

$TWV_1$ : Total time window violation type 1

$TWV_2$ : Total time window violation type 2

$TWV_3$ : Total time window violation type 3

$TWV_4$ : Total time window violation type 4

$w_1$ : Weight on total travel time

$w_2$ : Weight on total time window violation type 1

$w_3$ : Weight on total time window violation type 2

$w_4$ : Weight on total time window violation type 3

$w_5$ : Weight on total time window violation type 4

It can be seen that, on average, the total cost in STNN heuristic algorithm is 1062.45, for ISTNN heuristic algorithm is 899.69, and for the ISTNN+2OPT heuristic algorithm is 774.87. Therefore, the ISTNN heuristic algorithm has improved the total cost by 15.32% on average, and the ISTNN heuristic algorithm combined with two-opt algorithm has improved the total cost by 27.07% on average.

It is expected that improvements in the STNN algorithm and the addition of a two-opt

local search at the end will increase running time.[1] It can be seen that on average the total running time in STNN heuristic algorithm is 0.0122 seconds, for ISTNN heuristic algorithm is 0.0325 seconds and for the ISTNN+2OPT heuristic algorithm is 0.0821 seconds. Therefore, the running time is increased compared with the original STNN heuristic algorithm. However, both the ISTNN heuristic algorithm and ISTNN+2OPT heuristic algorithm are fast enough to be used to solve practical size instances of single vehicle dial-a-ride problems in a short time.

In Table 5.2, the total travel time, total time window violation, and total excess ride time of the STNN, ISTNN, and ISTNN+2OPT heuristic algorithms are compared. It can be seen that on average the total travel time, total time window violation, and total excess ride time in STNN heuristic algorithm are 33.74, 199.55, and 172.61, for ISTNN heuristic algorithm are 31.43, 216.31, and 50.32 and for the ISTNN+2OPT heuristic algorithm are 30.45, 145.45, and 16.19. Therefore, the ISTNN heuristic algorithm has reduced the total travel time, total time window violation, and total excess ride time by 6.85%, −7.75%, and 70.85% on average, and the ISTNN heuristic algorithm combined with two-opt algorithm has reduced the total travel time, total time window violation, and total excess ride time by 9.75%, 27.11, and 90.62% on average.

## 5.2 Insertion Heuristic Algorithm and Variants

In this section, the INS heuristic algorithm is compared with the IINS heuristic algorithm and the IINS+2OPT heuristic algorithm. The collection A cases are solved by these algorithms and results are reported in Tables 5.3, 5.4.

In Table 5.3, the total cost and running time of the INS, IINS, and IINS+2OPT heuristic

[1] Although the two-opt algorithm has a worst-case performance that is exponential (Englert et al. 2007), in practice it is often subquadratic (Johnson and McGeoch 1997).

Table 5.3: Comparison of total cost (TC) and total run time (RT) for collection A cases in INS, IINS, and IINS+2OPT heuristic algorithms.

| | | **INS** | | **IINS** | | **IINS+2OPT** | |
|---|---|---|---|---|---|---|---|
| **Case** | **Size** | **TC** | **RT** | **TC** | **RT** | **TC** | **RT** |
| 1 | 2 | 450.14 | 0.0010 | 415.70 | 0.0020 | 415.70 | 0.0020 |
| 2 | 3 | 145.57 | 0.0010 | 142.36 | 0.0030 | 142.36 | 0.0040 |
| 3 | 4 | 1055.95 | 0.0030 | 951.96 | 0.0100 | 951.96 | 0.0140 |
| 4 | 6 | 1072.54 | 0.0130 | 629.93 | 0.0390 | 625.81 | 0.0450 |
| 5 | 6 | 1224.24 | 0.0130 | 1139.46 | 0.0312 | 1139.46 | 0.0460 |
| 6 | 7 | 332.07 | 0.0220 | 237.23 | 0.0420 | 237.23 | 0.0520 |
| 7 | 7 | 750.78 | 0.0230 | 748.35 | 0.0680 | 748.35 | 0.0800 |
| 8 | 8 | 261.07 | 0.0312 | 236.96 | 0.0700 | 236.96 | 0.0880 |
| 9 | 9 | 941.48 | 0.0468 | 669.44 | 0.1690 | 669.44 | 0.1850 |
| 10 | 10 | 1146.77 | 0.0850 | 1226.50 | 0.1780 | 1226.50 | 0.2000 |
| 11 | 11 | 724.92 | 0.1200 | 574.26 | 0.3550 | 573.97 | 0.3890 |
| 12 | 11 | 1533.68 | 0.1232 | 964.24 | 0.2400 | 957.15 | 0.2706 |
| 13 | 11 | 1472.14 | 0.1220 | 772.15 | 0.3598 | 772.15 | 0.4066 |
| 14 | 11 | 846.79 | 0.1092 | 559.00 | 0.2340 | 559.00 | 0.2652 |
| 15 | 12 | 1397.21 | 0.1680 | 1213.26 | 0.6450 | 1213.26 | 0.6606 |
| 16 | 12 | 573.92 | 0.1670 | 443.85 | 0.4780 | 443.85 | 0.5050 |
| 17 | 12 | 447.70 | 0.1660 | 307.91 | 0.4680 | 307.91 | 0.5148 |
| 18 | 13 | 1012.65 | 0.2360 | 877.50 | 0.6552 | 877.50 | 0.6864 |
| 19 | 13 | 564.09 | 0.2290 | 544.83 | 0.6730 | 544.83 | 0.7190 |
| 20 | 13 | 828.57 | 0.2310 | 574.08 | 0.6780 | 574.08 | 0.7120 |
| 21 | 14 | 1424.03 | 0.3050 | 1229.81 | 0.8736 | 1197.04 | 0.9426 |
| 22 | 14 | 1093.39 | 0.3060 | 890.18 | 0.8932 | 890.18 | 0.9244 |
| 23 | 16 | 1036.71 | 0.5100 | 696.33 | 1.4430 | 696.33 | 1.5140 |
| 24 | 17 | 1253.78 | 0.6440 | 1136.48 | 1.8482 | 1136.48 | 1.8950 |
| | **Avg.** | 899.59 | 0.1531 | 715.91 | 0.4357 | 714.06 | 0.4634 |

Table 5.4: Comparison of total travel time (TT), total time window violation (TWV), and total excess ride time (ERT) for collection A instances in INS, IINS, and IINS+2OPT heuristic algorithms.

| | | INS | | | IINS | | | IINS+2OPT | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Case** | **Size** | **TT** | **TWV** | **ERT** | **TT** | **TWV** | **ERT** | **TT** | **TWV** | **ERT** |
| 1 | 2 | 20.69 | 26.65 | 9.67 | 20.48 | 12.48 | 0.00 | 20.48 | 6.09 | 0.00 |
| 2 | 3 | 7.12 | 0.74 | 2.52 | 7.12 | 1.29 | 0.00 | 7.12 | 0.00 | 0.00 |
| 3 | 4 | 38.24 | 163.33 | 127.78 | 38.14 | 190.65 | 8.54 | 38.14 | 189.19 | 0.00 |
| 4 | 6 | 35.96 | 211.92 | 136.06 | 27.03 | 89.28 | 0.02 | 27.03 | 85.15 | 0.02 |
| 5 | 6 | 26.54 | 612.22 | 81.12 | 24.76 | 635.46 | 5.90 | 24.76 | 635.46 | 5.90 |
| 6 | 7 | 11.86 | 31.61 | 63.25 | 11.86 | 202.80 | 0.02 | 11.86 | 0.00 | 0.02 |
| 7 | 7 | 22.85 | 187.98 | 105.86 | 22.80 | 231.32 | 60.96 | 22.80 | 231.32 | 60.96 |
| 8 | 8 | 11.85 | 3.19 | 20.93 | 11.85 | 39.34 | 0.01 | 11.85 | 0.00 | 0.01 |
| 9 | 9 | 25.97 | 170.20 | 251.82 | 24.79 | 173.38 | 15.55 | 24.79 | 167.41 | 6.15 |
| 10 | 10 | 43.73 | 191.79 | 51.57 | 31.25 | 640.15 | 20.66 | 31.25 | 520.15 | 20.66 |
| 11 | 11 | 26.49 | 47.88 | 147.31 | 26.51 | 23.13 | 20.87 | 26.50 | 23.16 | 20.82 |
| 12 | 11 | 32.48 | 333.15 | 507.63 | 32.48 | 453.86 | 14.75 | 32.48 | 302.27 | 5.21 |
| 13 | 11 | 37.51 | 223.74 | 498.26 | 29.19 | 188.34 | 0.09 | 29.19 | 188.34 | 0.09 |
| 14 | 11 | 24.19 | 29.92 | 333.05 | 24.19 | 275.66 | 87.31 | 24.19 | 0.00 | 75.18 |
| 15 | 12 | 38.83 | 224.81 | 367.83 | 34.22 | 431.64 | 57.26 | 34.22 | 431.64 | 57.26 |
| 16 | 12 | 18.73 | 129.47 | 69.76 | 18.75 | 68.86 | 0.04 | 18.75 | 68.86 | 0.04 |
| 17 | 12 | 14.51 | 46.32 | 111.27 | 14.51 | 9.45 | 8.35 | 14.51 | 9.45 | 8.35 |
| 18 | 13 | 30.97 | 284.76 | 108.43 | 30.99 | 257.66 | 0.08 | 30.99 | 257.66 | 0.08 |
| 19 | 13 | 25.68 | 25.62 | 24.92 | 25.67 | 30.51 | 0.83 | 25.67 | 31.29 | 0.05 |
| 20 | 13 | 23.66 | 162.81 | 192.58 | 23.66 | 100.89 | 0.00 | 23.66 | 100.89 | 0.00 |
| 21 | 14 | 46.33 | 336.78 | 131.91 | 44.31 | 312.79 | 1.81 | 42.66 | 314.45 | 0.46 |
| 22 | 14 | 35.76 | 205.74 | 172.39 | 34.43 | 187.54 | 12.54 | 34.43 | 187.54 | 12.54 |
| 23 | 16 | 31.64 | 132.12 | 271.89 | 31.66 | 62.39 | 0.67 | 31.66 | 62.39 | 0.67 |
| 24 | 17 | 48.16 | 175.58 | 115.08 | 53.55 | 75.36 | 0.19 | 53.55 | 65.36 | 0.19 |
| | **Avg.** | 28.32 | 164.93 | 162.62 | 26.84 | 195.59 | 13.19 | 26.77 | 161.59 | 11.45 |

algorithms are compared. Similar to the previous section, the objective function of INS heuristic algorithm is different than the objective function of IINS heuristic algorithm and IINS+2OPT heuristic algorithm. In order to compare the solutions obtained by these algorithms, each solution is evaluated using the cost function 5.1.

It can be seen that on average the total cost in INS heuristic algorithm is 899.59, for IINS heuristic algorithm is 715.91 and for the IINS+2OPT heuristic algorithm is 714.06. Therefore, the IINS heuristic algorithm has improved the total cost by 20.62% on average, and the IINS heuristic algorithm combined with two-opt algorithm has improved the total cost by 20.62% on average.

It is again expected that improvements in the INS algorithm and the addition of a two-opt local search at the end will increase running time. It can be seen that on average the total running time in INS heuristic algorithm is 0.1531 seconds , for IINS heuristic algorithm is 0.4357 seconds, and for the IINS+2OPT heuristic algorithm is 0.4634 seconds. Therefore, the running time is increased compared with the original INS heuristic algorithm. However, both the IINS heuristic algorithm and IINS+2OPT heuristic algorithm are fast enough to be used to solve practical size instances of single vehicle dial-a-ride problem in short time.

In Table 5.4, the total travel time, total time window violation, and total excess ride time of the INS, IINS, and IINS+2OPT heuristic algorithms are compared. It can be seen that on average the total travel time, total time window violation, and total excess ride time in INS heuristic algorithm are 28.32, 164.93, and 162.62, for IINS heuristic algorithm are 26.84, 195.59, and 13.19 and for the IINS+2OPT heuristic algorithm are 26.77, 161.59, and 11.45. Therefore, the IINS heuristic algorithm has improved the total travel time, total time window violation, and total excess ride time by 5.22%, −15.67%, and 91.89% on average, and the IINS heuristic algorithm combined with two-opt algorithm has improved the total travel time, total time window violation, and total excess ride time by 5.47%, 2.02, and 92.96% on average.

## 5.3 Comparison of the Heuristic Algorithms

In this section, the STNN, ISTNN, and ISTNN+2OPT heuristic algorithms are compared with INS, IINS, and IINS+2OPT heuristic algorithms.

The total cost and total run time for STNN heuristic algorithm variants reported in Table 5.1 and the total cost and total run time for INS heuristic algorithm variants reported in Table 5.3 are plotted in Figures 5.1 and 5.2, and summarized in Table 5.5.

Table 5.5: Comparison of average total cost (TC), and running time (RT) for collection A cases in STNN, INS, ISTNN, IINS, ISTNN+2OPT, and IINIS+2OPT heuristic algorithms.

| | **STNN** | **INS** | **ISTNN** | **IINS** | **ISTNN+2OPT** | **IINS+2OPT** |
|---|---|---|---|---|---|---|
| **TC** | 1062.45 | 899.59 | 899.69 | 715.91 | 774.87 | 714.06 |
| **RT** | 0.0122 | 0.1531 | 0.0325 | 0.4357 | 0.0821 | 0.4634 |

It can be seen that, on average, the STNN, ISTNN, and ISTNN+2OPT heuristic algorithms are much faster than the INS, IINS, and IINS+2OPT heuristic algorithms, respectively. On the other hand, the quality of solutions obtained by the INS, IINS, and IINS+2OPT heuristic algorithms are somewhat better than the STNN, ISTNN, and ISTNN+2OPT heuristic algorithms, respectively. The STNN heuristic algorithm is the fastest algorithm among all six heuristic algorithms introduced to solve the single vehicle dial-a-ride problem. On the other hand, it has the highest average total cost. Also, the IINS+2OPT heuristic algorithm has the best average total cost, it is about 33% lower than the average total cost in STNN. On the other hand it has the longest run time, it is about 38 times greater than the run time of the STNN heuristic algorithm.

To compare these algorithms, the solutions obtained by each of the six heuristic algorithms in 24 cases in collection A are normalized. A trade-off plot between time and total cost is created. This plot is shown in Figure 5.3 and the results of this plot are summarized

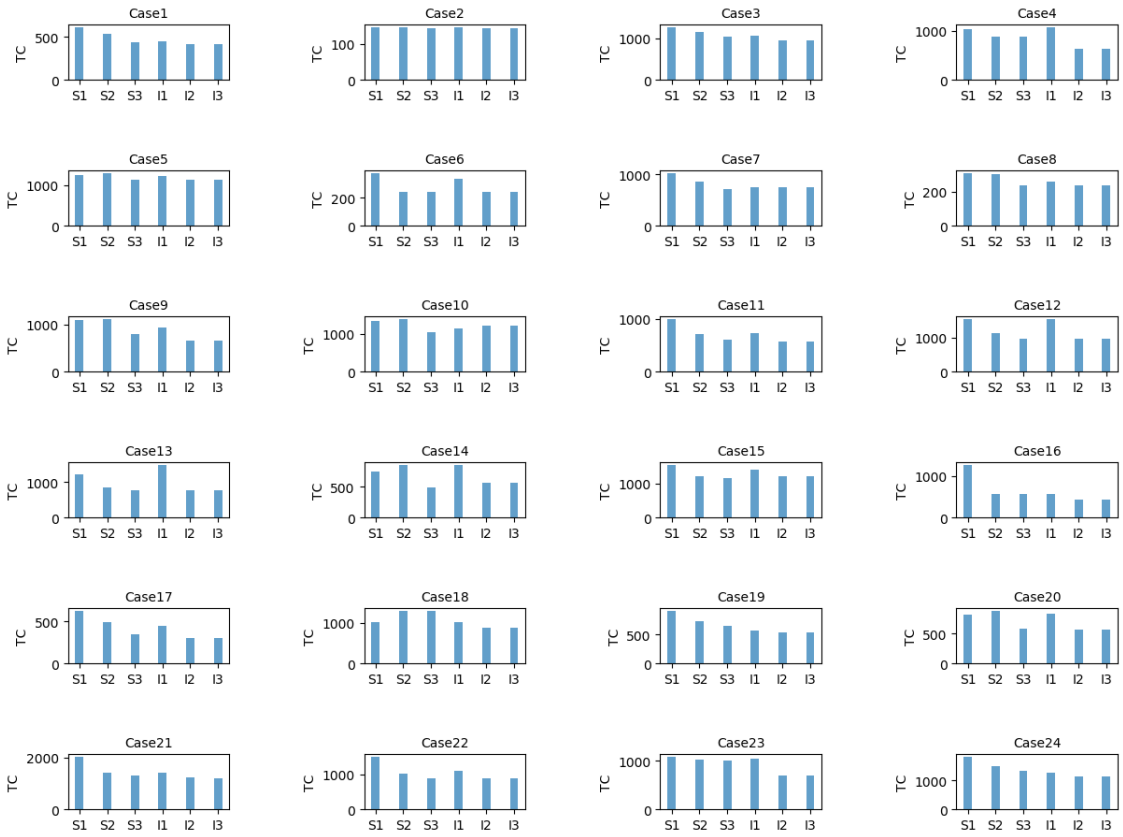


Figure 5.1: Comparison of total cost for collection A cases in STNN, ISTNN, ISTNN+2OPT, INS, IINS, ans IINS+2OPT heuristic algorithms.

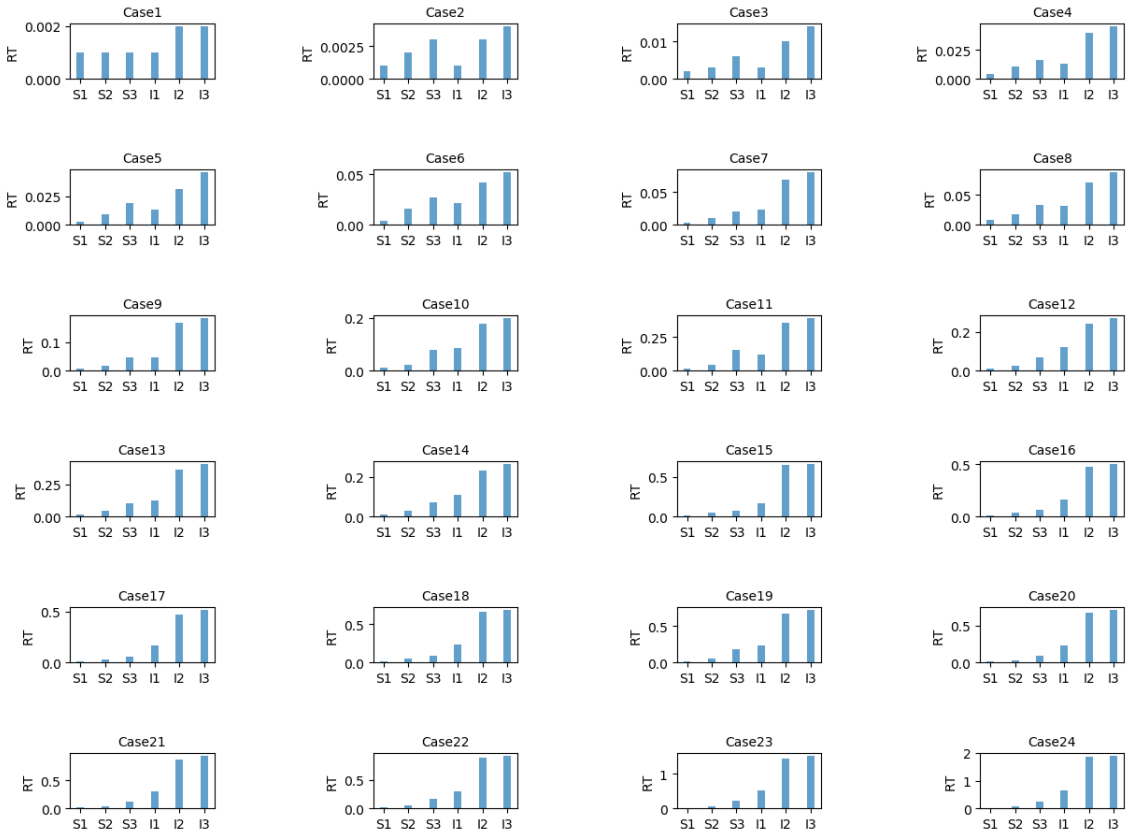


Figure 5.2: Comparison of total running time for collection A cases in STNN, ISTNN, ISTNN+2OPT, INS, IINS, ans IINS+2OPT heuristic algorithms.

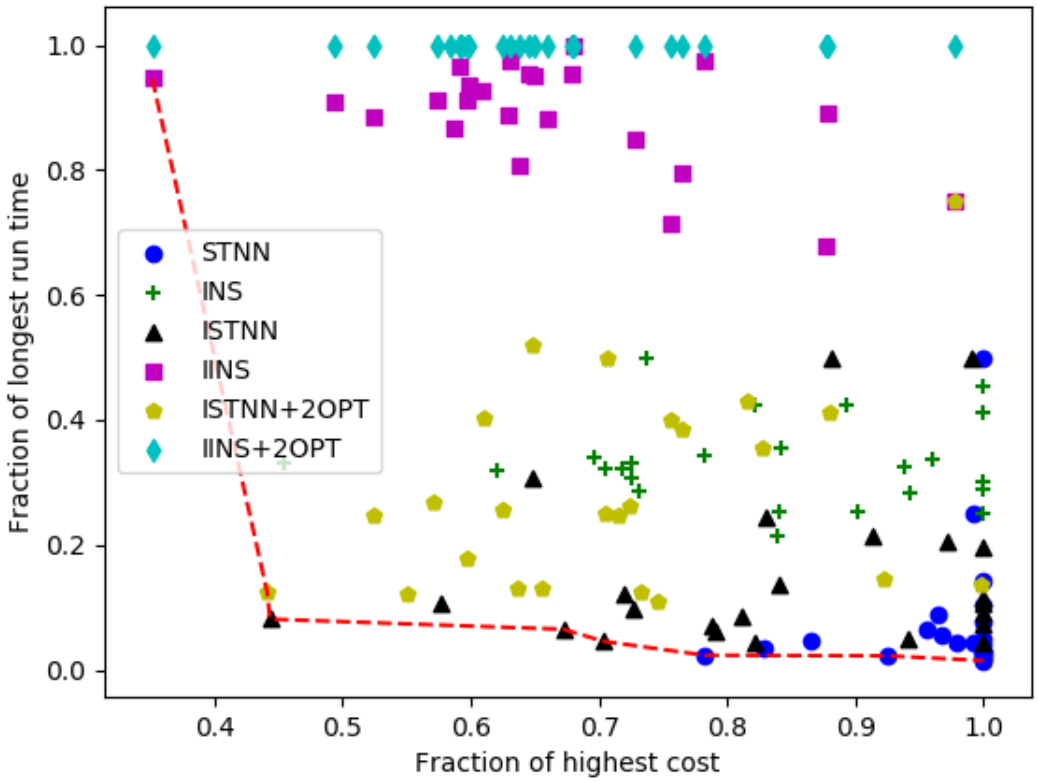


Figure 5.3: Trade-off between time and cost generated for collection A cases using STNN, ISTNN, ISTNN+2OPT, INS, IINS, IINS+2OPT heuristic algorithms. Each point is normalized to be a fraction of the highest cost and longest run time for the given problem instance.

in Table 5.6. The red line in the Figure 5.3 connects the non-dominated solutions. It can be seen that 3 solutions obtained by STNN heuristic algorithm, 3 solutions obtained by ISTNN heuristic algorithm, 1 solution obtained by IINS heuristic algorithm, and 1 solutions obtained by ISTNN+2OPT heuristic algorithm are the non-dominated solutions.

Table 5.6: Comparing STNN, ISTNN, ISTNN+2OPT, INS, IINS, IINS+2OPT heuristic algorithms based on the number of times they are selected as a non-dominated algorithm in collection A cases.

| **Algorithm** | **# non-dominated** |
|---|---|
| STNN | 3 |
| INS | 0 |
| ISTNN | 3 |
| IINS | 1 |
| ISTNN+2OPT | 1 |
| IINS+2OPT | 0 |

## 5.4 Comparison with Exact Solutions

In previous sections, the STNN and INS heuristic algorithm variants are compared with each other, and it can be seen that the improvements made to them, combined with a two-opt local search, yield better quality solutions. In this section, both the ISTNN+2OPT and IINS+2OPT algorithms are compared with an equivalent MILP model. The MILP model and heuristic algorithms all have the same objective function and service policies, facilitating comparisons between them.

The collection B cases are solved by MILP model, ISTNN+2OPT, and IINS+2OPT heuristic algorithms, and results are reported in Tables 5.3 and 5.8.

In Table 5.7, the total cost and run time of the MILP exact model, ISTNN+2OPT, and IINS+2OPT heuristic algorithms are compared. It can be seen that, on average, the total cost for the MILP model is 394.44, for the ISTNN+2OPT heuristic algorithm is 412.27, and for the IINS+2OPT heuristic algorithm is 405.36. Therefore, the total cost in ISTNN+2OPT is 4.32% more the the optimal total cost on average, and the total cost in IINS+2OPT is 2.69% more than the optimal total cost on average. Therefore, the total cost for both heuristic algorithms is very close to the exact solution.

It is expected that the heuristic algorithms have shorter run time. It can be seen that on

Table 5.7: Comparison of total cost (TC) and run time (RT) for collection B cases in MILP model, ISTNN+2OPT and IINS+2OPT heuristic algorithms.

| | | **MILP** | | **ISTNN+2OPT** | | **IINS+2OPT** | |
|---|---|---|---|---|---|---|---|
| **Case** | **Size** | **TC** | **RT** | **TC** | **RT** | **TC** | **RT** |
| 1 | 3 | 142.36 | 0.1409 | 142.36 | 0.0030 | 142.36 | 0.0040 |
| 2 | 4 | 950.15 | 1.6749 | 1026.08 | 0.0080 | 951.96 | 0.0170 |
| 3 | 5 | 298.38 | 1.2056 | 302.51 | 0.0160 | 302.51 | 0.0300 |
| 4 | 6 | 314.19 | 1.8284 | 362.98 | 0.0220 | 362.98 | 0.0530 |
| 5 | 6 | 494.14 | 39.3788 | 505.92 | 0.0312 | 495.03 | 0.0624 |
| 6 | 6 | 680.72 | 15.9472 | 684.96 | 0.0156 | 684.96 | 0.0312 |
| 7 | 6 | 316.49 | 3.7494 | 316.59 | 0.0320 | 316.59 | 0.0530 |
| 8 | 6 | 302.62 | 60.7582 | 302.62 | 0.0200 | 302.62 | 0.0540 |
| 9 | 6 | 236.95 | 8.2321 | 236.95 | 0.0190 | 236.95 | 0.0380 |
| 10 | 6 | 574.42 | 7.0488 | 622.63 | 0.0160 | 591.79 | 0.0370 |
| 11 | 6 | 249.61 | 43.3433 | 292.03 | 0.0170 | 250.31 | 0.0520 |
| 12 | 6 | 316.49 | 4.6251 | 316.59 | 0.0360 | 316.59 | 0.0468 |
| 13 | 7 | 223.73 | 9.5231 | 241.15 | 0.0290 | 237.23 | 0.0700 |
| 14 | 7 | 520.44 | 9.7181 | 534.80 | 0.0280 | 592.74 | 0.0640 |
| 15 | 7 | 295.85 | 0.0700 | 295.85 | 0.0312 | 295.85 | 0.0690 |
| **Avg.** | | 394.44 | 13.8163 | 412.27 | 0.0216 | 405.36 | 0.0454 |

Table 5.8: Comparison of total travel time (TT), total time window violation (TWV), and total excess ride time (ERT) for collection B cases in MILP model, ISTNN+2OPT and IINS+2OPT heuristic algorithms.

| | | **MILP** | | | **ISTNN+2OPT** | | | **IINS+2OPT** | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Case** | **Size** | **TT** | **TWV** | **ERT** | **TT** | **TWV** | **ERT** | **TT** | **TWV** | **ERT** |
| 1 | 3 | 7.12 | 0.00 | 0.00 | 7.12 | 0.00 | 0.00 | 7.12 | 0.00 | 0.00 |
| 2 | 4 | 38.11 | 188.00 | 0.00 | 45.71 | 104.32 | 7.57 | 38.14 | 189.19 | 0.00 |
| 3 | 5 | 14.49 | 8.55 | 0.01 | 14.49 | 4.41 | 8.29 | 14.49 | 4.41 | 8.29 |
| 4 | 6 | 13.98 | 34.53 | 0.00 | 16.35 | 35.88 | 0.00 | 16.35 | 35.88 | 0.00 |
| 5 | 6 | 23.19 | 22.68 | 7.68 | 23.19 | 23.55 | 18.60 | 23.19 | 23.55 | 7.70 |
| 6 | 6 | 28.93 | 102.17 | 0.00 | 28.93 | 102.18 | 4.27 | 28.93 | 102.18 | 4.27 |
| 7 | 6 | 14.71 | 22.38 | 0.00 | 14.71 | 22.38 | 0.03 | 14.71 | 22.38 | 0.03 |
| 8 | 6 | 14.50 | 4.40 | 8.28 | 14.50 | 4.40 | 8.28 | 14.50 | 4.40 | 8.28 |
| 9 | 6 | 11.85 | 0.00 | 0.00 | 11.85 | 0.00 | 0.01 | 11.85 | 0.00 | 0.01 |
| 10 | 6 | 25.32 | 63.76 | 4.21 | 26.79 | 86.44 | 0.36 | 25.32 | 81.11 | 4.21 |
| 11 | 6 | 12.38 | 0.00 | 2.08 | 14.60 | 0.00 | 0.03 | 12.51 | 0.00 | 0.03 |
| 12 | 6 | 14.71 | 22.38 | 0.00 | 14.71 | 22.38 | 0.03 | 14.71 | 22.38 | 0.03 |
| 13 | 7 | 11.19 | 0.00 | 0.02 | 12.06 | 0.00 | 0.02 | 11.86 | 0.00 | 0.02 |
| 14 | 7 | 25.57 | 9.05 | 0.00 | 25.57 | 23.41 | 0.01 | 24.64 | 99.50 | 0.48 |
| 15 | 7 | 14.79 | 0.00 | 0.00 | 14.79 | 0.00 | 0.00 | 14.79 | 0.00 | 0.00 |
| | **Avg.** | 18.05 | 31.86 | 1.49 | 19.02 | 28.62 | 3.17 | 18.21 | 39.00 | 2.22 |

average the total run time for MILP model is 13.8163 seconds, for ISTNN+2OPT heuristic algorithm is 0.0216 seconds and for the IINS+2OPT heuristic algorithm is 0.0454 seconds. Therefore, the run time for both heuristic algorithms is significantly less than the exact approach.

In Table 5.8, the total travel time, total time window violation, and total excess ride time of the MILP exact model, ISTNN+2OPT, and IINS+2OPT heuristic algorithms are compared. It can be seen that on average the total travel time, total time window violation, and total excess ride time in MILP model are 18.05, 31.86, and 1.49, for ISTNN+2OPT heuristic algorithm are 19.02, 328.62, and 3.17 and for the IINS+2OPT heuristic algorithm are 18.21, 39.00, and 2.22. Therefore, the total travel time is 5.09% higher and total excess ride time is 52.99% higher and the total time window violation is 10.16% lower in ISTNN+2OPT heuristic algorithm compared to the optimal solution. And, the total travel time is 0.88% higher and total excess ride time is 32.88% higher and the total time window violation is 18.31% higher in IINS+2OPT heuristic algorithm compared to the optimal solution.

This experiment shows that both heuristic approaches perform well, and the solutions are very close to the optimal solution. On average, the ISTNN+2OPT algorithm performs better than the IINS+2OPT algorithm with respect to the running time, but the IINS+2OPT algorithm works better than the ISTNN+2OPT algorithm with respect to the total cost. In fact, the run times of both heuristic algorithms is significantly faster than the run time of the exact approach.

The dial-a-ride problem is NP-hard and the exact approach cannot be used to solve practical size instances of this problem. These experiments show that, on small problems, the heuristic algorithms are able to produce high quality solutions in a very short time.

CHAPTER

# 6

# META-HEURISTIC APPROACHES FOR THE MULTI-VEHICLE DIAL-A-RIDE PROBLEM

## 6.1 Introduction

In practice, the number of the requests received for dial-a-ride service by transit agencies is large, and usually the requests are received from different parts of the city. Therefore, these requests must be satisfied by a fleet of vehicles.

A cluster-first, route-second approach is widely used in the literature to solve multi-vehicle dial-a-ride problems (Jorgensen et al. 2007; Baugh et al. 1998; Cubillos et al. 2009). In this approach, all the requests are clustered into a number of groups; then, a routing algorithm is used to schedule the pick-ups and drop-offs in each group. Meta-heuristic algorithms such as tabu search, genetic algorithms, and simulated annealing may be used to cluster the requests. In this research, a genetic algorithm (GA) is developed to cluster the requests, thereby creating a testbed for comparing the quality of the heuristic, single-vehicle routers described in prior chapters.

Genetic algorithms are a type of evolutionary computing, an approach inspired by Charles Darwin's theory of natural evolution. The process of natural selection starts with the selection of fittest individuals from a population. They produce offspring that inherit the characteristics of the parents and will be added to the next generation. If parents have better fitness, their offspring will be better than parents and have a better chance at surviving. This process keeps on iterating and at the end, a generation with the fittest individuals will be found.

A wide variety of optimizations problems have been addressed using genetic algorithms (Srinivas and Deb 1994; Grefenstette 1986; Ebrahimipour et al. 2011). The traveling salesman problem, vehicle routing problems, and dial-a-ride problems also can be solved by genetic algorithms (Grefenstette et al. 1985; Baker and Ayechew 2003; Jorgensen et al. 2007). The approach has been shown to produce good quality solutions for the dial-a-ride problem and is therefore chosen for this study.

In the section below, the genetic algorithm approach is described in detail.

## 6.2 Genetic Algorithms

It is discussed earlier that genetic algorithms are designed to simulate a biological process; therefore, most of the terms used for GAs are borrowed from biological concepts.

### 6.2.1 Basic Terminology

Commonly used terms include the following:

**Population:** A population is a set of individuals. Each individual is a possible solution to the problem one wishes to solve. It can also be defined as a set of chromosomes.

**Chromosome:** The term chromosome refers to a numerical value or values that represent a candidate solution to the problem that the genetic algorithm is trying to solve.

**Genes:** An individual is characterized by a set of parameters (variables) known as genes. Genes are joined into a string to form a chromosome (solution). In a genetic algorithm, the set of genes for an individual is represented using a string, in terms of an alphabet. Often, binary values are used (strings of 1s and 0s).

In the context of this study, a genetic algorithm is used to solve the multi-vehicle dial-a-ride problem with cluster-first, route-second method. Specifically, the GA is used for the clustering part. Therefore, the solution of the GA for this problem is a set of clusters. The number of clusters in a solution is equal to the number of available vehicles. Solutions are the chromosomes. Therefore, one chromosome contains a set of clusters, and one population contains a set of chromosomes.

A two-level binary matrix representation for chromosomes is introduced by Jorgensen et al. (2007). The same representation is used in this study. In this representation, the number of rows in this matrix is equal to the number of available vehicles, and the number of columns in this matrix is equal to to the number of customers.

Consider an example of a dial-a-ride problem with 10 customers and 4 available vehicles. One chromosome can be represented as Figure 6.1.

| | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 |
|---|---|---|---|---|---|---|---|---|---|---|
| V1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| V2 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| V3 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| V4 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |

Figure 6.1: An example of the two-level binary chromosome representation. Number of the rows is equal to the number of available vehicles and number of columns is equal to the number of customers. *Vi* indicate the $i$th vehicle, and *Ci* indicates the $i$th customer.

In this representation, each position $(v_i, c_j) \quad \forall i \in V, j \in C$, in which $V$ is the set of all available vehicles and $C$ is the set of all available customers, is a gene and it gets a binary value. The binary value at any position $(v_i, c_j)$ indicates that the customer represented by column $c_j$ is assigned to the vehicle represented by row $v_i$. Each customer should be allocated to one and only one vehicle in a solution (chromosome). Otherwise, the solution is not feasible. To verify that the solution is feasible, the sum of the entries in each column in the matrix chromosome representation must be equal to 1. With this representation, it is easy to verify that a solution is feasible or not.

Again, consider the previous example of a dial-a-ride problem with 10 customers and 4 available vehicles. One population can be represented as Figure 6.2. This population contains three chromosomes.

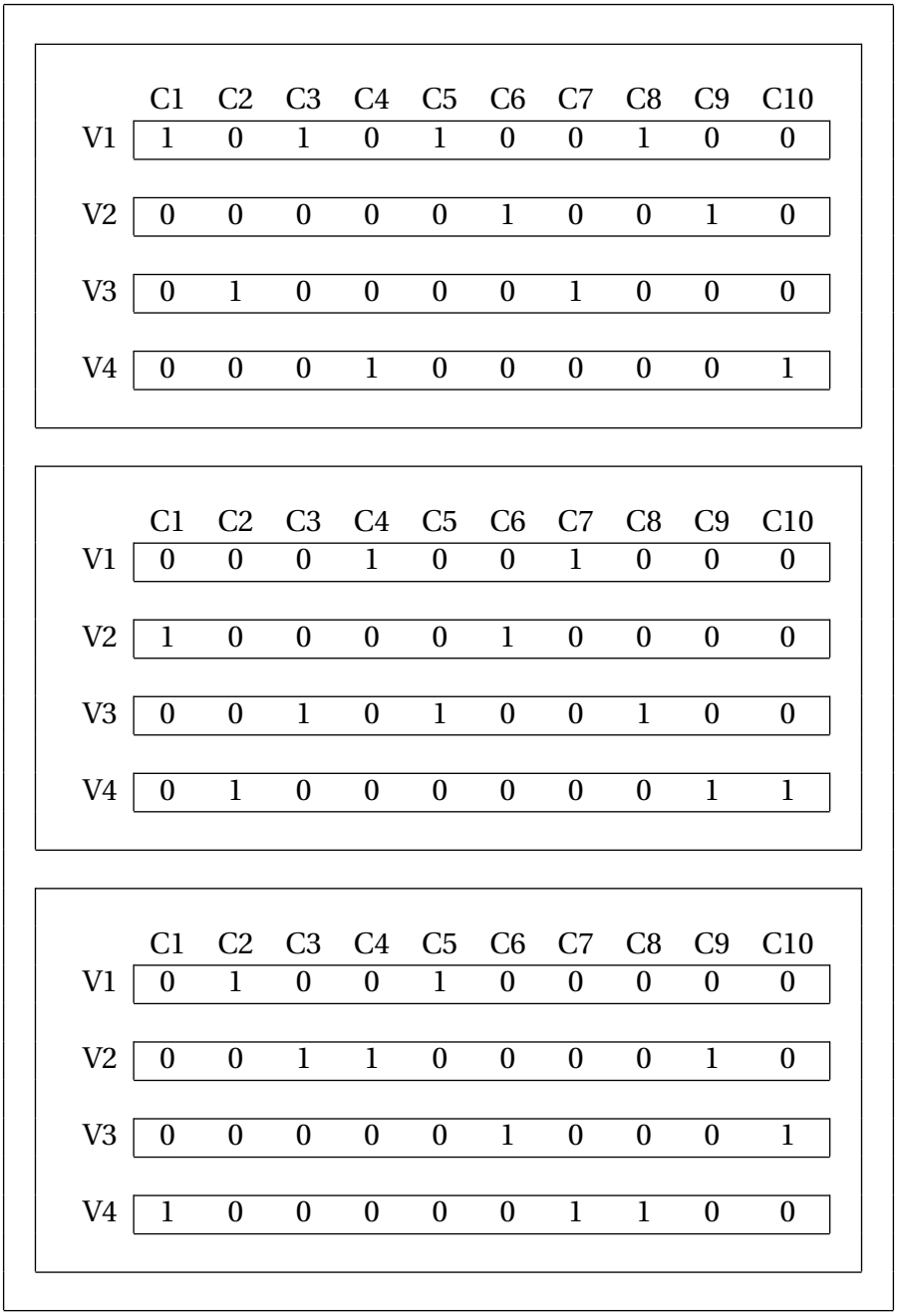


Figure 6.2: An example of a population containing three chromosomes.

### 6.2.2 Basic Structure

In this section, the basic structure of a genetic algorithm is described. Each GA has six components: initial population, fitness function, selection, crossover, mutation, and termination. A brief description of the computational process is as follows. A GA starts with an initial population, two parents from this population are selected, crossover and mutation operations are applied on the parents to create new offspring, and replace the existing individuals in the population with new offspring based on their fitness values. This process repeats until the termination criteria satisfies.

#### Initial Population

All the GAs start with an initial population, and then updates this population by replacing the current solutions with better solutions. There are two primary approaches to create the initial population in a GA. These methods are defined below:

- Random initial population: A set of random solutions created. The initial population contains this set of random solutions.
- Heuristic initial population: A heuristic algorithm is used to create a set of solutions. The initial population contains this set of solutions.

It is very important to have diversity in the initial population. Otherwise, it may cause the premature convergence. Therefore, it is not recommended to create all the initial population with a heuristic algorithm. the heuristic algorithm creates similar solutions with little diversity. Some researchers use a heuristic algorithm to seed the population with a few good quality solutions, then the rest of the solutions will be created using random method.

The size of the initial population also impacts the quality of the solution. If the initial population size is small, the quality of the solution would not be good. On the other hand, if a large initial population is chosen, the run time may be excessive. Therefore, it is often a trade-off between time and quality.

In this study, the random approach is used to create the initial population. Each solution in the initial population is a set of clusters of customers. Each solution in the initial population is created in a way to avoid infeasiblity. Therefore, all the solutions in the initial population are feasible, that means in each solution, each customer is assigned to one and only one cluster (vehicle).

The population size is decided by trial and error. It is large enough to generate good quality solutions, and it is not that large to take so much time for convergence.

**Fitness Function**

The fitness function is a function that gets a solution of the problem as an input and returns a value that measures how the solution is fit or good. The fitness value for the initial population and all the new offspring should be calculated. That means it should be calculated repeatedly in the GA. Therefore, the fitness function should be quick. Otherwise, it affects the GA and makes it slow.

In this study, the input for the fitness function is a set of clusters of customers. Within the fitness function a route should be generated for each cluster, and the function should return the total cost (objective function value) of the routes for the clusters.

One cluster is equivalent to one vehicle. The fitness function measures how it is good to put that customers in the same vehicle.

One of the heuristic algorithms described in Chapter 4 (STNN, ISTNN, ISTNN+2OPT, INS, IINS, or IINS+2OPT) can be used to generate the routes in the fitness function. The cost (objective function value) for each cluster is calculated based on the following function.

Consider the following terms:

$T_T$ : Total travel time

$TWV_1$ : The total amount of time the vehicle arrived early to drop-off the customers

$TWV_2$ : The total amount of time the vehicle arrived late to drop-off the customers on outbound requests

$TWV_3$ : The total amount of time the vehicle arrived late to drop-off the customers on inbound requests

$W_T$ : The total amount of time the vehicle arrived early to pick-up the customers

$R_T$ : Total excess ride time

$w_1$ : Weight on $T_T$

$w_2$ : Weight on $TWV_1$

$w_3$ : Weight on $TWV_2$

$w_4$ : Weight on $TWV_3$

$w_5$ : Weight on $W_T$

$w_6$ : Weight on $R_T$

The cost of each route in the fitness function can be calculated by the following formula:

$$\textit{Cost (Objective Function Value)} = w_1 * T_T + w_2 * TWV_1 + w_3 * TWV_2 + w_4 * TWV_3 + w_5 * W_T + w_6 * R_T$$

It includes six objectives so that the following are minimized: total travel time, total excess ride time, total waiting time, and total time window violation, and where time window violation is further separated into three terms to account for differences in relative importance.

**Selection**

In the selection phase of the GA, two parents (solutions) are selected to generate new offspring (new solutions). This phase is very important in the convergence of the GA. If just the best two solutions are selected as parents, the new offspring would not be diverse enough and it leads to premature convergence. Therefore, the parents should be selected in a way to make sure the diversity will be maintained in the population. It is also very important to generate fitter and better new solutions, so the new population is fitter than the previous population.

There are different methods to select two parents, like: Roulette Wheel Selection, Stochastic Universal Sampling, Tournament Selection, Rank Selection, and etc.

Roulette wheel selection and tournament selection are two popular selection methods for dial-a-ride problems. For example, Jorgensen et al. (2007) have used roulette wheel selection, and Masmoudi et al. (2017) and Cubillos et al. (2009) have used tournament selection for dial-a-ride problems.

In this study, tournament selection is chosen. Different types of the tournament selection are described in the literature. In the approach used in this study, each parent is selected by the following procedure:

1. Put $p = 0.95$ as a fixed value.
2. Select two random chromosomes from the population.
3. Generate a random number, $r$, in the $(0, 1)$ interval.
4. If $r \leq p$, choose the chromosome with better fitness value (lower objective function value or lower cost) as a parent. If $r > p$, choose the chromosome with worse fitness value (higher objective function value or higher cost) as a parent.

The steps 2-4 are repeated to select the second parent.

**Crossover**

In the crossover phase of a GA, two parents, selected from the selection phase, exchange their genes to create one or more new offspring. Crossover is an important phase in a GA. Because a percentage of the chromosomes in current population will be replaced by the new offspring generated in crossover phase. Usually, about 75% to 90% of the chromosomes (solutions) will be replaced with new chromosomes (new solutions).

There are different methods for the crossover, like: One point crossover, multi point crossover, uniform crossover, and etc.

The crossover process in a GA may produce infeasible solutions in dial-a-ride problems. Most of the works used the GA for dial-a-ride problems have allowed producing infeasible solutions by crossover process; however, they need to spend some time after the process to eliminate infeasibility (for example, see Jorgensen et al. (2007); Cubillos et al. (2009); Masmoudi et al. (2017)). However, to save time, in this study, generating infeasible solutions is not allowed in the crossover process.

In this study, uniform crossover is used to produce two offspring from two parents. In this method for each customer a random number in $(0,1)$ would be generated. If it is less than 0.5, that customer would be assigned to the vehicle based on the first parent in the first off-spring, and to the vehicle based on the second parent in the second off-spring. If it is greater than or equal to 0.5, that customer would be assigned to the vehicle based on the second parent in the first off-spring, and to the vehicle based on the first parent in the second off-spring.

This crossover method is described using a simple example. Figure 6.3 shows two parents. There are 3 vehicles and 5 customers in each parent (solution).

For each column a random number in $(0,1)$ interval is generated. If that is less than 0.5, that column is assigned to the first offspring from the first parent (in the left), and that

| | C1 | C2 | C3 | C4 | C5 |
|---|---|---|---|---|---|
| V1 | 1 | 0 | 1 | 0 | 0 |
| V2 | 0 | 0 | 0 | 1 | 0 |
| V3 | 0 | 1 | 0 | 0 | 1 |

| | C1 | C2 | C3 | C4 | C5 |
|---|---|---|---|---|---|
| V1 | 0 | 1 | 0 | 0 | 0 |
| V2 | 1 | 0 | 0 | 1 | 1 |
| V3 | 0 | 0 | 1 | 0 | 0 |

Figure 6.3: Example of two parents in the crossover procedure.

column is assigned to the other offspring from the other parent (in the right). If the random number is greater than or equal to 0.5, the reverse is applied.

In the example considered in Figure 6.3, there are 5 customers (column). For each of them a random number is generated as $0.4, 0.6, 0.9, 0.4, 0.6$. The new two offspring are plotted in Figure 6.4.

| | C1 | C2 | C3 | C4 | C5 |
|---|---|---|---|---|---|
| V1 | 1 | 1 | 0 | 0 | 0 |
| V2 | 0 | 0 | 0 | 1 | 1 |
| V3 | 0 | 0 | 1 | 0 | 0 |

| | C1 | C2 | C3 | C4 | C5 |
|---|---|---|---|---|---|
| V1 | 0 | 0 | 1 | 0 | 0 |
| V2 | 1 | 0 | 0 | 1 | 0 |
| V3 | 0 | 1 | 0 | 0 | 1 |

Figure 6.4: Example of two offspring produced in crossover procedure.

**Mutation**

In the mutation phase of a GA, a small random change or tweak is applied to a portion of chromosomes in the population. As a result, new chromosomes (solutions) will be obtained, and they replace the old chromosomes in the population. Usually, the mutation is applied to just a small portion of the population (usually less than 10%); otherwise, the GA will convert to a random search algorithm.

The mutation process allows the GA to explore the search space; therefore, it maintains the diversity in the population and avoids premature convergence. Mutation is very important in the convergence of the GA; on the other hand; crossover is not essential to the convergence of the GA.

There are different methods for mutation, like: Bit Flip Mutation, Swap Mutation, Scramble Mutation, Inversion Mutation, and etc.

In dial-a-ride problems there are two popular methods to perform the mutation process. In the first method, one or more customers may be removed from their current cluster and assigned to other clusters. In the second method, the position of a customer in a cluster may change (see for example Jorgensen et al. (2007); Cubillos et al. (2009); Masmoudi et al. (2017)). In this study, the second method cannot be used, because the GA is used for just clustering and the position of each customer in a cluster is decided by the routers.

The mutation process used in this study is described below. To the best of our knowledge, this mutation process is not used for dial-a-ride problems. It contains multiple implementations of the first mutation method described above.

1. Select one random solution (chromosome) in the population.
2. Select one random customer (random column in the solution).
3. Select one random vehicle (random row in the solution).
4. Move the random customer from its current vehicle to the new random vehicle.
5. Repeat steps 2, 3, and 4 for a random number of iterations.
6. Select two random customers (random columns) in the solution.
7. Exchange these two customers in that solution.
8. Repeat steps 6 and 7 for a random number of iterations.

If it is decided to apply the mutation $m$ times, this process should be repeated $m$ times.

This mutation method is described using a simple example. Consider an example with 5 customers and 3 vehicles. Assume the chromosome shown in Figure 6.5 in the population is randomly chosen. For simplicity, steps 5 and 8 are skipped.

| | C1 | C2 | C3 | C4 | C5 |
|---|---|---|---|---|---|
| V1 | 1 | 0 | 1 | 0 | 0 |
| V2 | 0 | 0 | 0 | 1 | 0 |
| V3 | 0 | 1 | 0 | 0 | 1 |

Figure 6.5: Example of one chromosome as an input for mutation procedure..

Assume the second customer and the second vehicle are selected randomly. In the new chromosome, the second customer will be removed from its current vehicle (third vehicle), and it will be assigned to the second vehicle. This new chromosome is represented in Figure 6.6.

| | C1 | C2 | C3 | C4 | C5 |
|---|---|---|---|---|---|
| V1 | 1 | 0 | 1 | 0 | 0 |
| V2 | 0 | 1 | 0 | 1 | 0 |
| V3 | 0 | 0 | 0 | 0 | 1 |

Figure 6.6: Example of one new chromosome produced after the 4th iteration of the mutation procedure.

Now, assume the first and fifth customers are selected randomly. These two customers will be exchanged to result the new solution. This solution is represented in Figure 6.6.

| | C1 | C2 | C3 | C4 | C5 |
|---|---|---|---|---|---|
| V1 | 0 | 0 | 1 | 0 | 1 |
| V2 | 0 | 1 | 0 | 1 | 0 |
| V3 | 1 | 0 | 0 | 0 | 0 |

Figure 6.7: Example of final chromosome produced by the mutation procedure.

**Termination**

The termination condition in a GA is important as it affects the practicality and effectiveness of the algorithm. If the GA terminates in early iterations, the solution would not be good enough. However, if it takes so much time to terminate, the algorithm would not be practical.

The termination condition for a GA is usually selected to be as one of the following conditions:

- Terminate the algorithm if there is no improvement for $n$ generations (iterations)
- Terminate the algorithm after a fixed number of generations (iterations)
- Terminate the algorithm if the fitness value is reached to a specific value

Most of the works used the GA for dial-a-ride problems have terminated the algorithm after a fixed number of generations (second condition), for example see Masmoudi et al. (2017); Cubillos et al. (2009). In this study, the first condition is used. Because, it it is important to terminate the algorithm when the GA has converged. If the algorithm terminates in a fixed number of generations, the convergence may not be achieved. Therefore, in this study, the GA would terminate if no improvement is seen for 20 iterations.

The figure 6.8 shows a flowchart representing the whole procedure of the GA used in this study. The Algorithm 9 also shows a pseudo-code of this procedure. The steps within

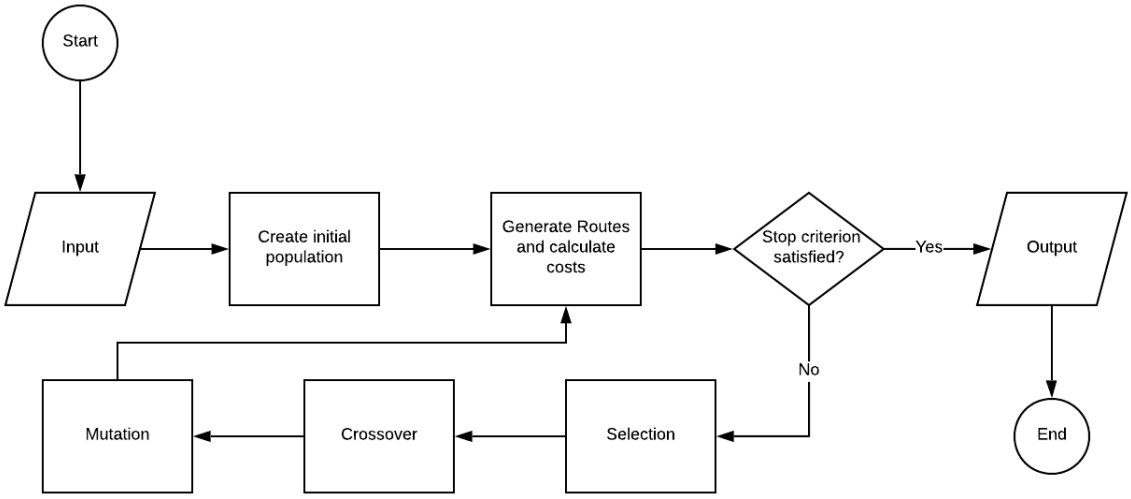


Figure 6.8: A flow chart describing the genetic algorithm.

the GA are explained in the previous sections. Just needs to mention that the input for the GA is:

- The location for depot
- The origin and destination locations for all requests
- The pick-up and drop-off time windows for all requests
- number of available vehicles
- The capacity of each vehicle

The output includes clusters of the customers with routing schedules for each cluster. The number of clusters should not be more than the number of available vehicles.

**Algorithm 9** The Genetic Algorithm.

```
1: procedure GA
2:     Generate the initial population
3:     for i ∈ {0, 1, 2, ..., N} do
4:         Select two parents from the current population
5:         Perform crossover
6:         Perform mutation
7:         Update the population
8:     return best chromosome, best cost
```

## 6.3 Post Optimization

It is mentioned that within the *Fitness Function*, a route should be generated for each cluster and the total cost is returned as a fitness value. Any of the heuristic algorithms described in Chapter 4 (STNN, ISTNN, ISTNN+2OPT,INS, IINS, or IINS+2OPT) can be used to generate the routes in the fitness function. It is also mentioned that the speed of the fitness function is very important and it should be quick.

In Chapter 5, it is seen that the fastest algorithm among the six algorithms, is the STNN heuristic algorithm. On the other hand, IINS+2OPT heuristic algorithms is the slowest.

The GA is used to solve the multi-vehicle dial-a-ride problem. The size of the problem is large, so it is not practical to use the slow algorithms in the fitness function. Therefore, in this study a fast heuristic algorithm is used within the fitness function of the GA to generate a route for each cluster and calculate the fitness value. To utilize slower algorithms in the context of multi-vehicle dial-a-ride problem, they are used within a local search algorithm to post optimize the solution of the GA. This local search algorithm is described in the Algorithm 10.

The local search described in Algorithm 10, gets a solution of the GA (clusters of customers) as an input. It uses the *Improve* function described in Algorithm 11 to improve the solution of the GA.

**Algorithm 10** A Local Search Algorithm.

```
1: procedure LOCAL SEARCH(S)
2:     Generate a route for the solution S
3:     S_cost = fitness_value(S)
4:     while True do
5:         tmp, tmp_cost = Improve(S, S_Cost)
6:         if tmp_cost < S_cost then
7:             S, S_cost = tmp, tmp_cost
8:         else
9:             return S, S_cost
```

The *Improve* procedure takes customers one by one and reallocates them to other clusters if the change can improve the current solution. To measure whether there is an improvement or not, for new clusters routes should be generated and the total cost of the new clusters is compared with the total cost of the current clusters. To generate the routes one of the heuristic algorithms introduced in Chapter 4 for single vehicle dial-a-ride problems can be used.

**Algorithm 11** The Improve procedure.

```
1: procedure IMPROVE(x, x_cost)
2:     for i ∈ {1, 2, ..., |C|} do
3:         for j ∈ {1, 2..., |V|} do
4:             Move the customer i from its current vehicle to vehicle j
5:             Generate a route for the new clusters and calculate the fitness value
6:             if If the fitness value of the new clusters is better then
7:                 return new clusters, new cost
8:     return x, x_cost
```

Therefore, a multi-vehicle dial-a-ride problem is solved using the following procedure:

1. A set of clusters of customers is generated using the GA. The pick-ups and drop-offs the customers in each cluster is scheduled by the fast heuristic algorithm as the router

within the GA.

2. A local search is performed to post optimize the clusters by exchanging the customers within the clusters. The pick-ups and drop-offs the customers in new clusters is scheduled by slower heuristic algorithms that produce better solutions as the router within the local search.

CHAPTER

# 7

# EXPERIMENTS WITH THE MULTI-VEHICLE DIAL-A-RIDE PROBLEM

In this chapter, the results of computational experiments with the genetic algorithm are presented. The objective of the experiments is to compare the quality of the heuristic approaches, the single vehicle routers, in the context of multi-vehicle dial-a-ride problem.

To this end, two instances (A and B) with different sizes are used. Instance A contains 24 customers with 3 available vehicles, and instance B contains 48 customers with 5 available vehicles. These instances used in the following experiments are obtained from data sets created by Cordeau and Laporte (2003). These data sets are available online.[1]

[1] Cordeau and Laporte data sets can be found at https://chairelogistique.hec.ca/en/scientific-data/

All algorithms are implemented in Python. All experiments are performed on a 3.4 GHz Core i7 computer with 16 GB memory.

## 7.1 Parameters

The genetic algorithm used in this study has various parameters that are mentioned below:

- Population size
- termination condition (number of iterations without improvement)
- Crossover probability and mutation probability
- Mutation parameters (parameters in the fifth step and eight step of the mutation process)

In a GA, the usual value for crossover probability is about 75 % to 90%, and the mutation probability is less that or equal to 10 %. In this chapter, the crossover probability is set to be 80 % and the mutation probability is 10 %. The experiments performed on these two parameters showed that the change in these values does not affect the performance of the GA significantly. The values for mutation parameters are obtained by trial and error for each instance. For instance A, these parameters are 5 and 2, and for instance B, these parameters are 10 and 5. Moreover, if no improvement has been seen within 20 iterations that means the GA has converged and there is no need to run for more iterations.

The population size is very important and the quality of the solution and convergence of the algorithm depends on this parameter. Therefore a sensitivity analysis has performed on the population size for both instance A and instance B.

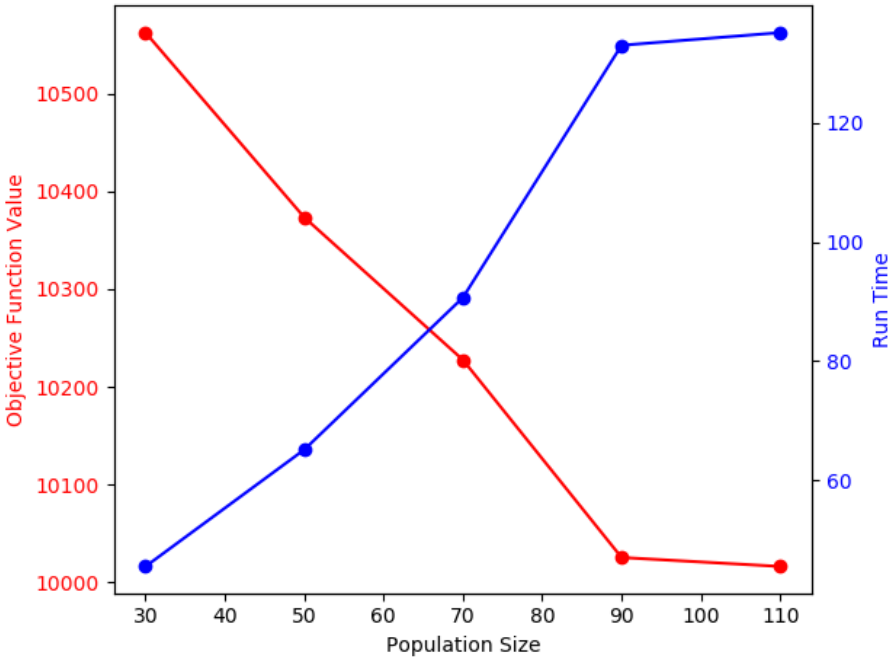


Figure 7.1: The results obtained by the GA with STNN heuristic algorithm for instance A with different population sizes.

**Instance A**

The GA with STNN heuristic algorithm as the router is solved for different population sizes for instance A. Each run is repeated 10 times and the average results are reported in Table 7.1 and in Figure 7.1.

Instance A contains 24 customers. The population sizes considered for this experiment are 30, 50, 70, 90, and 110. From Table 7.1 and figure 7.1 it can be seen that the objective function value gets better with an increase in the population size. The run time also increases with increase in the population size. Therefore, there is a trade-off between quality and run time. In this chapter the population size of 50 is considered for instance A.

Table 7.1: The results obtained by the GA with STNN heuristic algorithm for instance A with different population sizes.

| Pop size | Run Time | Objective Function Value |
|---|---|---|
| 30 | 45.5 s | 10562.415 |
| 50 | 1 min 5.1 s | 10373.454 |
| 70 | 1 min 30.6 s | 10227.640 |
| 90 | 2 min 13 s | 10025.426 |
| 110 | 2 min 15.1 s | 10016.264 |

**Instance B**

The GA with STNN heuristic algorithm as the router is solved for different population sizes for instance B. Each run is repeated 10 times and the results are reported in Table 7.2 and Figure 7.2.

Table 7.2: The results obtained by the GA with STNN heuristic algorithm for instance B with different population sizes.

| Pop size | Run Time | Objective Function Value |
|---|---|---|
| 50 | 3 min 1.3 s | 20719.945 |
| 70 | 5 min 24.5 s | 19921.045 |
| 100 | 6 min 57.2 s | 19236.980 |
| 120 | 9 min 51 s | 19112.020 |
| 140 | 12 min 8.1 s | 18769.033 |

Instance B contains 48 customers. The population sizes considered for this experiment are 50, 70, 100, 120, and 150. Similar to the previous instance, from Table 7.2 and Figure 7.2 it can be seen that the objective function value gets better with increase in the population size. The run time also increases with increase in the population size. In this chapter the population size of 100 is considered for instance B.

It should be mentioned that the initial population for each experiment and for each run of the algorithm is generated randomly and independently.

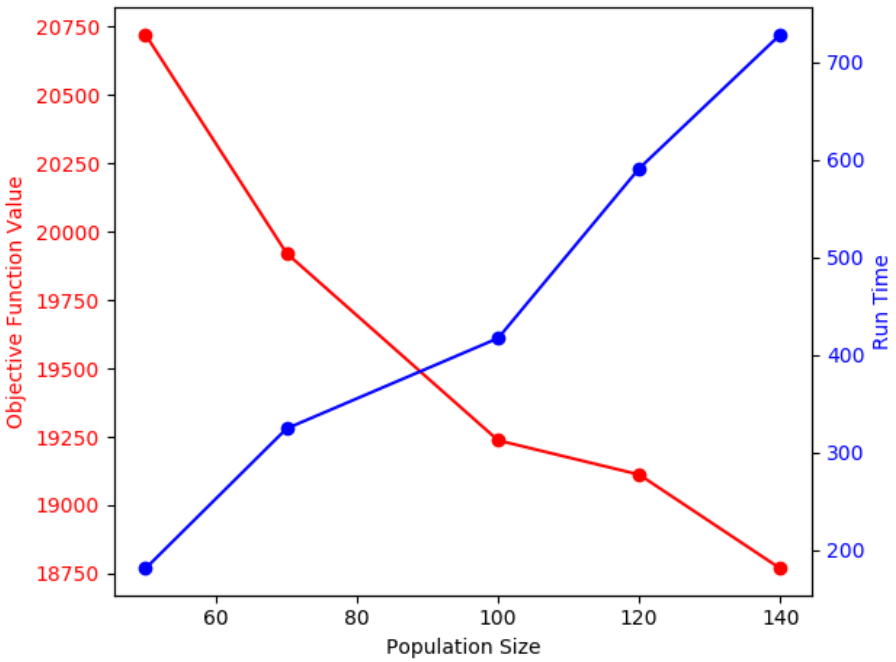


Figure 7.2: The results obtained by the GA with STNN heuristic algorithm for instance B with different population sizes.

## 7.2 Experiment 1: Comparison of Routers, Fixed Run Time

It is discussed earlier that among the six single vehicle routers, the STNN heuristic algorithm is the fastest algorithm. Other 5 algorithms are slower than the STNN heuristic algorithm; however, they can produce better solutions. In this section, the GA is run with each router (STNN, ISTNN, ISTNN+2OPT, INS, IINS, IINS+2OPT) separately and compared to evaluate the quality of the solutions obtained by different routers. For this purpose, the GA is terminated after a fixed amount of run time for both instance A and instance B. The results for each instance are summarized in the following two sections.

**Instance A**

For instance A, the GA is run with each router (STNN, ISTNN, ISTNN+2OPT, INS, IINS, IINS+2OPT) separately and compared to evaluate the quality of the solutions obtained by different routers. The algorithm is terminated after 1 hour and the results for 30 minutes and 1 hour run are reported in Table 7.3 and in Figure 7.3.

Table 7.3: The results obtained by 1 hour run of the GA with different routers for instance A (No. Gen. represents the number of generations and OFV represents the objective function value).

| Routing Method | No. Gen., 30 mins | OFV, 30 mins | No. Gen., 1 hr | OFV, 1 hr |
|---|---|---|---|---|
| STNN | 1409 | 10215.550 | 2848 | 10215.550 |
| ISTNN | 538 | 8405.038 | 1079 | 8405.038 |
| ISTNN+2OPT | 173 | 8114.503 | 338 | 8114.503 |
| INS | 269 | 8642.455 | 513 | 8325.305 |
| IINS | 81 | 7631.099 | 148 | 7631.099 |
| IINS+2OPT | 74 | 7626.292 | 147 | 7626.292 |

In Figure 7.3, the convergence of the GA with different routers is plotted. The solid lines are the results for the first 30 minutes of run and the dashed lines are the results for the next

30 minutes of run. For the STNN heuristic algorithm the results for the second 30 minutes are not plotted, because the algorithm has converged in early iterations.

This figure and Table 7.3 show that the GA with STNN heuristic algorithm is very fast. The algorithm has iterated for 1409 generations in 30 minutes and for 2848 generations in 1 hour. However, from Figure 7.3 it can be seen that this algorithm has converged in early iterations, and the objective function value is not very good compared to the GA with other routers. The best solution has obtained from the GA with IINS+2OPT heuristic algorithm within 147 iterations. However, 147 iterations in 1 hour indicates that the router is slow.

**Instance B**

For instance B, the GA is run with each router (STNN, ISTNN, ISTNN+2OPT, INS, IINS, IINS+2OPT) separately and compared to evaluate the quality of the solutions obtained by different routers. The algorithm is terminated after 2 hours and the results are reported in Table 7.4 and in Figure 7.4.

Table 7.4: The results obtained by 2 hours run of the GA with different routers for instance B (No. Gen. represents the number of generations and OFV represents the objective function value).

| **Routing Method** | **No. Gen., 1hr** | **OFV, 1 hr** | **No. Gen., 2 hrs** | **OFV, 2 hrs** |
|---|---|---|---|---|
| STNN | 735 | 18341.969 | 1475 | 18341.969 |
| ISTNN | 247 | 14291.569 | 502 | 14002.006 |
| ISTNN+2OPT | 102 | 13987.127 | 199 | 13940.675 |
| INS | 40 | 15175.511 | 64 | 14892.033 |
| IINS | 29 | 13672.310 | 53 | 12444.268 |
| IINS+2OPT | 22 | 14678.180 | 39 | 12775.022 |

In Figure 7.3, the convergence of the GA with different routers is plotted. The solid lines are the results for the first 1 hour of run and the dashed lines are the results for the next

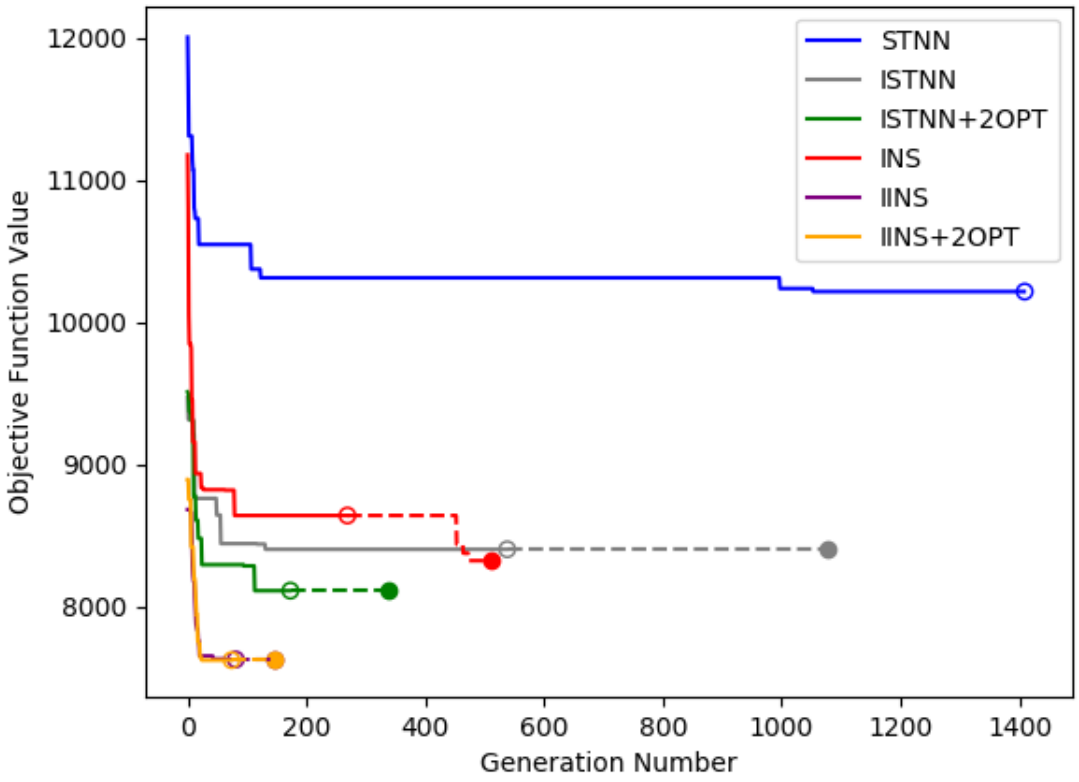


Figure 7.3: The results obtained by 1 hour run of the GA with different routers for instance A. The solid lines are the results for the first 30 minutes of run, and the dashed lines represent the results for the second 30 minutes of run. The dashed line for the STNN heuristic algorithm is not plotted as it has converged in early iterations.

1 hour of run. For the STNN heuristic algorithm the results for the second 1 hour are not plotted, because the algorithm has converged in early iterations.

This figure and Table 7.4 show that the GA with STNN heuristic algorithm is so fast. The algorithm has iterated for 735 generations in 1 hour and for 1475 generations in 2 hours. However, from Figure 7.4 it can be seen that this algorithm has converged in early iterations, and the objective function value is not very good compared to the GA with other routers. The best solution has obtained from the GA with IINS heuristic algorithm within 53 iterations. 53 iterations in 2 hours indicates that the router is slow. However, it not the slowest router. The results show that the IINS+2OPT is slower. In instance B the results show that the GA with both INS and IINS have not convereged yet. They need to iterate for more generations to converge.

## 7.3 Experiment 2: Comparison of Routers, Local Search

In this section, the procedure explained in the previous chapter is tested. Two instances are solved by the GA in which a fast heuristic algorithm is used to generate the routes in the fitness function. Then the solution is improved by the local search algorithm with slower routers (these routers produce better solutions). The results for each instance are summarized in the following two sections.

#### Instance A

In this section, the results obtained for instance A are summarized. The number of population considered for this instance is 50, and the parameters in the fifth step and eight step of the mutation are 5 and 2, respectively. The crossover probability is 80% and the mutation probability is 10%.

The Table 7.5 shows the results for instance A. Different routers within the fitness

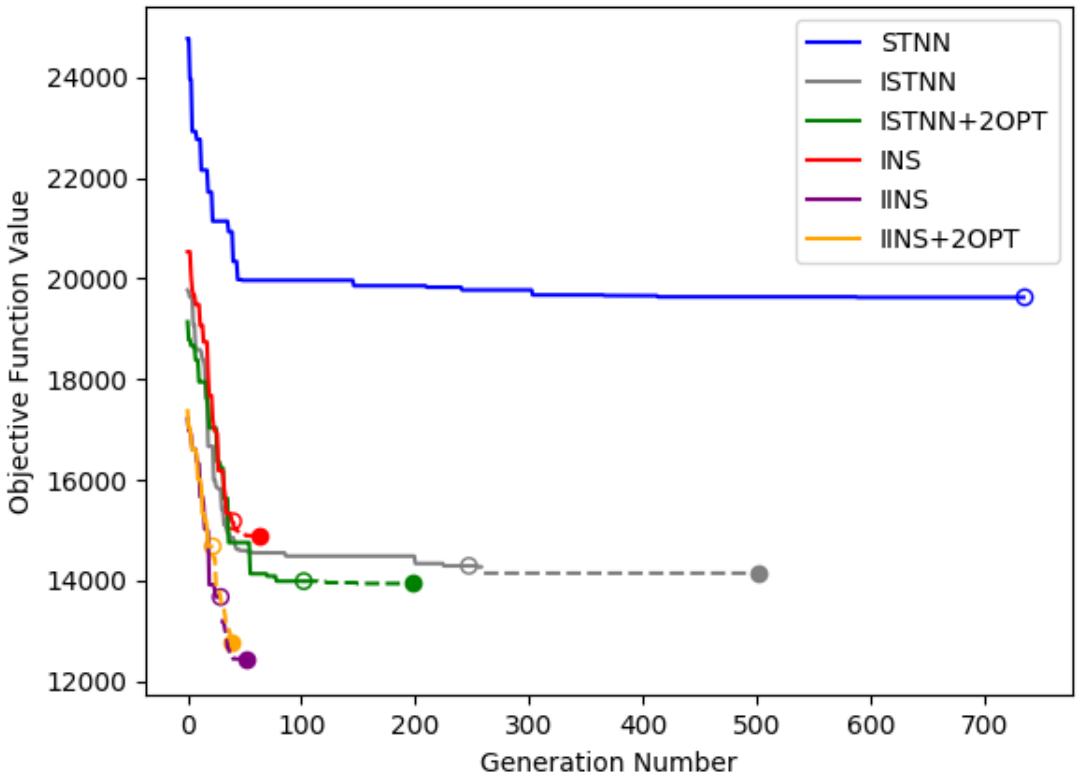


Figure 7.4: The results obtained by 2 hours run of the GA with different routers for instance B. The solid lines are the results for the first 1 hour of run, and the dashed lines represent the results for the second 1 hour of run. The dashed line for the STNN heuristic algorithm is not plotted as it has converged in early iterations.

function of the GA and the local search are used.

The GA solved with STNN and ISTNN as the router are improved by local search with ISTNN+2OPT, and the GA solved with INS and IINS as the router are improved by the local search with IINS+2OPT. The GA with ISTNN+2OPT and IINS+2OPT are not improved with local searches. The results show that using a fast router within the GA and using a slower router (which produces better solutions) within the local search is a good approach to solve the problem. For example, if the ISTNN+2OPT heuristic algorithm is used within the GA, the objective function value would be 8361.161, and this solution can be obtained in about 8 minutes. However, the GA with ISTNN, then local search with ISTNN+2OPT can reach to a slightly better solution is less time, 3 minutes. Another conclusion that can be obtained from this table is that the GA with IINS as the router, then a local search with IINS+2OPT as the router produces the best solution, however it takes so much to get to this solution. Therefore, there is a trade-off between time and cost.

Table 7.5: The results obtained by applying the GA with different routers and improvements in the GA solution using different routers within the local search for instance A.

| **Router (GA)** | **OFV** | **Run Time** | **Router (LS)** | **OFV** | **Run Time** |
|---|---|---|---|---|---|
| STNN | 10297.132 | 1 min 10 s | ISTNN+2OPT | 8483.014 | 38.9 s |
| ISTNN | 8553.910 | 2 min 19 s | ISTNN+2OPT | 8347.574 | 18.7 s |
| ISTNN+2OPT | 8361.161 | 8 min 11 s | - | - | - |
| INS | 8755.437 | 5 min 22 s | IINS+2OPT | 7613.552 | 1 min 16 s |
| IINS | 7281.011 | 13 min 18 s | IINS+2OPT | 7192.966 | 2 min |
| IINS+2OPT | 7252.427 | 17 min 11 s | - | - | - |

The Figure 7.5 represents the results obtained for different runs of the GA with a fast heuristic algorithm as the router, and the results of the local search with slower routers for instance A. Each line in this figure represents the convergence of the GA with different routers. The circles below each line represents the best objective function value obtained

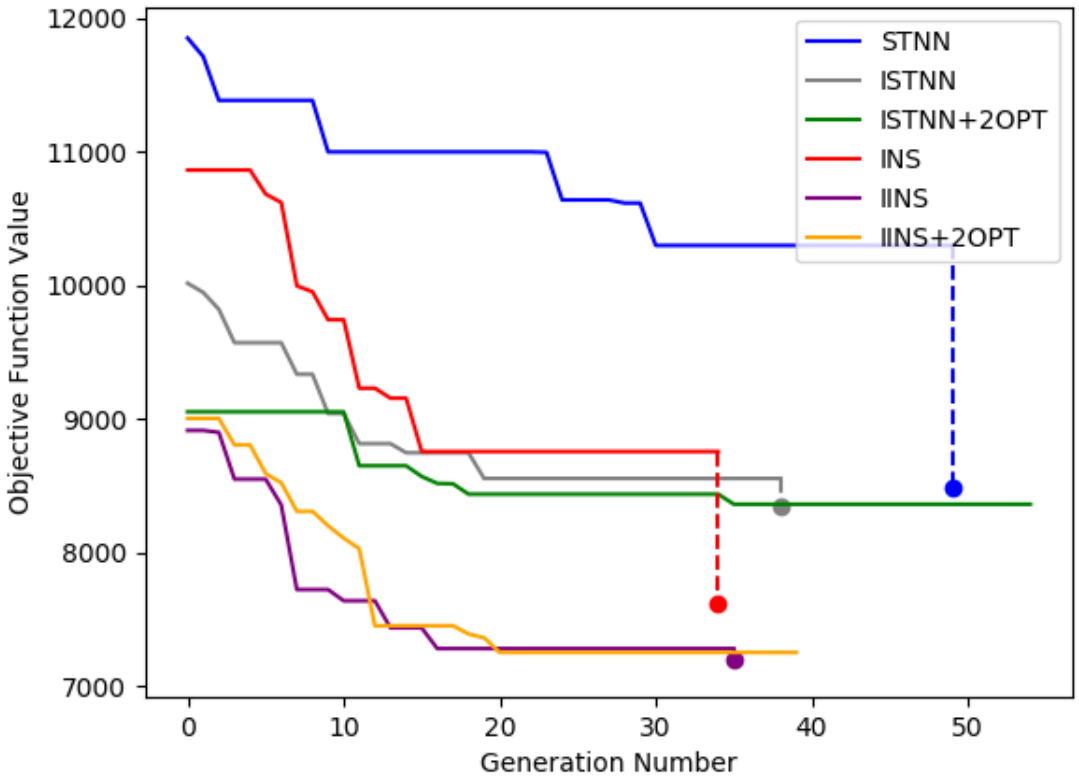


Figure 7.5: The results obtained by different runs of the GA with a fast heuristic algorithm as a router, and the results of local search with slower routers for instance A.

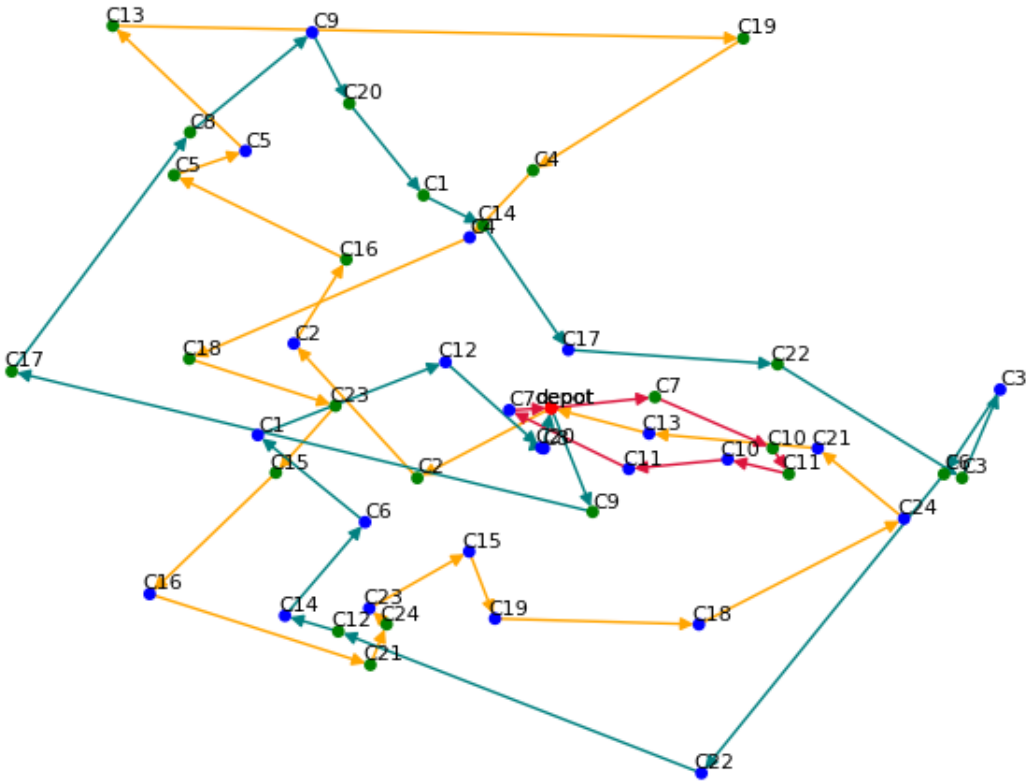


Figure 7.6: The network representing the best solution obtained for instance A by experiment 2.

by the local search algorithm with better routers. From this figure, it can be seen that the local search algorithm with better routers is helpful to improve the solutions obtained by the GA.

From Figure 7.5 it can be seen that the best solution has been obtained from the GA with IINS as the router and the local search with IINS+2OPT heuristic algorithm. The objective function value for this solution is 7192.966. The solution corresponds to this objective function value is plotted in Figure 7.6. This figure shows that in this solution, the 24 customers are allocated to 3 vehicles . 11 customers are assigned to the first vehicle, 3

customers are assigned to the second customer, and 10 customers are assigned to the third vehicle. This figure shows the schedules for pick-ups and drop-offs for each vehicle.

**Instance B**

In this section, the results obtained for instance B are summarized. The number of population considered for this instance is 100, and the parameters in the fifth step and eight step of the mutation are 10 and 5, respectively. The crossover probability is 80% and the mutation probability is 10%.

The Table 7.6 shows the results for instance B. In each row of this table, the results for different routers within the GA and within the local search are provided.

The GA solved with STNN and ISTNN as the router are improved by local search with ISTNN+2OPT, and the GA solved with INS and IINS as the router are improved by the local search with IINS+2OPT. The GA with ISTNN+2OPT and IINS+2OPT are not improved with local searches. Similar to instance A, the results show that using a fast router within the GA and using a slower router (which produces better solutions) within the local search is a good approach to solve the problem. For example, if the ISTNN+2OPT heuristic algorithm is used within the GA, the objective function value would be 14249, and this solution can be obtained in about 42 minutes. However, the GA with ISTNN, then the local search with ISTNN+2OPT can reach to a slightly better solution is less time, 28 minutes. Another conclusion that can be obtained from this table is that the GA with IINS as the router, then a local search with IINS+2OPT as the router produces the best solution, however it takes so much to get to this solution. Therefore, there is a trade-off between time and cost.

The Figure 7.7 represents the results obtained for different runs of the GA with a fast heuristic algorithm as the router, and the results of local search with slower routers for instance B. Each line in this figure represents the convergence of the GA with different routers. The circles below each line represents the best objective function value obtained

Table 7.6: The results obtained by applying the GA with different routers and improvements in the GA solution using different routers within the local search for instance B.

| **Router (GA)** | **OFV** | **Run Time** | **Router (LS)** | **OFV** | **Run Time** |
|---|---|---|---|---|---|
| STNN | 19240.303 | 6 min 52 s | ISTNN+2OPT | 15172.284 | 4 min 30 s |
| ISTNN | 15796.950 | 17 min 23 s | ISTNN+2OPT | 13293.377 | 10 min 13 s |
| ISTNN+2OPT | 14249.000 | 41 min 34 s | - | - | - |
| INS | 15842.272 | 59 min 40 s | IINS+2OPT | 12435.562 | 24 min 48 s |
| IINS | 12795.080 | 3 hr 45 min 36 s | IINS+2OPT | 11259.984 | 28 min 48 s |
| IINS+2OPT | 12480.906 | 4 hr 24 min 45 s | - | - | - |

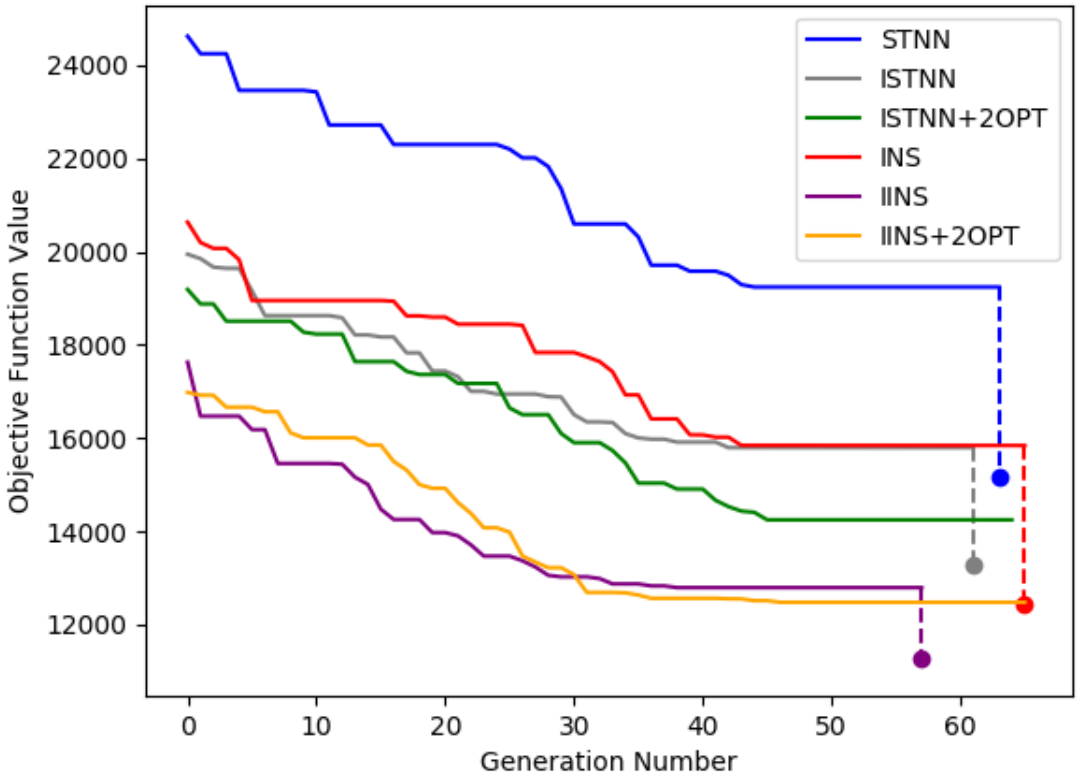


Figure 7.7: The results obtained for three runs of the GA with STNN heuristic algorithm as a router, and the results of local search with six different routers (STNN, ISTNN, ISTNN+2OPT, INS, IINS, IINS+2OPT) for instance B.

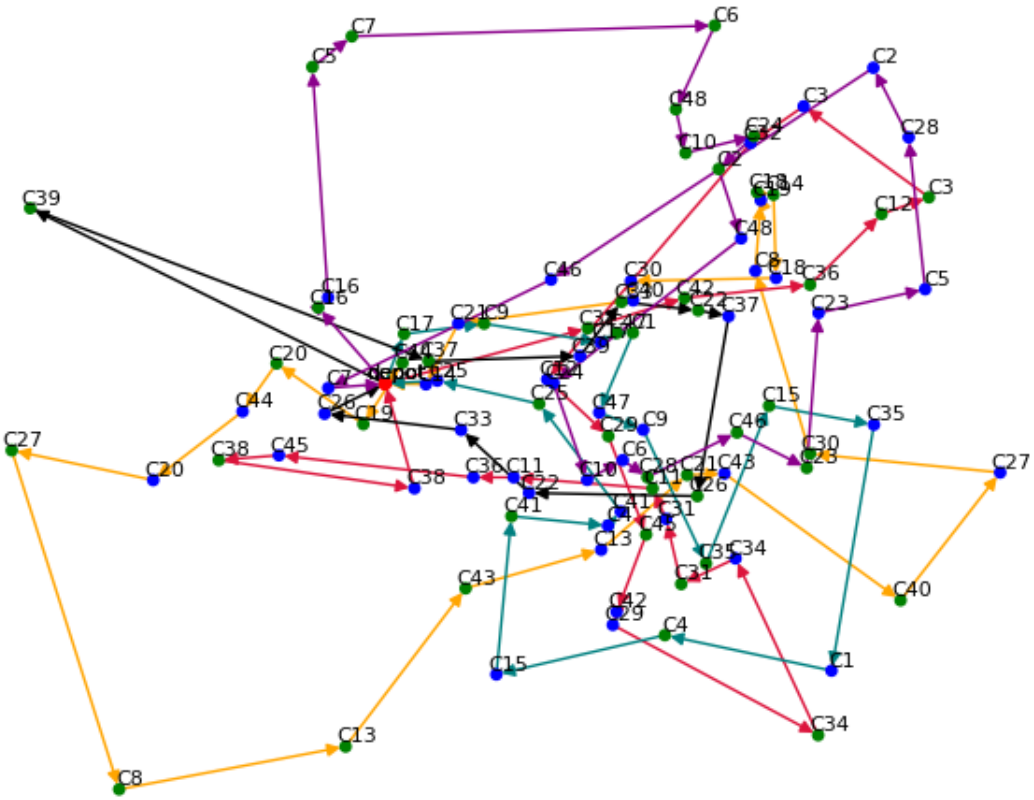


Figure 7.8: The network representing the best solution obtained for instance B by experiment 2.

by the local search algorithm with better routers. From this figure, Similar to instance A, from this figure, it can be seen that the local search algorithm with better routers is helpful to improve the solutions obtained by the GA.

From Figure 7.7 it can be seen that the best solution has been obtained from the GA with IINS and the local search with IINS+2OPT heuristic algorithm. The objective function value for this solution is 11259.984. The solution corresponds to this fitness value is plotted in Figure 7.8. This figure shows that in this solution, the 48 customers are allocated to 5 vehicles. 12 customers have assigned to the first vehicle, 11 customers assigned to the

second vehicle, 19 customers are assigned to third vehicle, 5 customers assigned to fourth vehicle, and 11 customers are assigned to the fifth vehicle. This figure shows the schedules for pick-ups and drop-offs for each vehicle.

## 7.4 Experiment 3: Comparison of Routers, Fixed Objective

In this section, the six heuristic algorithms, single vehicle routers, are compared in another way. The termination condition in the GA changed. The GA is terminated if the fitness value is reached to a specific value. The GA is run with each router (STNN, ISTNN, ISTNN+2OPT, INS, IINS, IINS+2OPT) separately. The purpose of this experiment is to evaluate the routers within the GA. It is done for both instance A and instance B. The results for each instance are summarized in the following two sections.

**Instance A**

In this instance, the GA with new termination condition and with each router is run for instance A. Each run is repeated three times and the average results are reported. The GA has stopped when the objective function value is reached to 10228. The results are summarized in Table 7.7.

In Table 7.7, each row represents the results for the GA with each of the routers. The table shows the GA with STNN heuristic algorithm does not perform well in respect to the number of the generations. It takes 35.99 generations for the algorithm to reach to the specific objective function value. On the other hand, the GA with ISTNN, ISTNN+2OPT, IINS, and IINS+2OPT has reached to that value in the first generation. However, the run times show that one generation of the GA with IINS and IINS+2OPT takes much time compared to other approaches. In this experiment, the GA with ISTNN performs well in both run time and number of generations. It has reached to the specific value in just one

iteration and in about 3.6 seconds.

Table 7.7: The results obtained by applying the GA with different routers for instance A. The GA is terminated once the objective function value is reached to 10228.

| Routing Method | Average Run Time | Average Number of Generations |
|---|---|---|
| STNN | 49.6 s | 35.66 |
| ISTNN | 3.6 s | 1 |
| ISTNN+2OPT | 7.7 s | 1 |
| INS | 73.5 s | 8.33 |
| IINS | 20.9 s | 1 |
| IINS+2OPT | 22.9 s | 1 |

**Instance B**

In this instance, the GA with new termination condition and with each router is run for instance B. Each run is repeated three times and the average results are reported. The GA has stopped when the objective function value is reached to 19800. The results are summarized in Table 7.8.

In Table 7.8, each row represents the results for the GA with each of the routers. The table shows the GA with STNN heuristic algorithm does not perform well in respect to the number of generations. It takes 55.33 iterations for the algorithm to reach to the specific objective function value. On the other hand, the GA with ISTNN+2OPT, and IINS, and IINS+2OPT has reached to that value in the first generation. However, the run times show that one generation of the GA with IINS and IINS+2OPT takes about so much time. In this experiment, the GA with ISTNN performs well in both run time and number of generations. It has reached to the specific value in about one iteration and in about 22 seconds.

Table 7.8: The results obtained by applying the GA with different routers for instance B. The GA is terminated once the objective function value is reached to 19800.

| **Routing Method** | **Average Run Time** | **Average Number of Generations** |
|---|---|---|
| STNN | 6 min 11 s | 55.33 |
| ISTNN | 22.4 s | 1.33 |
| ISTNN+2OPT | 38.7 s | 1 |
| INS | 3 min 59 s s | 4.33 |
| IINS | 2 min 14 s | 1 |
| IINS+2OPT | 2 min 23 s | 1 |

CHAPTER

# 8

# CONCLUSION

The dial-a-ride problem is a subclass of vehicle routing problems with pick-up and delivery, where customers make pick-up and drop-off requests that are serviced by a transportation center. The application of the problem mostly arises in door-to-door transportation for elderly or disabled people.

The main contribution of this dissertation is redefining the dial-a-ride problem. It addresses customer satisfaction factors in new ways that may be considered more practical from the perspective of the customer, overcoming some of the limitations of prior approaches.

A mixed integer linear programming model is introduced for the dial-a-ride problem with new definition. In this MILP model, multiple objective functions to minimize total

traveling time, total time window violation, and total excess ride are considered. Weighted sum method is used to make a single objective function. In this model, soft time windows are considered. The reason is that in many instances the hard time windows make the problem infeasible. Even if the problem is feasible, it may cause an increase in total traveling time that impacts the service cost. Customers may prefer a few minutes violation from the determined time windows, but pay less for the service. The time window violations are minimized in the objective function to avoid huge violations from time windows. Another aspect that differentiates this model from other models is that different weights for different types of time window violation are considered.

Space-time nearest neighbor heuristic algorithm is first designed to create routes in a cluster-first route-second approach to solve single vehicle dial-a-ride problems. This algorithm is a very fast algorithm to generate routes and schedule pick-ups and drop-offs in single vehicle dial-a-ride problems. This algorithm is used multiple times in the literature. However, it is not well defined. In this dissertation, a comprehensive description of this algorithm is provided. Insertion heuristic algorithm is another algorithm that used extensively in the literature for different types of dial-a-ride problems. An insertion heuristic algorithm is defined for the single vehicle dial-a-ride problem studied in this dissertation. As another contribution in this dissertation, these two well known algorithms improved and modified to solve the single vehicle dial-a-ride problem with new definition.

These algorithms extensively tested on a set of instances obtained from the Winston Salem Transit Authority. These heuristic algorithms compared with exact approach on small size instances. The computational results show that both space-time nearest neighbor heuristic algorithm and variants and insertion heuristic algorithm and variants produce high quality solutions (very close to the optimal solution) in short time for small size instances. In larger size instances these algorithm compared together. The computational results show that the space-time nearest neighbor heuristic algorithm and variants are

faster; on the other hand, insertion heuristic algorithm and variants produce better solutions. The STNN heuristic algorithm is the fastest algorithm, and the second fast algorithm is about 2.6 times slower than the STNN heuristic algorithm. However, the quality of the solutions produced by this approach on average is about 33% worse than the quality of the solutions produced by the IINS+2OPT heuristic algorithm. The IINS+2OPT heuristic algorithm produces the best solutions. However, it is very slow. On average, it is about 38 times slower than the STNN heuristic algorithm.

To solve the multi-vehicle dial-a-ride problem, a cluster-first, route-second approach is introduced. A genetic algorithm is designed for the clustering, and one of the heuristic algorithms introduced to solve single vehicle dial-a-ride problem can be used for the routing for each cluster. Moreover, a local search algorithm introduced that can be used to post optimize the solution of the GA. The local search algorithm gets the solution of the GA (clusters of the customers) as an input and exchange the customers between clusters to get better solutions. To measure whether there is an improvement or not, for new clusters, routes should be generated and the total cost of the new clusters would be compared with the total cost of the current clusters. To generate the routes one of the single vehicle routers can be used. The main purpose is to compare the quality of the heuristic, single-vehicle routers developed.

Extensive and different experiments performed to compare the single vehicle routers in the context of the multi-vehicle dial-a-ride problem. The experiments are performed on a set of datasets created by Cordeau and Laporte. The results show that the GA with STNN heuristic algorithm as the router is so fast, and for an instance with 48 customers, the algorithm converges in less than 7 minutes. However, for the same instance, the run time of one generation of the GA with IINS+2OPT heuristic algorithm takes 22 times more than one generation of the GA with STNN heuristic algorithm. Therefore, for large size instances it would not be practical to use the IINS+2OPT heuristic algorithm within the GA. Another

observation was that the GA with STNN heuristic algorithm as the router could not produce good quality solutions compared to other routers. For example, after 2 hours run of the GA with different routers for the instance with 48 customers, the solution obtained by the GA with STNN heuristic algorithm as the router was about 32% worse than the best solution obtained by the GA with other routers. Therefore, to utilize both the speed and the quality, the solution of the GA with a fast heuristic algorithm as the router post optimized by a local search with better but slower routers. The results show that the post optimization with better routers could improve the quality of the solution obtained by GA with fast heuristic algorithm as the router significantly within a reasonable amount of time. For example, for the instance with 48 customers, the GA with ISTNN heuristic algorithm, then the local search with ISTNN+2OPT heuristic algorithm could reach to the objevtive function value of 13293.377 in about 27 minutes. However, the GA with ISTNN+2OPT heuristic algorithm as the router would reach to this value in about 2 hours.

Therefore, it can be concluded that it is better to solve the GA with a fast heuristic algorithm as the router; then, improve and post optimize the solution by the local search with better but slower routers.

## 8.1 Future Research

- One observation in this study was that the space-time nearest neighbor heuristic algorithm and variants are faster than the insertion heuristic algorithm and variants; however, the quality of the solutions were worse. One possible future direction for the research is to use other fast neighborhood search heuristics with better solutions.

- In this study two data sets are used for experimental results. The WSTA data set is for 1995. The data set created by Cordeau and Laporte it is the only public data set exists

for dial-a-ride problem. However, it is randomly generated, and the time window for either origin or destination of each request does not exist. Therefore, one other possible future direction would be to create or use more realistic data sets to test the models and algorithms.

- More realistic and practical assumptions might be considered for future research. For example, in this study, all the vehicles are considered to be homogeneous that might not be a very realistic assumption. Heterogeneous vehicles might be considered for future research.

# REFERENCES


Baker, B. M. and Ayechew, M. (2003). A genetic algorithm for the vehicle routing problem. *Computers & Operations Research*, 30(5):787–800.

Baugh, J. W., Kakivaya, G. K. R., and Stone, J. R. (1998). Intractability of the dial-a-ride problem and a multiobjective solution using simulated annealing. *Engineering Optimization*, 30(2):91–123.

Campbell, A. M. and Savelsbergh, M. (2004). Efficient insertion heuristics for vehicle routing and scheduling problems. *Transportation science*, 38(3):369–378.

Cordeau, J.-F. (2006). A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, 54(3):573–586.

Cordeau, J.-F. and Laporte, G. (2003). A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B: Methodological*, 37(6):579–594.

Cordeau, J.-F. and Laporte, G. (2007). The dial-a-ride problem: models and algorithms. *Annals of operations research*, 153(1):29–46.

Cubillos, C., Urra, E., and Rodríguez, N. (2009). Application of genetic algorithms for the darptw problem. *International Journal of Computers Communications & Control*, 4(2):127–136.

Desrosiers, J., Dumas, Y., and Soumis, F. (1986). A dynamic programming solution of the large-scale single-vehicle dial-a-ride problem with time windows. *American Journal of Mathematical and Management Sciences*, 6(3-4):301–325.

Diana, M. and Dessouky, M. M. (2004). A new regret insertion heuristic for solving large-scale dial-a-ride problems with time windows. *Transportation Research Part B: Methodological*, 38(6):539–557.

Ebrahimipour, V., Tohidi, A., and Rahmanniya, F. (2011). Simultaneously optimizing of reliability, cost and completion time in series-parallel systems considering redundant allocation using a genetic algorithm. In *2011 IEEE International Conference on Quality and Reliability*, pages 385–390. IEEE.

Englert, M., Röglin, H., and Vöcking, B. (2007). Worst case and probabilistic analysis of the 2-opt algorithm for the tsp. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1295–1304. Society for Industrial and Applied Mathematics.

Gendreau, M., Hertz, A., and Laporte, G. (1992). New insertion and postoptimization procedures for the traveling salesman problem. *Operations Research*, 40(6):1086–1094.

Gendreau, M., Hertz, A., Laporte, G., and Stan, M. (1998). A generalized insertion heuristic for the traveling salesman problem with time windows. *Operations Research*, 46(3):330–335.

Golden, B., Bodin, L., Doyle, T., and Stewart Jr, W. (1980). Approximate traveling salesman algorithms. *Operations research*, 28(3-part-ii):694–711.

Grefenstette, J., Gopal, R., Rosmaita, B., and Van Gucht, D. (1985). Genetic algorithms for the traveling salesman problem. In *Proceedings of the first International Conference on Genetic Algorithms and their Applications*, volume 160, pages 160–168. Lawrence Erlbaum.

Grefenstette, J. J. (1986). Optimization of control parameters for genetic algorithms. *IEEE Transactions on systems, man, and cybernetics*, 16(1):122–128.

Häme, L. (2011). An adaptive insertion algorithm for the single-vehicle dial-a-ride problem with narrow time windows. *European Journal of Operational Research*, 209(1):11–22.

Jaw, J.-J., Odoni, A. R., Psaraftis, H. N., and Wilson, N. H. (1986). A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. *Transportation Research Part B: Methodological*, 20(3):243–257.

Johnson, D. S. and McGeoch, L. A. (1997). The traveling salesman problem: A case study in local optimization. *Local search in combinatorial optimization*, 1(1):215–310.

Jorgensen, R. M., Larsen, J., and Bergvinsdottir, K. B. (2007). Solving the dial-a-ride problem using genetic algorithms. *Journal of the operational research society*, 58(10):1321–1331.

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

Liu, M., Luo, Z., and Lim, A. (2015). A branch-and-cut algorithm for a realistic dial-a-ride problem. *Transportation Research Part B: Methodological*, 81:267–288.

Madsen, O. B., Ravn, H. F., and Rygaard, J. M. (1995). A heuristic algorithm for a dial-a-ride problem with time windows, multiple capacities, and multiple objectives. *Annals of operations Research*, 60(1):193–208.

Masmoudi, M. A., Braekers, K., Masmoudi, M., and Dammak, A. (2017). A hybrid genetic algorithm for the heterogeneous dial-a-ride problem. *Computers & operations research*, 81:1–13.

Paquette, J., Cordeau, J.-F., Laporte, G., and Pascoal, M. M. (2013). Combining multicriteria analysis and tabu search for dial-a-ride problems. *Transportation Research Part B: Methodological*, 52:1–16.

Parragh, S. N., Doerner, K. F., and Hartl, R. F. (2008). A survey on pickup and delivery problems. *Journal für Betriebswirtschaft*, 58(1):21–51.

Psaraftis, H. N. (1980). A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science*, 14(2):130–154.

Psaraftis, H. N. (1983a). Analysis of an o (n2) heuristic for the single vehicle many-to-many euclidean dial-a-ride problem. *Transportation Research Part B: Methodological*, 17(2):133–145.

Psaraftis, H. N. (1983b). An exact algorithm for the single vehicle many-to-many dial-a-ride problem with time windows. *Transportation science*, 17(3):351–357.

Salhi, S. and Nagy, G. (1999). A cluster insertion heuristic for single and multiple depot vehicle routing problems with backhauling. *Journal of the operational Research Society*, 50(10):1034–1042.

Sexton, T. R. and Bodin, L. D. (1985a). Optimizing single vehicle many-to-many operations with desired delivery times: I. Scheduling. *Transportation Science*, 19(4):378–410.

Sexton, T. R. and Bodin, L. D. (1985b). Optimizing single vehicle many-to-many operations with desired delivery times: II. Routing. *Transportation Science*, 19(4):411–435.

Srinivas, N. and Deb, K. (1994). Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary computation*, 2(3):221–248.

Wong, K. and Bell, M. G. (2006). Solution of the dial-a-ride problem with multi-dimensional capacity constraints. *International Transactions in Operational Research*, 13(3):195–208.

Yu, B., Yang, Z.-Z., and Yao, B. (2009). An improved ant colony optimization for vehicle routing problem. *European journal of operational research*, 196(1):171–176.