Vehicle Pick-up and Drop-off Schedule Optimization in a University Setting

By

Teo Chun Kit

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

JUNE 2024

Vehicle Pick-up and Drop-off Schedule Optimization in a University Setting

By

Teo Chun Kit

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

JUNE 2024

UNIVERSITI TUNKU ABDUL RAHMAN

REPORT STATUS DECLARATION FORM

Title: _____Vehicle Pick-up and Drop-off Schedule Optimization in a ______ University Setting

Academic Session: ____June 2024____

I ____________TEO CHUN KIT____________
(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

	Verified by,
[illegible]	[illegible]
(Author's signature)	(Supervisor's signature)
Address:	
_2059, Jalan Seksyen 2/3, ____	
_Taman Bandar Barat________	_Prof Liew Soung Yue______
_31900 Kampar, Perak_______	Supervisor's name
Date: ____12/9/2024________	**Date**: 12/9/2024

Typ

Universiti Tunku Abdul Rahman			
Form Title : **Sample of Submission Sheet for FYP/Dissertation/Thesis**			
Form Number: **FM-IAD-004**	Rev No.: **0**	Effective Date: **21 JUNE 2011**	Page No.: **1 of 1**

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

UNIVERSITI TUNKU ABDUL RAHMAN

Date: ____12/9/2024_______

SUBMISSION OF FINAL YEAR PROJECT /DISSERTATION/THESIS

It is hereby certified that ______***Teo Chun Kit***__________________________ (ID No: __***22ACB00091***___) has completed this final year project entitled "____________*Vehicle Pick-up and Drop-off Schedule Optimization in a University Setting*_______" under the supervision of ____Prof Liew Soung Yue_________________ (Supervisor) from the Faculty of ___Information and Communication Technology_____ .

I understand that University will upload softcopy of my final year project in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,

[illegible]

____
(*Teo Chun Kit*)

*Delete whichever not applicable

DECLARATION OF ORIGINALITY

I declare that this report entitled "**TITLE Vehicle Pick-up and Drop-off Schedule Optimization in a University Setting**" is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature	:	[illegible]
Name	:	Teo Chun Kit
Date	:	12/9/2024

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisor, Prof. Liew Soung Yue who is willing to supervise me in this project. Your bright ideas and your experiences in this field helped me a lot to keep improving this project. A million thanks to you.

Next, I would like to express my thanks to Mr. Ch’ng Chee Henn who was willing to provide me with ideas for improvements to my project and helped me to solve some of my unsolvable problems.

Finally, I must say thanks to my parents and my family for their love, support, and continuous encouragement throughout the course.

ABSTRACT

This project aims to enhance the convenience and safety of university students and staff by developing an optimized vehicle pick-up and drop-off scheduling system, integrating aspects of the Dial-a-Ride Problem (DARP) and the Carpooling Problem (CPP) to better suit the university setting. The formulated problem is a multi-objective, many-to-many, one-day scheduling problem with both static and dynamic components. Uniquely, participants can serve as both drivers and passengers, with fairness constraints applied equally to both roles. The primary objectives include minimizing earliness waiting times, reducing DARP-like cases, and lowering total expenses, with penalties imposed for unserved requests. Lateness will be removed using a lateness waiting time rollback mechanism. A simulated annealing-based multi-directional iterative local search algorithm is employed for solution optimization. The initial solution is generated by distributing requests across vehicles, and local searches are performed through request swapping and movement. Simulated annealing explores the solution space in multiple directions to avoid convergence to suboptimal solutions, with iterative loops preventing premature convergence. For dynamic requests, a handler evaluates acceptance based on time constraints, and schedule re-optimization is triggered as necessary, using the same methods as in the static case. Extensive experiments validate the algorithm's effectiveness, optimize parameters, and demonstrate the dynamic handler's ability to manage real-time requests accurately. The results confirm the efficiency and robustness of the proposed approach in both static and dynamic scenarios.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

$\sum$	Sigma, the total of
km	kilometers
RM	Ringgit Malaysia

LIST OF ABBREVIATIONS

CPP	Carpooling Problem
DARP	Dial-a-ride Problem
IDE	Integrated Development Environment
ILS	Iterated Local Search
SA	Simulated Annealing
VRP	Vehicle Routing Problem
EDOD	Effective Degree of Dynamism
DDARP	Dynamic Dial-a-ride Problem
QOS	Quality of Service

CHAPTER 1

Introduction

In this chapter, we present the background and motivation of our research, our contributions to the field, and the outline of the report.

1.1 Problem Statement and Motivation

In recent years, the number of vehicles on the road has increased rapidly due to urbanization and rapid city growth. The living standards of people have improved significantly over the years, leading more and more people able to afford a private car. This will cause a lot of problems, for example, traffic congestion, parking problems, environmental pollution, and energy crisis. According to Su et al., traffic congestion has cost the world an economic loss of almost 2.5% of GDP per year [1].

The situation is similar in a university. Imagine that there is a university with 20,000 students and employees. Some of them live in hostels, and some of them live outside of university. Each hostel is in a different location around the university, and 60% of all students and employees drive cars to the university and park their cars in the university. The parking space in the university is limited, and the destinations of each person are mostly different due to having different courses and these courses are distributed across different blocks in the university.

As most students and lecturers will go home if there is a long empty time slot before their next class, traffic congestion will not only happen between 8 am and 6 pm but also the time in between if there are quite some people going back home or coming to the university. Even though public transport may be provided, its capacity is limited, and it cannot send passengers to each of their specific destination points, as public transport will only stop at some specific stops in the university, rather than each building stop once. Passengers will still need to walk to their specific destination if it is not a public transport stop.

Other than that, it is difficult for a university to have enough parking space to accommodate all cars. Some may find a parking place, but they will need to park at a place far away from their destination, which is inconvenient. This is not a comfortable experience for them, as they will need to walk under the hot sun to their destination, and sometimes under the rain. Some more unfortunate ones will not find any parking place that is relevant to their destination. They can only either choose to park very far away or just keep turning around the parking lot.

Moreover, as not all students or staff have car, some will need to ride a bike to university. Even though there are speed limit signs every way in the university, the students will still drive very fast. There may be bumps or pedestrian lines to make the lives of bikers better, but not all cars will slow down on bumps and let bikers or pedestrians cross first at the pedestrian line. Sometimes they will not even slow down when bikers are about to cross junctions, thinking all bikes have the same speed as cars. These situations are dangerous to bikers, and the risks will be even higher when the number of cars increases. Accidents may happen because of these issues.

Although the above-mentioned problems may not cause serious monetary loss to university employees or students, it will cause a huge deal of inconvenience to them, and sometimes traffic accidents may also happen from it.

1.2 Background Information

There are some background information readers should know to better understand this project. To solve the ever-growing congestion problem, researchers have come out with a solution, which is the vehicle routing problem (VRP). VRP encompasses the dial-a-ride problem (DARP) and carpooling problem(CPP). DARP is a type of problem where customers call for a ride (make a request), and then drivers will be assigned to serve the request of that customer. Nowadays e-hailing services and taxi services are all types of DARP.

VRP is an NP-hard problem according to [2]. One can understand the basic concept of VRP by studying the traveling salesman problem in [2]. With VRP, several people can share one vehicle, thus helping to reduce the number of cars on the road to a significant degree. The concepts of DARP and CPP are near, just that their main objective is different, thus some of their constraints are also different. DARP mainly focuses on minimizing total travel costs and passenger service quality, while CPP mainly focuses on ensuring fairness among all participants.

The ways to optimize DARP and CPP are the same. A mathematical model will be proposed to represent the problem to be solved, and then an optimization algorithm, whether heuristic or meta-heuristic will be proposed to find the best solution that can best meet the main objective for the proposed problem. Both DARP and CPP have two variants, static and dynamic. For static problems, all information concerning passengers' requirements is defined and fixed beforehand [2]. For dynamic problems, passenger requirements can be changed in real-time. New requests will be expected to come in, and the program need to decide to either accept or reject them. If the requests are accepted, the program needs to find a way to serve them while making sure the route or solution is still feasible [3]. DARP has two variants, which are homogenous DARP, where each vehicle has the same starting and ending depots, and heterogenous DARP, where each vehicle has different starting and ending depots.

CPP can be categorized into two distinct types: Daily Carpooling and Pick-up (DCPP) and Long-Term Carpooling and Pick-up (LTCPP). In the context of DCPP, drivers pick up passengers and return them on the same day. The primary aim here is to efficiently allocate passengers to drivers while considering time constraints and car capacity limitations [1]. LTCPP, conversely, involves the sharing of a private vehicle by multiple individuals. These individuals collectively follow a semi-common route connecting different starting and ending points during a specific period. It's worth noting that each participant in LTCPP will be assigned to take on the roles of both driver and passenger [4]. Participants of carpool in CPP will only have two requests per day: from origin to destination and vice versa. For each request, there will be other requests with similar pick up and drop off time, close origins, and destinations to be handled by the same vehicle. According to the origin and destination points of each participant, the CPP can be many-to-many, one-to-one, or many-to-one. Most CPPs formulated by researchers are of the one destination variant.

To start the optimization process, we first need to come out with an initial solution. The distribution of different solutions in a solution space can be likened to a graph. The optimization process is a solution-searching process that aims to find the lowest point in that graph, which is the best solution. There will be some scenarios where the search process will reach a low point, but not the lowest point in the graph, and converge at that low point. This will be the local minimum problem. Thus, most optimization algorithms will also handle the escaping from the local minimum [5].

Moving on to the dynamic scenario, dynamic optimization will also have objectives to be met [6]. It is like static optimization in terms of constraints, as it has a fleet size, a car capacity, a time window, etc. Request rejection can be accepted in dynamic scenarios, as the satisfying of the hard time window constraints of the existing static requests should be prioritized. Other than that, a program may choose to not entertain urgent requests to ensure the comfort of those whose requests are being currently served. There are several different ways to measure the dynamism of the scenario [7], which means how dynamic the scenario is, but in this project, we will only use the effective degree of dynamism (EDOD) to measure dynamism. Two main ways to handle dynamic requests are insertion heuristics, as shown in [8], and metaheuristics such as tabu search and genetic algorithm.

Lastly, the university scenario is special. It has both the static and the dynamic nature. Students and staff have a fixed timetable, and based on these timetables static requests will be generated. But, in a university, same-day class cancelations or additions may happen, students and staff may have unplanned, urgent meetings, or they may only remember they have something to do later the same day. These scenarios will generate dynamic requests.

1.3 Project Scope

The aim of this project is to formulate the problems stated in the problem statement and develop a model to solve the problems. The model will generate a schedule to handle all passenger requests from a starting point to a destination, subjective to several constraints that will be explained in Chapter 3. A solution searching and optimization algorithm will be used to optimize the schedule to better meet the passengers' and drivers' objectives. After the best schedule has been generated, the schedule will be simulated, and new requests will be coming in based on probability to simulate the dynamic scenario.

1.4 Project Objectives

This project aims to propose a model for generating schedules (solutions) that can ensure all students and employees have a pleasant experience coming to the university and going back by reducing the number of cars thus reducing congestion during peak hours. By reducing the number of cars in the university, the parking space provided by the university is enough to accommodate the number of cars that are parked in the university so there is no need to introduce policies such as bidding of car stickers. Students will no longer need to try out their luck just to get a chance to park at the university. Other than that, the solution will focus on point-to-point pick up and drop off so users will not need to walk too long to reach their destination points from their vehicles or vice versa.

To build a model to solve the problems stated in the problem statement, we first need to formulate the problems. The general approaches by researchers to formulate these kinds of challenges are by formulating them as CPP or DARP. However, as both methods come with their pros and cons, they should be tweaked according to the exact scenario of the problems to solve. This project will attempt to formulate the problems as a combination of DARP and CPP to better suit the university setting of the problems to be solved by this project. Drivers and passengers will not be treated as separated roles, meaning that drivers can be passengers after finish serving a request, and passengers can be drivers after reaching their destinations.

Multi-objective optimization will be studied and implemented in this project, mainly focusing on optimizing the overall traveling expenses, earliness and lateness waiting time, and the number of dial-a-ride-like cases. Apart from that, this project will study the possibility of tweaking the traditional simulated annealing (SA) algorithm to make it more suitable for the formulated problem of this project. Therefore, an SA-based multi-directional iterative local search algorithm will be built in this project as the solution searching and optimization algorithm.

After that, university students may have sudden changes in their class schedules, for example, additional classes or unscheduled meetings. This means dynamic requests may come in during the running of the vehicle schedule. This project will study a way to serve these dynamic requests, while not changing the service details of those requests that are currently being served. This dynamic optimization solution will also ensure the constraints and service quality assurance of the static scenario can be met.

This project will only focus on the algorithm part and will not develop an application based on it, meaning that the solution will be a runnable program on IDE, not a well-decorated mobile or web application. The expected output will be a schedule in simple form on a text file, together with the values of objective-related indicators such as total lateness and total cost.

1.5 Contributions

This project formulates problems from a university setting. This is rare among the currently available research work as most of them focus on generic settings, which may not always match the nature of a university setting. Moreover, both the solution to the static part and the dynamic part of the university scenario will be studied. This project will show that the proposed model, algorithms, and dynamic scenario-handling mechanism can effectively optimize the formulated problem.

Apart from that, the combination of DARP and CPP will be studied. Incorporating some aspects of DARP into CPP in problem formulating will make the problem-solving model more robust to serve many-to-many scenarios. Although there is already research done to formulate many-to-many CPP, the model proposed in this project attempts to formulate the problems in an easier-to-understand way and to better meet the real-world scenario. Aside from that, a cost model that takes care of both the satisfaction of the drivers and the customers will be formulated.

Other than that, the feasibility of the SA algorithm as the solution optimization algorithm will be studied. The main optimization algorithm used in this project will be SA-based, and it will be tweaked to further improve its performance. The parameters will be tested to find the best configuration for the university scenario.

1.6 Report Organization

The details of this research are shown in the following chapters. In Chapter 2, some similar models with different objectives are reviewed. Moreover, different optimization algorithms will be studied to compare with the algorithm used in this project. Various approaches to handle dynamic scenarios will also be studied. Then, in Chapter 3, the details of the model proposed in this project, the algorithms, and the dynamic scenario handling mechanism will be explained. Next, in Chapter 4, the simulation and experiment configuration of both the static and dynamic scenarios will be presented. Chapter 5 will show the experiment results and explanation. Chapter 6 concludes this project and suggests recommendations for this project.

CHAPTER 2

Literature Reviews

2.1 Previous Works on Problem Formulating on Similar Challenges

2.1.1 DARP

The previous works done on formulating DARP mainly differ on their optimization objectives and constraints, but the challenges they formulate their problems on are quite similar, mainly to solve congestion-related challenges.

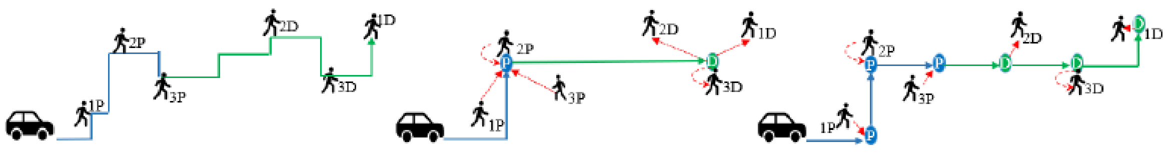
A classic DARP is like the one proposed by J. F. Cordeau and G. Laporte. There are a certain number of passengers, and a certain number of cars, and passengers each have requests for pick up and drop off. Drivers and passengers are separated, meaning that the roles of passenger and driver cannot interchange. Each request will be specified with a time window, and the arrival time exceeding that time window will be penalized. This time window will ensure passengers are less likely to be late to reach their destinations. The requests will be handled by each car by complying with certain constraints. The optimization objective of their work is also the general one that is used in many other similar works, which is to reduce total cost. However, their model is generic, so it needs some tweaking to meet specific scenarios, for example, the university setting studied in this project [9].

G. R. Mauri et al. proposed a DARP that further prioritizes passenger satisfaction in terms of total costs, total traveling time and distance, and waiting time. This is a multi-objective optimization, as it not only has multiple objectives but also uses multiple objective functions to handle different passenger satisfaction indicators. The DARP model proposed in this work is heterogeneous, which means vehicles have different start and end depots [10]. Their model handles customers' satisfaction very well, as their objectives are often what customers will prioritize when choosing an e-hailing or taxi service. However, if they want the pick up and drop off time of each request to be very precise according to the exact traveling time, the formulated problem may be too complex to solve, thus sometimes may not have a feasible solution.

Other than that, S. Ouasaid and M. Saddoune proposed a DARP that takes driver's preferences into account. This is a novel work as most DARP formulations will only give priority to passengers' comfort without considering the drivers' side. Their model penalizes both excess traveling time and excess waiting time for drivers. Their model has only a single optimization objective, which is to reduce the total traveling time. Although only have one objective, their model consists of three objective functions to handle both the penalty imposed on excess travel and waiting time for drivers [11]. This approach should be taken into account when formulating DARP, as drivers are also an important part of DARP.

M. Posada et al. proposed another variant of DARP, the integrated dial-a-ride problem (IDARP). IDARP in essence means the integration of different types of transportation together, meaning that for a passenger's request, the transportation that serves the passenger may change from car to bus and then to bike throughout the whole trip [12]. This study even included human walking as a type of transportation and included wheelchairs as a type of transport to meet the needs of disabled ones. This approach accepts vehicles with different speeds and capacities. IDARP is a more robust way of problem formulation and can better represent real-world scenarios in a general way but it doesn't take the comfort of passengers into serious consideration, as they will need to do extra actions such as walking or changing transportation, rather than just hop in a vehicle on starting point and hop off on destination, which is more comfortable for a passenger.

A. Ham introduced a DARP with an additional feature known as Express-Pool and with Friend-Only constraint. Express-Pool enables several passengers to be picked up by a car at a pickup point near the starting points of those passengers. It is different from the usual door-to-door pick up used by DARPs and can remove more detours compared to the Meeting Places concept, where passengers move to the starting point of one or two passengers to be picked up by a car, as shown in figure 2.1. Express-Pool is similar to public transport [13]. This feature does reduce the total travel distance and travel time of a car, but passengers will need to walk or find a way to the alternative pick up point, which is so much more inconvenient compared to door-to-door pick up. On the other hand, Friend-Only is a constraint to only enable the passenger permitted by the driver to be served by that car. This constraint improves the safety of the dial-a-ride process, but it will make the problem so much more complex that it is hard to find feasible solutions for a larger number of passengers and requests.



(a) Door to Door: a vehicle visits every original pickup and dropoff node.

(b) Meeting Places: some of original pickup (dropoff) nodes are selected for meeting places (hub).

(c) Express Pool: alternative pickup (dropoff) nodes are identified to minimize a detour.

Figure 2.1 Concept of door-to-door, Meeting Places, and Express-Pool and comparison between them [13]

2.1.2 CPP

Most previous works done on formulating CPP differ in their number of objectives, and whether it is many-to-many, or many-to-one. The challenges they formulate their problems on are quite similar, and also similar even to DARP, which are also congestion-related challenges.

S. Yan et al. proposed a long-term many-to-many CPP (LMMCPP). In their work, passengers can have different origins and destinations, but if for example a car has 4 seats, with only 3 passengers in it including the driver, and there is a request along their route to their specific destinations that the car can serve if making a detour, the car will not serve that request. The output of the model is a long-term carpooling schedule, meaning that the schedule will at least encompass a few days. Passengers with close origin and destination will be made into a passenger group, and different cars will serve them on different days [4]. The person who will be assigned as driver among that group will also differ across different days. As CPP prioritizes fairness, this proposed model optimization objective will be to reduce the difference between costs paid by each passenger, and it has only one objective. This approach is good as it can solve many-to-many problems, but it may limit the number and quality of solutions as it does not permit detours.

S. Su et al. proposed a long-term CPP (LTCPP). It is a many-to-one single objective model. Its objective is to find the shortest route for all participants [1]. This model ensures some fairness on the driver side by restricting a participant to be assigned as driver for two consecutive days. However, the number of times each participant is assigned as driver will not be taken into account. This model's taking care of drivers' satisfaction is not complete enough.

On top of their previous work, S. Su et al proposed another model to tackle LTCPP with multiple objectives. It still retains the many-to-one attribute of the previous model. This model showed that multiple-objective CPP is doable. This model aims to minimize the total travel distance and total waiting time of passengers only at the starting point [14]. It considered the satisfaction of passengers, but in real life, people are more sensitive to being late to reach destinations than to waiting for too long at origin. Minimizing the waiting time at the destination side, in other words, minimizing lateness should be implemented in this model as an objective instead.

2.2 Previous Works on Different Optimization Algorithms

Below are several algorithms used by researchers to either find a good quality initial solution or to find the optimum solution.

2.2.1 Tabu Search

Tabu search is a metaheuristic approach to finding the best solution while escaping the local minimum. This method creates a tabu list to record the action done by the searching algorithm, called steps. For example, car A swap request 1 with request 2 of car B. Then this action cannot be done again by the searching algorithm for a specific number of iterations, or tenure period. After the tenure period, the step in the tabu list will be removed and the searching algorithm can take that step again [5]. Figure 2.2 shows the pseudocode of a tabu search algorithm.

```
Begin Tabu Search
  $s_0$ : initial solution
  $s_{current} = s_0 ; s_{best} = s_0 ; \theta = 0; TL = 0;$
  for ( $j = 1; j < nbclient; j++$ )
   { //apply nearest neighbour
      $s = Nearest\_Neighbour (s_{current});$
      for ( $i = 1; i < \theta_{\max} ; i++$ )
        {//find the best non-tabu solution
           update TL;
           //use the permutation local search
           $s' = permutation (s_{current});$
           $s_{current} = 2 - move (s');$
           $value = evaluate (s_{current});$
           if ( $value < s_{best}$ )
             {
               $s_{best} = value;$
               $\theta++;$
             }
         update TL;
        }
   }
end Tabu Search
```

Figure 2.2 Pseudocode of Tabu Search Algorithm [15]

In the work of S. Ho et al., they introduced an improved version of tabu search implementation. They use construction heuristic to find a high-quality initial solution within a short time, and then use the tabu search algorithm to find the best solution [16]. This can be viewed as a hybrid approach to implementing optimization algorithms. Based on their test results, their version of the tabu search algorithm outperformed the ordinary tabu search algorithm in terms of the time needed to obtain the optimal solution and the quality of the solution. Things to take note of are in more complex test cases, the improved tabu search outperformed the original tabu search by nearly 84% for runtime. The usage of construction heuristic in finding the initial solution clearly helps in this improvement as the algorithm will no longer need so many steps to reach the best solution.

2.2.2 Simulated Annealing

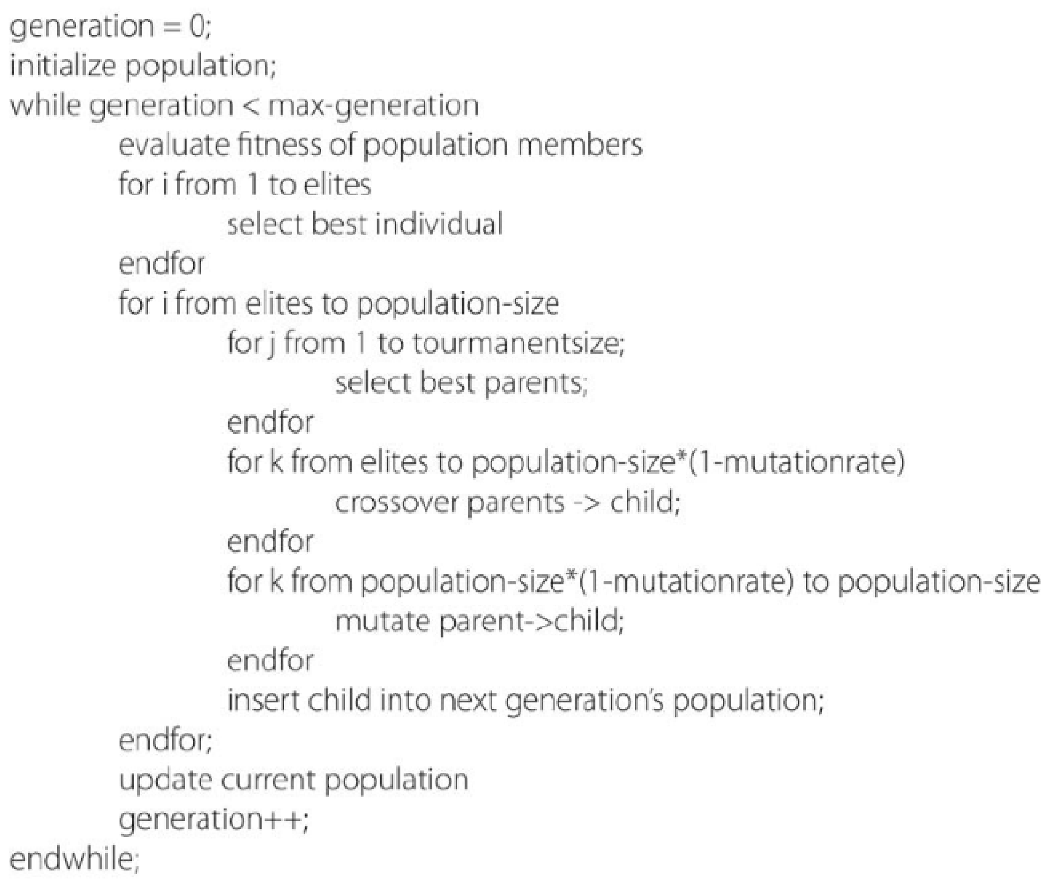
Simulated annealing (SA) is another metaheuristic approach to search for the best solution and escape the local minimum. In SA, the better solution is always accepted. The worse solution will have a probability of being accepted, called temperature. Temperature does not necessarily need to be implemented as probability, but it must represent the algorithm’s willingness to accept worse solutions. The temperature of a SA algorithm will be adjusted across the increasing number of iterations. SA sometimes accepts worse solutions as the solution may have something good in it. If we view the search process as a graph, a worse solution may be a means for the search algorithm to hop to the best solution, thus escaping the local minimum [5]. Figure 2.3 shows the pseudocode for the SA algorithm.

```
Initialize parameters;
S=generate initial solution ( );
$T = T_0$;
While $(T < T_{final})$
{
    Until $(N \leq I - Iter)$
    {
        Generate solution $S'$ in the neighborhood of $S$
                if $f(S'') < (f(S)$
                        $S \leftarrow S'$
                else
                        $\Delta = f(S') - f(S)$
                        $r = random(\,);$
                        if $(r < \exp(-\Delta / k * T))$
                        $S \leftarrow S'$
    }
        $T = a \times T;$
}
Return the best solution found;
```

Figure 2.3 Pseudocode of Simulated Annealing Algorithm [17]

2.2.3 Genetic Algorithm

Genetic algorithm is a metaheuristic approach that imitates the biological evolution process, in other words, survival of the fittest. To apply genetic algorithm, first, a solution space needs to be generated. The number of solutions in the initial solution space can be adjusted according to specific problems. Throughout the solution-finding process, the size of the solution pool must be fixed to the same number. For each solution, a fitness score is calculated to show how good is the solution. Different solutions in a solution space will then crossover, which means exchanging some parts between them. The fitness score will be calculated for the new solution. Solutions with a higher fitness score have a higher probability of survival, which means to not be removed. Removal of solutions from the solution list is done in every iteration to maintain the number of solutions in the solution list. In rare occasions, mutation will occur, and the action done for mutation can be modified according to needs. The probability of mutation will also be set beforehand, and it can be adjusted. In the end, the solution with the highest fitness score will be chosen as the optimum solution [5]. Figure 2.4 illustrates the pseudocode for genetic algorithm.



```
generation = 0;
initialize population;
while generation < max-generation
        evaluate fitness of population members
        for i from 1 to elites
                select best individual
        endfor
        for i from elites to population-size
                for j from 1 to tourmanentsize;
                        select best parents;
                endfor
                for k from elites to population-size*(1-mutationrate)
                        crossover parents -> child;
                endfor
                for k from population-size*(1-mutationrate) to population-size
                        mutate parent->child;
                endfor
                insert child into next generation's population;
        endfor;
        update current population
        generation++;
endwhile;
```

Figure 2.4 Pseudocode of Genetic Algorithm [18]

J. Li et al. proposed an improved version of the genetic algorithm for optimizing DARP. Their work solved the slow convergence problem for the original genetic algorithm. They replace the crossover and mutation action with four genetic operators: transfer, swap, segment exchange, and reshuffle [19]. Figure 2.5 illustrates the 4 different genetic operators. Only one genetic operator will be chosen in an iteration, and the operators are chosen based on a roulette wheel-like probability generator. The other parts of the algorithm are the same as the original genetic algorithm. The whole process of generating offspring and eliminating solutions is shown in Figure 2.6.

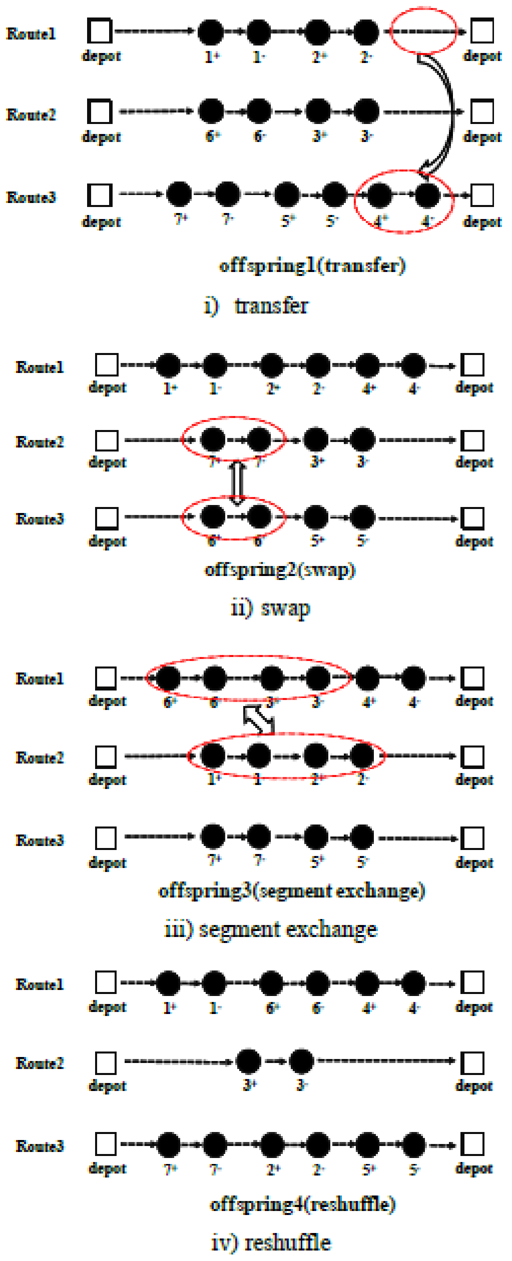


Figure 2.5 The Four Genetic Operators to Generate Offsprings in The Improved Genetic Algorithm

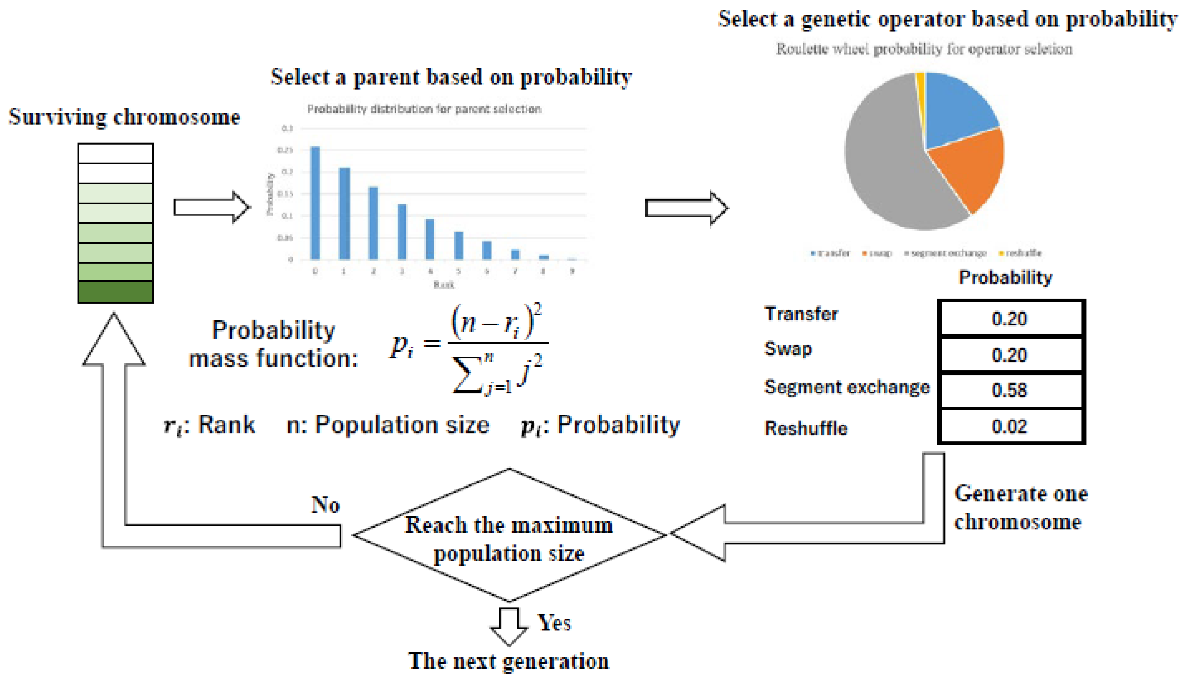
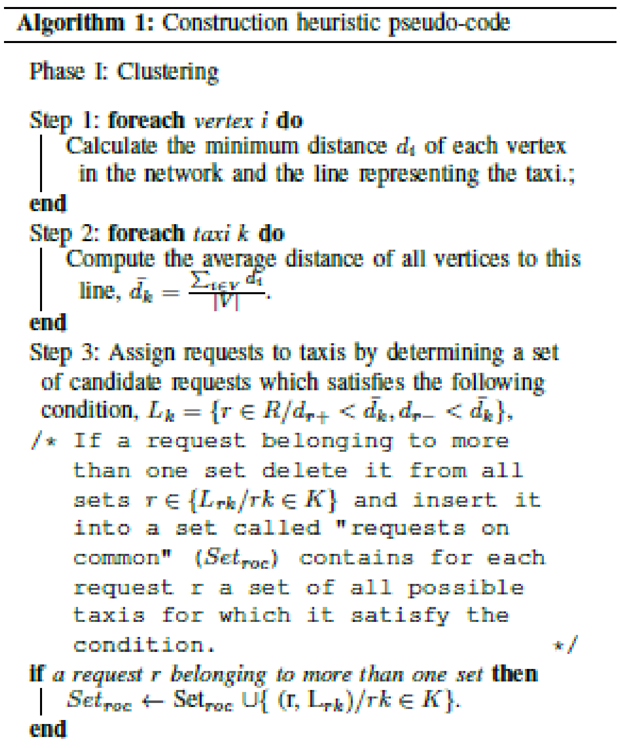

Select a genetic operator based on probability
Roulette wheel probability for operator selection
Select a parent based on probability
Probability distribution for parent selection
Surviving chromosome
Probability mass function: $p_i = \frac{(n-r_i)^2}{\sum_{j=1}^{n} j^2}$
r_i: Rank n: Population size p_i: Probability
Probability
Transfer 0.20
Swap 0.20
Segment exchange 0.58
Reshuffle 0.02
Generate one chromosome
Reach the maximum population size
No
Yes
The next generation

Figure 2.6 Offspring Generation and Solution Elimination Process in The Improved Genetic Algorithm [19]

According to their experiment results, their improved version of genetic algorithm had outperformed both the original genetic algorithm, tabu search algorithm, and simulated annealing algorithm in terms of time needed to obtain the optimum solution and the quality of the optimum solution [19]. However, this can only prove that the improved genetic algorithm performs well in their type test case settings. It is not guaranteed to be better than the algorithms used as a comparison in different test case settings.

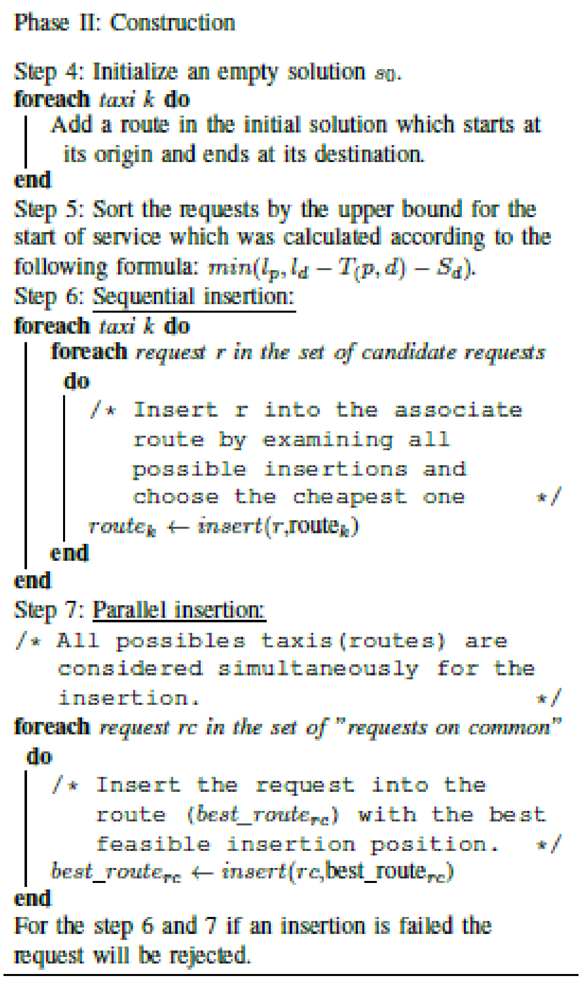
2.2.4 Construction Heuristics

Construction heuristics are methods used to find good quality initial solutions quickly. S. Ouasaid and M. Saddoune proposed the use of construction heuristics in finding better initial solution for their DARP. They proposed a "Cluster-First Route-Second" approach, in which they first group requests with geographically close origins and destinations into a cluster [20]. They called this the "clustering phase". Requests that are along the chosen route of a car will also be grouped together. Then these requests will be assigned to each car. The requests will kept being reassigned between each car to find the best possible initial solution, in the expense of some requests will not be served. They called this the “construction phase”. The pseudocode of the two phases of this algorithm is shown in Figure 2.7 and Figure 2.8.



```
Algorithm 1: Construction heuristic pseudo-code

Phase I: Clustering

Step 1: foreach vertex i do
    Calculate the minimum distance $d_i$ of each vertex
      in the network and the line representing the taxi.;
end
Step 2: foreach taxi k do
    Compute the average distance of all vertices to this
      line, $\bar{d_k} = \frac{\sum_{i \in V} d_i}{|V|}$.
end
Step 3: Assign requests to taxis by determining a set
  of candidate requests which satisfies the following
  condition, $L_k = \{r \in R / d_{r+} < \bar{d_k}, d_{r-} < \bar{d_k}\}$,
/* If a request belonging to more
   than one set delete it from all
   sets r $\in \{L_{rk} / rk \in K\}$ and insert it
   into a set called "requests on
   common" $(Set_{roc})$ contains for each
   request r a set of all possible
   taxis for which it satisfy the
   condition.                              */
If a request r belonging to more than one set then
    $Set_{roc} \leftarrow Set_{roc} \cup \{$ (r, $L_{rk}) / rk \in K\}$.
end
```

Figure 2.7 Pseudocode of The First Phase of The Construction Heuristic [20]



```
Phase II: Construction

Step 4: Initialize an empty solution $s_0$.
foreach taxi k do
    Add a route in the initial solution which starts at
    its origin and ends at its destination.
end
Step 5: Sort the requests by the upper bound for the
start of service which was calculated according to the
following formula: $min(l_p, l_d - T(p,d) - S_d)$.
Step 6: Sequential insertion:
foreach taxi k do
    foreach request r in the set of candidate requests
    do
        /* Insert r into the associate
           route by examining all
           possible insertions and
           choose the cheapest one          */
        $route_k \leftarrow insert(r, route_k)$
    end
end
Step 7: Parallel insertion:
/* All possibles taxis(routes) are
   considered simultaneously for the
   insertion.                               */
foreach request rc in the set of "requests on common"
do
    /* Insert the request into the
       route ($best\_route_{rc}$) with the best
       feasible insertion position.      */
    $best\_route_{rc} \leftarrow insert(rc, best\_route_{rc})$
end
For the step 6 and 7 if an insertion is failed the
request will be rejected.
```

Figure 2.8 Pseudocode of The Second Phase of The Construction Heuristic [20]

This algorithm indeed obtained a good initial solution, and this will be a great help in getting a better quality optimum solution with a faster convergence speed, but in real-world scenarios, people will care more about whether their requests are served rather than the speed their requests are being arranged into a schedule, or the quality of that schedule. So a balancing point should be determined to maximize the quality of the initial solution in the condition of all requests can be served.

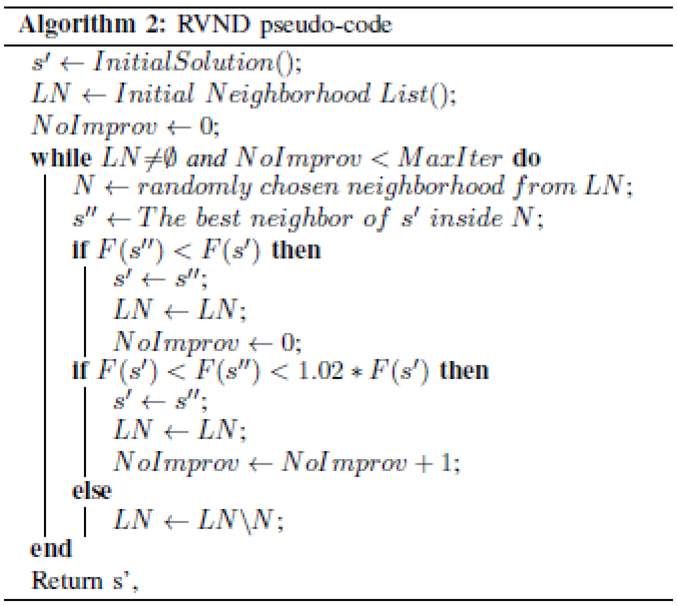
2.2.5 Mixed integer linear programming

This approach formulates the whole model into lines of linear equations and attempt to solve it using software tools. Most of the time this approach will be used to find the initial solution, but with the help of powerful software tools like CPLEX, it can also be used to find the optimum solution [21]. For examples of mixed integer linear programming approach, we can study the work of M. Posada et al. They proposed two linear programming model to find the initial solution for their DARP. The linear equations for both models are almost the same, just that model 2 will record the route visited by each car to prevent cars from making cycles, thus reducing the time needed to obtain the initial solution and improve the solution quality [12]. As to best utilize the strength of linear programming in optimizing vehicle routing-related problems software tools are needed, linear programming approach is not too suitable for this project.

2.2.6 Iterated Local Search With Two Local Search Methods

The iterated local search with two local search methods is an improvement made to the iterated local search (ILS) heuristics. This improved algorithm is proposed by S. Ouasaid and M. Saddoune. They claim this solution-searching algorithm can come out with a better solution compared to the traditional ILS algorithm [11].

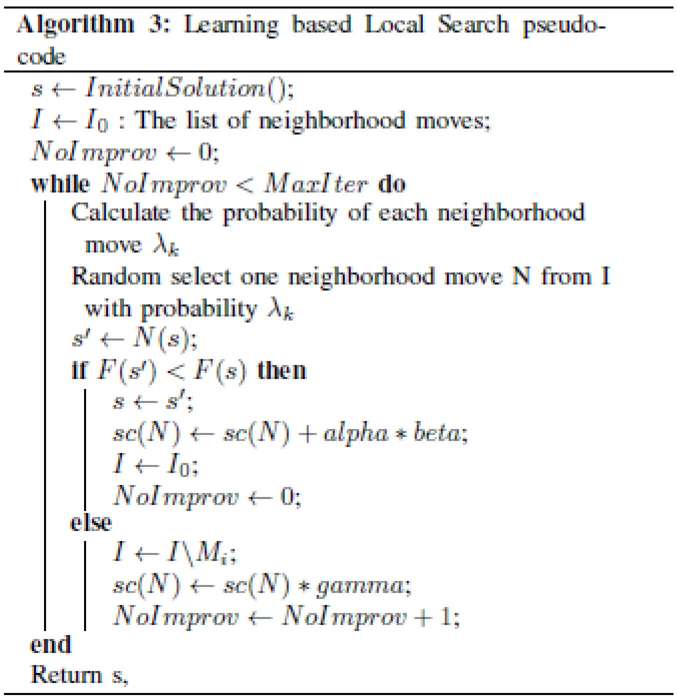
The first local search method they implemented is the random neighbourhood visit order (RVND). The RVND does neighbourhood visitation in three ways: moving a request from one car to another car, swapping lists of requests between two cars, and swapping the order of two empty cars. The decision of which type of visitation is chosen is randomized. Then the best solution obtained by the visitation is taken. The pseudocode of RVND is shown in Figure 2.9 [11].



```
Algorithm 2: RVND pseudo-code
s' ← InitialSolution();
LN ← Initial Neighborhood List();
NoImprov ← 0;
while LN≠∅ and NoImprov < MaxIter do
    N ← randomly chosen neighborhood from LN;
    s'' ← The best neighbor of s' inside N;
    if F(s'') < F(s') then
        s' ← s'';
        LN ← LN;
        NoImprov ← 0;
    if F(s') < F(s'') < 1.02 * F(s') then
        s' ← s'';
        LN ← LN;
        NoImprov ← NoImprov + 1;
    else
        LN ← LN\N;
end
Return s',
```

Figure 2.9 Pseudocode of The RVND Algorithm [11]

The second local search method they used was the learning-based local search. Each neighbourhood or search method is given a score. Methods with higher scores will have a higher probability of being chosen. In the beginning, each method has the same score, and the score of each method will either be added or deducted after certain iterations based on the quality of the solution they obtained. The pseudocode of the learning-based local search is shown in Figure 2.10. The score deduction or addiction rate can be adjusted by the variable alpha, and the addition and deduction value can be adjusted by variable beta and gamma, respectively.



Algorithm 3: Learning based Local Search pseudo-code

```
$s \leftarrow InitialSolution()$;
$I \leftarrow I_0$ : The list of neighborhood moves;
$NoImprov \leftarrow 0$;
while $NoImprov < MaxIter$ do
    Calculate the probability of each neighborhood
      move $\lambda_k$
    Random select one neighborhood move N from I
      with probability $\lambda_k$
    $s' \leftarrow N(s)$;
    if $F(s') < F(s)$ then
        $s \leftarrow s'$;
        $sc(N) \leftarrow sc(N) + alpha * beta$;
        $I \leftarrow I_0$;
        $NoImprov \leftarrow 0$;
    else
        $I \leftarrow I \backslash M_i$;
        $sc(N) \leftarrow sc(N) * gamma$;
        $NoImprov \leftarrow NoImprov + 1$;
end
Return s,
```

Figure 2.10 Pseudocode of The Learning-Based Local Search Algorithm [11]

The approach implemented in this ILS to randomly choose the solution searching method in each iteration is a good approach as they can diversely explore through the solution space. However, its “swap” solution searching method may lose out on some good solutions as it is swapping a list of requests. We can imagine the solution space as a line graph and the searching method as how large the step is to move from one point on the graph to another point. If they are swapping several requests at once, the “step” to move to another solution may be too large thus they may skip over the best solution in a certain step.

2.3 Previous Works on Dynamic Scenarios

To handle dynamic requests, we should first know the level of dynamism of our scenario. The dynamism of the scenario is measured using EDOD, which A. Larsen proposed. According to him, a scenario can be classified as a weak, moderate, or strong dynamic scenario if its EDOD is lesser than 0.3, between 0.3 and 0.8, or more than 0.8 respectively [22]. With different strengths of dynamism, different approaches will be taken to handle the scenario. The DDARP and its similar problems are classified as moderate dynamic scenarios [7], so the review of past approaches will be focused on moderate dynamic scenarios.

Below is the formula for the calculation of EDOD:

Let:

T = end time of the planning horizon.

t_i = the time the accepted dynamic request comes in.

N_i = the number of accepted dynamic requests

N_{total} = the total number of requests, including the dynamic requests

$$EDOD = \frac{\sum_{i=1}^{N_i} \frac{t_i}{T}}{N_{total}},$$ as seen in [7].

As mentioned in the background information, two main ways to handle DDARP are through insertion or insertion with metaheuristics methods. The most basic insertion methods are by inserting a detour in a route to accommodate the dynamic requests [8]. When the dynamic request is too urgent, adding a detour for it may cause the time or cost of the ongoing requests, which are those that are currently being served, to change, and this will cause serious dissatisfaction. So some researchers will attempt to reject urgent requests [7] or lock the ongoing requests.

G. Berbeglia et al. proposed a tabu search and constraint programming (CP) based algorithm to optimize the DDARP. The way the tabu search works is like that of the static cases, which are used to find the best or optimal solution. CP was used to ensure that the solution obtained by the metaheuristics after dynamic requests had been inserted would still comply with the constraints of the formulated problem [23]. With metaheuristics being used, dynamic optimization will come with objectives. Like static optimization, most objectives are either monetary-related or time constraint-related.

As the dynamic requests may be coming in at any time, the solution will need to be reoptimized to insert the requests and ensure the best possible solution. Two main re-optimization approaches are continuous re-optimization and periodic re-optimization. Continuous re-optimization means the solution will be re-optimised throughout the execution of the solution, whereby the rate is based on a preset time interval. This approach ensures the best possible solution, but participants will see their schedules being updated frequently [22]. Periodic re-optimization means the solution will only be updated when some conditions are triggered, for example when new requests come in. This will ensure the least amount of changes on the schedule, but the quality of the solution may not be the best [22].

2.4 Summary

2.4.1 Comparison Between Problem Formulating Approaches

DARP-based formulating approaches

Table 2.1 Comparison Between DARP-based Approaches

Approaches	**Strengths**	**Shortcomings**
Classic DARP	- Handling passengers' satisfaction in terms of lateness	- Generic model, not always suitable for some scenarios
DARP that further prioritize passenger's satisfaction	- Handle passengers' satisfaction very well	- Formulated problem may be too complex to solve - Sometimes may not have feasible solutions
DARP with driver preferences	- Take care of the satisfaction of drivers	
Integrated DARP	- Better represent the real-world scenario - A more robust way to formulate a problem	- Does not take passengers' comfort into serious consideration
DARP with Express-Pool and Friend-Only	- Express-Pool reduces the complexity of the formulated problem - Friend-Only enhance safety	- Not convenient for passengers - Friend-Only make the problem more complex - Hard to find feasible solutions for large numbers of passengers and requests

CPP-based formulating approaches

Table 2.2 Comparison Between CPP-based Approaches

Approaches	**Strengths**	**Shortcomings**
Long-term many-to-many CPP	- Handle many-to-many scenario	- Limit the number of solutions as it does not permit detour
Long-term many-to-one CPP	- Take care of fairness on the drivers' side	- Fairness constraints on the drivers' side are not complete enough
Multi-objective CPP	- Handle multi-objective scenario	- Do not have lateness constraints at the destination side

2.4.2 Comparison Between Solution Searching and Optimization Algorithms

Table 2.3 Comparison Between Different Solution Searching and Optimization Algorithms

Algorithm	**Strengths**	**Shortcomings**
Improved tabu Search	- Less time needed to find the best solution	- Have memory overhead - Not good enough for too complex problems
Simulated annealing	- Can adjust the search intensity - Easier to be implemented - Faster convergence speed	- May affect solution quality due to the possibility of accepting worse solutions

Improved genetic Algorithm	- Reduces convergence time - Improved quality of the best solution	- May only perform this well on specific problems - Need more processing power
Construction heuristics	- Improved quality of initial solution - Less time needed to obtain a good initial solution	- Less suitable to find the optimum solution - May sacrifice some requests to not be served to get a better initial solution
Mixed-integer linear programming	- Can be a very good approach for optimization with the help of software tools	- Need software tools, not suitable for the setting of this project
Iterated local search with random neighbourhood visit order and learning-based local search	- Good solution space exploration approach	- Large solution space exploration “step” - May miss out on better solutions

2.4.3 Comparison with The Approaches Used in this Project

The problem formulation approach used in this project will be a hybrid approach of both DARP and CPP. For implementing classic DARP in the university setting used in this project, a group of drivers will need to be brought into the university to serve the requests of students, which is not realistic. Thus, in this project participants can be assigned as both drivers and passengers, so that students or staff of a university can serve all their requests by themselves. The formulated problem is a many-to-many problem and has two objectives, which are to minimize expenses and earliness waiting time. As the compensation to specially assigned drivers will also be included in the expenses, by minimizing the expenses the frequency of special driver assignment will also be minimized. A multi-objective problem can take care of more aspects of the satisfaction of both the drivers and passengers' side.

Moreover, the model used in this project comes with an expense calculation model that excludes the driver in the petrol fee calculation and imposes extra payments on the passengers if the driver is specially assigned. These will further take care of the fairness and satisfaction on the drivers' side, yet not increase problem complexity as the calculation of payment is just simple math calculations. If the driver also has requests and is currently serving the driver's own request, the same time window applied to passengers will be assigned to the driver. If lateness waiting time is present, it will be removed using a lateness waiting time rollback mechanism. The time window is not a hard constraint, thus not increasing the complexity of the problem. Other than that, the unserved requests for both drivers and passengers will be penalized heavily. This ensured the fairness of between the drivers and the passengers.

The algorithm used in this project for solution searching and optimization is an SA-based multi-directional iterative local search algorithm. The SA-based algorithm is used as it is easier to implement, and compared to genetic algorithm and tabu search, it has less memory overhead and requires less processing power, thus taking less toll on the computational devices. SA also has a faster convergence speed. For the local search method, the methods used in [11] are referred to, and an improved version of their swapping method together with their request moving method is implemented as the local search method for the solution searching and optimization algorithm used in this project. The swapping process will only swap one request at a time. This will reduce the possibility of missing out on better solutions due to little exploration steps.

Apart from that, to avoid early convergence, the iteration will only stop if the quality of the best solution has not been updated for a certain number of iterations. This number will be determined after experiments. To amend the shortcomings of SA, a vector of five solutions will be created, and the SA-based local search will be done for all five solutions in the vector in every iteration. This enables the solution-searching algorithms to have a more diverse exploration direction in the solution space, as it will now search in five directions, and therefore reduce the possibility of converging with worse solutions.

For the handling of dynamic scenarios, this project will use a similar approach to the one proposed in [23]. The SA-based multi-directional iterative local search algorithm will be used to re-optimize the solution, and the optimization objective will be the same as the static scenario. Periodic re-optimization will be used, as the need to check the solution continually over a time interval will bring a great deal of inconvenience to the staff and students. The requests with a starting time within 1 hour from the current time will be locked, and dynamic requests with a starting time within one hour from the current time will also be rejected. This is to ensure participants have sufficient reaction time if their schedule is updated. Participants will only need to check the schedule 1 hour before their requests start, and the arrangement for that request will not be changed after that.

Chapter 3

System Model

3.1 General Workflow

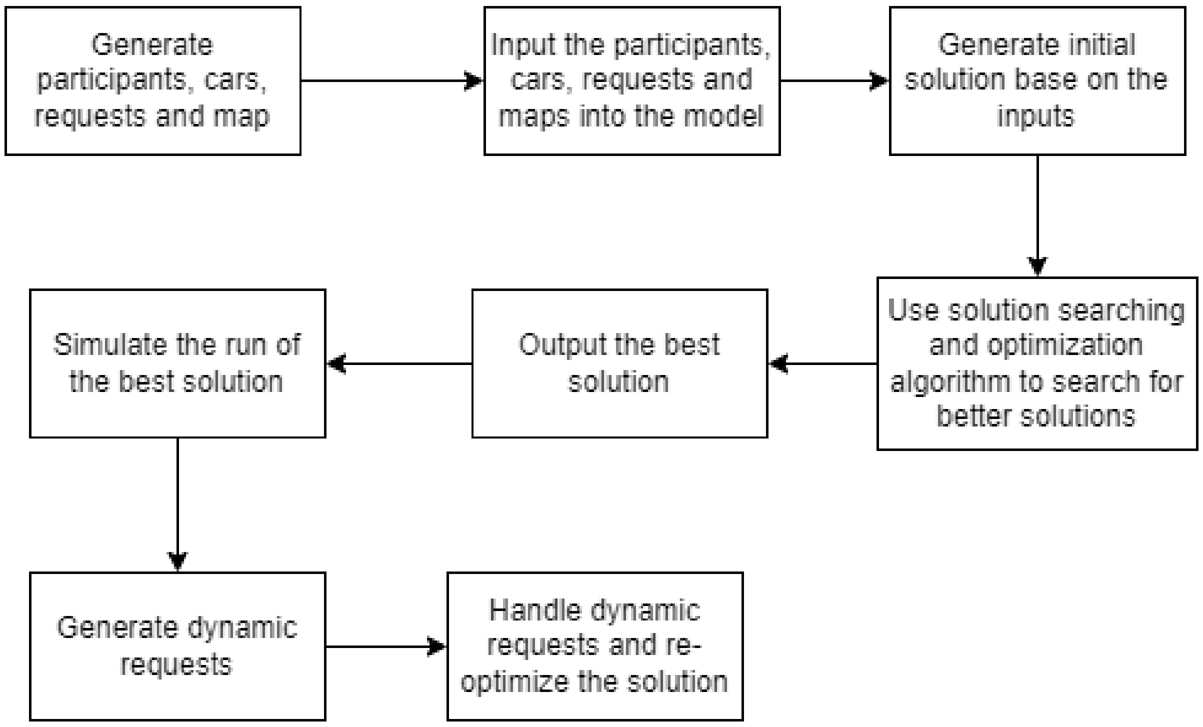

Generate participants, cars, requests and map
Input the participants, cars, requests and maps into the model
Generate initial solution base on the inputs
Use solution searching and optimization algorithm to search for better solutions
Output the best solution
Simulate the run of the best solution
Generate dynamic requests
Handle dynamic requests and re-optimize the solution

Figure 3.1 Pipeline of The Whole Procedure of The Proposed Method

The outputted solution will be a route schedule of the pickup sequence of the cars **on only a day**. The route schedule includes all nodes that each car will pass through together with the time they reach the nodes. The total inconvenience cost, travel expenses, earliness waiting time, lateness waiting time, and frequency of special driver assignment will also be outputted to the same text file. In the dynamic scenario, every time the solution has been re-optimized, the new solution will also be outputted to a text file with the same format as the static scenario.

3.2 System Design

3.2.1 Problem Formulation

This is a many-to-many DARP-based problem, which the means schedule will only be arranged for one day. However, some aspects of CPP will be incorporated into this problem formulation.

Inputs:

- A map in the adjacency list
- Car objects
- Participant objects
- Requests

Variable notations:

Notation	Meaning
Pen	Penalty
S	Travel expenses
C	Total expenses
r	Requests
V	Cars
p	Participants
L	Lateness waiting time

Notation	Meaning
E	Earliness waiting time
x	Decision variable
N_V	Total number of car
N_r	Total number of requests
I	Inconvenience costs
W	Weights in the inconvenience score function
Pen	Penalty

Table 3.1 Variable Notations and Their Meaning

Decision variable:

The decision variable, x for each request is as follows:

$$x = \begin{cases} 1, if\ request\ is\ served \\ 0, otherwise \end{cases}$$

Expenses function:

$$C = \left(\sum_{i=1}^{N_p} S_{p_i} \right)$$

The travel expenses to be paid by each passenger consist of two parts, the petrol fee to travel from one node to another node, and the compensation payment paid to drivers who are specially assigned to serve them. The petrol fee from one node to another node is divided by all participants in the car at that moment except the driver, which means the driver will not need to pay for the petrol fee. The compensation to the specially assigned drivers will be handled in the calculation of the travel expenses.

Penalty:

$$Pen = \left(\sum_{i=1}^{N_r} x_{r_i} \times 5000 \right)$$

In the route schedule, if the arrival time of a car at a participant's destination point to drop off the participant is later than the participant's intended arrival time, lateness waiting time will be recorded. Lateness waiting time will be removed using a lateness rollback mechanism for both drivers and passengers, so it will not be penalized here. Apart from that, any unserved requests will be penalized heavily, as shown in the penalty function above.

Travel expenses calculation:

For normal cases:

$$travel\ expenses = distance\ \times petrol\ price$$

For special driver assignment cases:

$$travel\ expenses = 1.5 \times (distance \times petrol\ price)$$

Where the 0.5 × travel expenses will go to the specially assigned driver as compensation. As stated in the expenses function, these expenses will be divided by all passengers except drivers. The petrol price is fixed at a rate of RM 3 per km.

Objective functions:

$$\min\,(C)$$

$$\min\,(\sum_{i=1}^{p} E_{p_i})$$

In the route schedule, if the arrival time of a car at a participant's origin point to pick up the participant is earlier than the participant's intended departure time, earliness waiting time will be added for the customer. Each customer object will have an attribute storing their total earliness waiting time. As the compensation for the special driver assignment had been included in the total expenses, minimizing total expenses will also minimize the number of special assignment cases.

Inconvenience cost calculation:

$$I = (C \times W_C) + (E_{total} \times W_E) + Pen$$

A weight will be multiplied by each variable to be minimized to denote their importance level. Each variable must have a different importance level for a better optimization result. As there are only two weights to optimize, no matter how we change these weights they can be simplified into a ratio of $1{:}\,x$. For example, if $W_C = 50, W_E = 200$, the ratio of the two weights, $W_C : W_E = 1{:}\,4$. If $W_C = 100, W_E = 200$, $W_C : W_E = 1{:}\,2$. Thus, we will fix W_C to 1 and make W_E changeable to find the best weight combination. This also means that more priority will be given to the lower earliness waiting time. The inconvenience cost function will be as follows:

$$I = C + (E_{total} \times W_E) + Pen$$

The inconvenience cost functions can be summarized as follows:

$$\min\,(I)$$

Driver assignment:

If the starting point of a participant's request is the same as the current location of a car on the intended starting time of that participant, that participant will be assigned as the driver. If the car is at a certain location and there is no request now whose starting point is the same as the car's location, the model will choose one person who is currently at the car's location node to be the driver. This is the case in which participants are specially assigned as drivers to serve some requests, and this case will be referred to as a DARP-like case.

Parameters to be tuned of the algorithm:

- Weight for earliness waiting time
- Number of iterations of not updated lowest inconvenience score to converge

Constraints:

1. Each car will have a fixed capacity of 5 seats including the driver seat.
2. Each car will have a fixed speed of 30 km/h.
3. The intended arrival time of each participant will be fixed to 30 minutes after their intended departure time.
4. The maximum earliness waiting time for each request cannot exceed 10 minutes (QOS constraint).
5. After a car departs from a node, it cannot move back to that node directly. However, if let's say a car moves from node A to node B and then back to node A again, this is acceptable.
6. The origin and destination of a request must not be the same.
7. The number of participants a car can take cannot exceed its capacity.
8. The request of each participant will only be served by one car.
9. The origin and destination of a participant's request will only be served by one car.
10. Drivers who are specially assigned to serve requests will drive the car back to the driver's location before serving the request after serving the request.
11. The request serving sequence of a car should be arranged according to the intended start time of each request, from earlier to later.
12. Drivers with a request must serve the request of all passengers in the car first before going to the driver's own destination.
13. Only participants' requests with an intended start time difference within a 12-minute range will be served together. Or else the request of a participant will only be served after the request of another participant was served.
14. All participants must be able to drive.
15. Each node will have participants on it.

3.2.2 Initial Solution Generating Algorithm

```
Pseudocode for initialization
Inputs:
    • sol - Solution object containing lists of cars (CList) and requests (req)
Outputs:
    • A vector of pointers to car objects (CList)
Initialize variables:
    • count = 0 (Index for iterating through CList do distribute the requests evenly)
Loop through each request in sol.req:
    1. Add request to the current car's service list and sort the service list according
       to the starting time of each request in ascending order:
          o sol.CList[count] adds the request sol.req[i] to its service list.
    2. Update count:
          o If count + 1 equals the size of CList
                1. reset count to 0.
          o Else
                1. increment count by 1.
Loop through each car in sol.CList:
    1. Generate the route schedule for each car:
          o Each car outputs its route to "initial_solution.txt".
Return sol.CList
```

The algorithm above assigns the requests evenly to each car. An initial solution will be deemed feasible even if it had unserved requests or lateness waiting time.

3.2.3 Local Search Algorithm

```
Pseudocode for localSearch
Inputs:
    • sol - Solution object containing a list of cars (CList) and requests
    • dynam - Boolean flag to indicate whether the current scenario is dynamic
Outputs:
    • A vector of car objects (CList)
Initialize random generator:
    • Create a random floating point number generator that range from 0 to 1
Generate a random choice:
    • choice = generate random number
Determine the local search operation based on choice:
    1. If choice is lesser than or equal to 0.5:
        o Call swapRequest() – swap one request between two car
        o Update sol.CList with the result of swapRequest
    2. Else:
        o Call moveRequest() – move one request from a car to another car
        o Update sol.CList with the result of moveRequest
Return sol.CList
```

The random number generator is used to determine the probability of either choosing to swap requests or to move requests. Both the swapping requests and moving requests will have a 50% probability of being chosen.

3.2.4 Solution Searching and Optimization Algorithm

Pseudocode for Solution optimization

```
Inputs:
    • s - Initial solution object
    • initial - Initial solution object
    • loopCount - Maximum number of loops = 5
    • Iter - Maximum number of iterations of not updated solution quality to enter
      into the next loop or converge = 35
    • w - Vector of weights for inconvenience cost calculation
    • optim - Boolean flag for displaying the optimization process
Outputs:
    • best - Best solution found
Initialize variables:
    • ii = 0 (loop counter)
    • solList (a vector with a size of 5) = initialized with 5 copies of initial in the first
      loop, initialized in the following loops with 5 copies of the best solution obtained
      from the previous loop
    • best = initialized with the initial solution in the first loop, initialized in the
      following loop with the best solution obtained from the previous loop
    • temperature (a floating point number) = 7
    • ver = empty list of vectors to store metrics
    • best_score = the inconvenience cost of the best solution
Loop until ii = loopCount:
    1. Break if initial fitness is 0
    2. Clean and reinitialize the solution s
    3. Output the score of initial solution
    4. Initialize variables:
        o counter = 0
        o iter = 0
    5. Start the main loop until counter = Iter:
        a. Increment iter with 1
        b. For each solution sol in solList:
```

```
        o Clear the solution record of sol (all records apart from the requests will
          be cleared)
        o Copy sol to a solution object curr
        o Run solution searching function on curr to update the CList of curr
        o Generate routes for each car from the requests
        o Recalculate metrics and inconvenience cost for curr
        o If cur is better than sol
            - update sol
        o Else
            - generate a random floating point number from 1 to 10
            - If the number is larger than the temperature
                ▪ update sol
      c. Increase temperature by 1 after every 15 iterations
      d. Sort solList by inconvenience score in ascending order
      e. If new best solution is found:
        o Check if the generated schedule complies with the quality of service
          constraints
        o If yes
            - update best, store metrics in ver, initialize counter to 0
            - Sort ver by inconvenience score in ascending order
        o If no
            - initialize counter to 0
      f. Increment counter if no new best solution is found
      g. Limit ver to top 3 ranked solutions
      h. Output the optimization process if optim flag is true
      i. Clean up memory by deleting routes in solList
   6. End of main loop
   7. Output final results and print metrics
Return the best solution found after all iterations and loops
```

This optimization algorithm will run 5 loops, and each loop will end if the quality of the best solution is not updated for 35 iterations. The vector of solutions is initialized with the initial solution when the optimization begins, which means the first loop, and will be initialized with the current best solution after each loop ends. This is to let the algorithm start the search from the current best solution, in the hope that the quality of the solution neighbourhood around the current best solution is better. This is similar to the idea of neighbourhood updating in the ant colony optimization algorithm [24] , but the updating mechanisms are very different.

The probability of accepting a worse solution will be increased every 15 iterations, and after the 45th iteration, the worse solution will not be accepted. But at the beginning of every loop, the temperature will be reset to 7 again. This is to let the algorithm continue doing global search in the solution space around the current best solution to have a chance of finding a better solution. Solutions that do not comply with the quality-of-service (QOS) constraint will not be rejected, as the algorithm might find better solutions from them. But to ensure that the QOS constraint is met, the best solution will only be updated if the better solution does not violate the constraint.

3.2.5 Lateness Avoiding

To avoid lateness waiting time in our schedule as it is very intolerable, a lateness waiting time rollback mechanism has been implemented. The figure below visualizes the rollback mechanism.

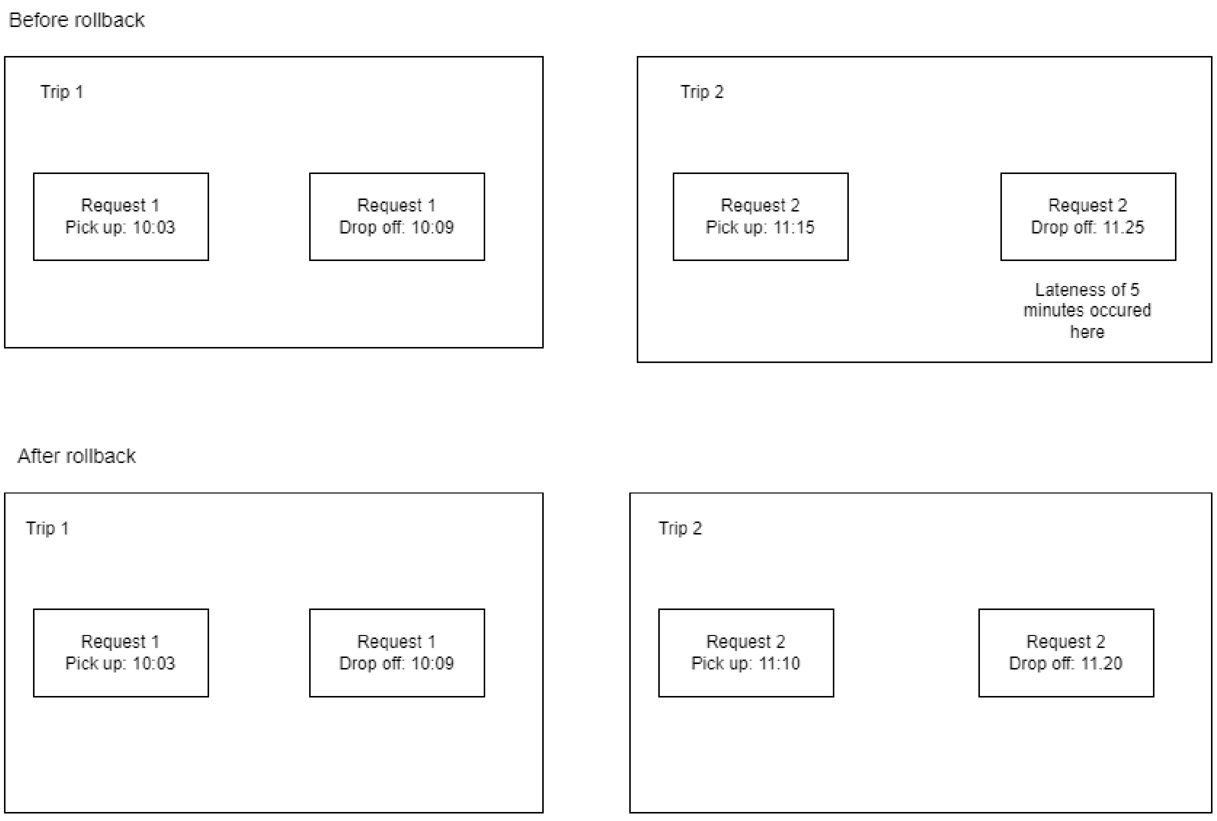


Figure 3.2 Lateness Waiting Time Roll Back Mechanism

We define a trip as from a driver on board the car until the driver finishes serving requests and leaves the car. As we can see from the diagram above, if after rolling back the pickup time of the first request of the current trip is not earlier than the drop off time of the last request of the previous trip, the rollback will only happen on the current trip. Or else, the rollback will also happen to the previous trip.

3.2.6 Output Format

The output format of the route schedule to a text file will be as below:

Car ID: {time; node; D: driver type; number of passengers} → { time; node; D: driver type; number of passengers } → so and so forth
Car ID: so and so forth

Highest single request earliness waiting time: x minutes
Total earliness waiting time for all participant: y minutes
Total lateness waiting time for all participant: z minutes
Total travel expenses for all participant: RM a
Total number of darp-like cases: b

The route schedule for each car will be printed out in this way. This route schedule shows when will a car reach a certain node, and at that moment who is the driver and who is the passenger. The pick-up and drop-off of passengers will all be reflected in this route schedule by adding and removing passengers from the passenger list. The total DARP-like case denotes the total number of times drivers are assigned specially to serve some requests.

3.3 Dynamic Scenario Handling

Pseudocode for Dynamic Scenario Handling

```
Inputs:
    • request - The incoming dynamic request
    • current_time - The current system time
    • curr_solution - The current solution
Check if the request arrives after 5 pm:
    • If request.arrival_time > 5:00 PM:
        o Reject the request
        o Return the current request
Check if the request's start time is within 1 hour from the current time:
    • If request.start_time ≤ current_time + 1 hour :
        o Reject the request
        o Return the current request
If the request is accepted:
    • Sort the car list by inconvenience cost
    • Sort solution.CList in ascending order based on the inconvenience cost of each car
    • Insert the dynamic request:
        o Insert the request into the car with the lowest inconvenience cost (first car in the sorted list)
        o Recalculate the inconvenience cost for the new solution
        o Compare the new solution with the current solution
        o If new_solution.inconvenience_cost < curr_solution.inconvenience_cost :
            ▪ If the new solution meets the QOS constraints:
                - Output the new solution as the best solution
            ▪ Else:
                - Enter solution re-optimization
        o Else:
            ▪ Reoptimize the new solution using the solution optimization algorithm from the static scenario
```

- *Output the newly optimized solution*

Return the new solution

In the dynamic scenario, the requests will be removed from the solution after it is being served, and new dynamic requests might be added in. So, we will need to compare the quality of two solutions with different numbers of requests. It is normal for a solution with more requests to have a higher cost than the one with fewer requests. Thus, to ensure fairness during solution quality comparison, the cost of the solution will be divided by its current number of requests. The inconvenience cost of each car will be calculated by cumulating the cost of the requests assigned to that car. The dynamic request will be inserted directly into the car with the lowest cost, and that car will make a detour to serve that request.

Chapter 4

Experiment and Simulation Setup

4.1 Setting Up

4.1.1 Hardware Setup

The hardware involved in this project is a laptop. Table 3.1 shows the specifications of the laptop used in this project.

Table 4.1 Specifications of Laptop

Description	**Specifications**
Model	Acer Nitro 5 2022 AN515-45
Processor	AMD Ryzen 7 5800H
Operating System	Windows 11
Graphic	NVIDIA GeForce GTX 1650
Memory	16GB DDR4 RAM
Storage	500GB SSD

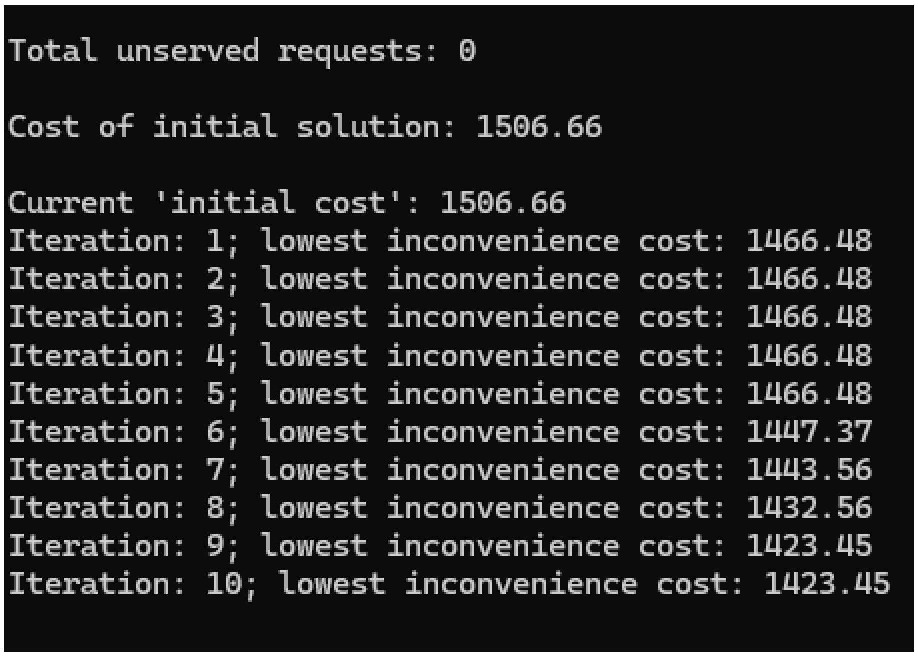
4.1.2 Software Setup

The software system of this project is fully coded using C++. The software tools required in this project are a C++ integrated development environment (IDE) and Windows PowerShell for collecting experiment data. Below are the software specifications used in this project.

- IDE: Visual Studio Enterprise 2022
- C++ version: ISO C++ 20 Standard
- Windows PowerShell version: 5.1

4.2 Running the Program

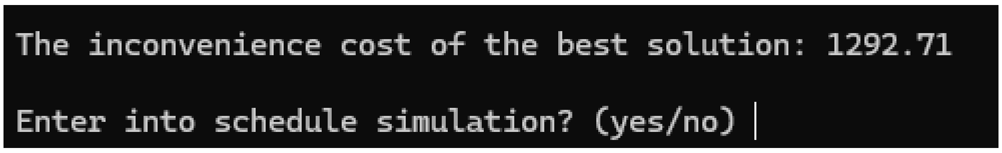
After running the program, it will randomly generate test cases, generate the initial solution from the test cases, and do solution searching and optimization all in one run. The program will stop temporarily after the solution searching and optimization algorithm converges.



```
Total unserved requests: 0

Cost of initial solution: 1506.66

Current 'initial cost': 1506.66
Iteration: 1; lowest inconvenience cost: 1466.48
Iteration: 2; lowest inconvenience cost: 1466.48
Iteration: 3; lowest inconvenience cost: 1466.48
Iteration: 4; lowest inconvenience cost: 1466.48
Iteration: 5; lowest inconvenience cost: 1466.48
Iteration: 6; lowest inconvenience cost: 1447.37
Iteration: 7; lowest inconvenience cost: 1443.56
Iteration: 8; lowest inconvenience cost: 1432.56
Iteration: 9; lowest inconvenience cost: 1423.45
Iteration: 10; lowest inconvenience cost: 1423.45
```

Figure 4.1 Start of Running Demo of The System

As shown in Figure 4.1, during the solution searching and optimization process, the current lowest inconvenience cost will be printed out every iteration for us to view the optimization process.



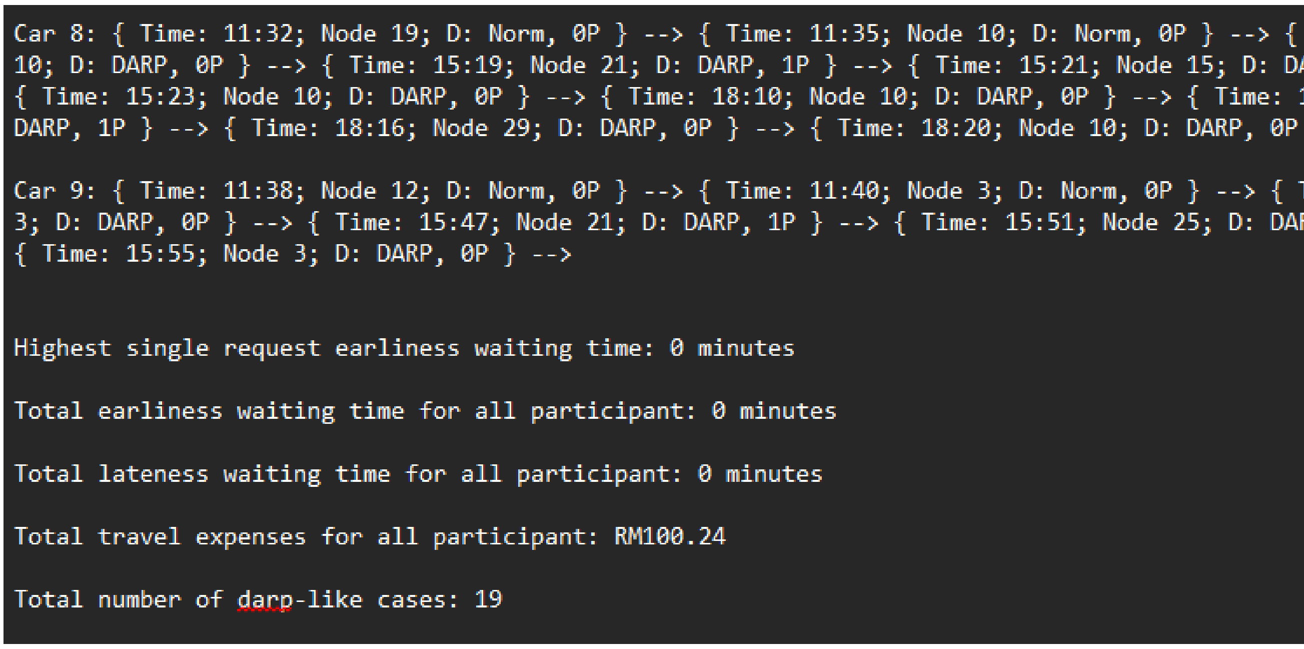
```
The inconvenience cost of the best solution: 1292.71

Enter into schedule simulation? (yes/no) |
```

Figure 4.2 End of Running Demo of The System

After the algorithm converges, the system will stop running. The final lowest inconvenience cost will be outputted, as shown in Figure 4.2 above. If users type “yes” on the terminal, the program will enter into schedule simulation, or else the program will exit.

4.2.1 Solution

Both the initial and the best solution will be outputted into a text file. Figure 4.4 shows the format of the output. The format is the same for both the initial and the best solution.



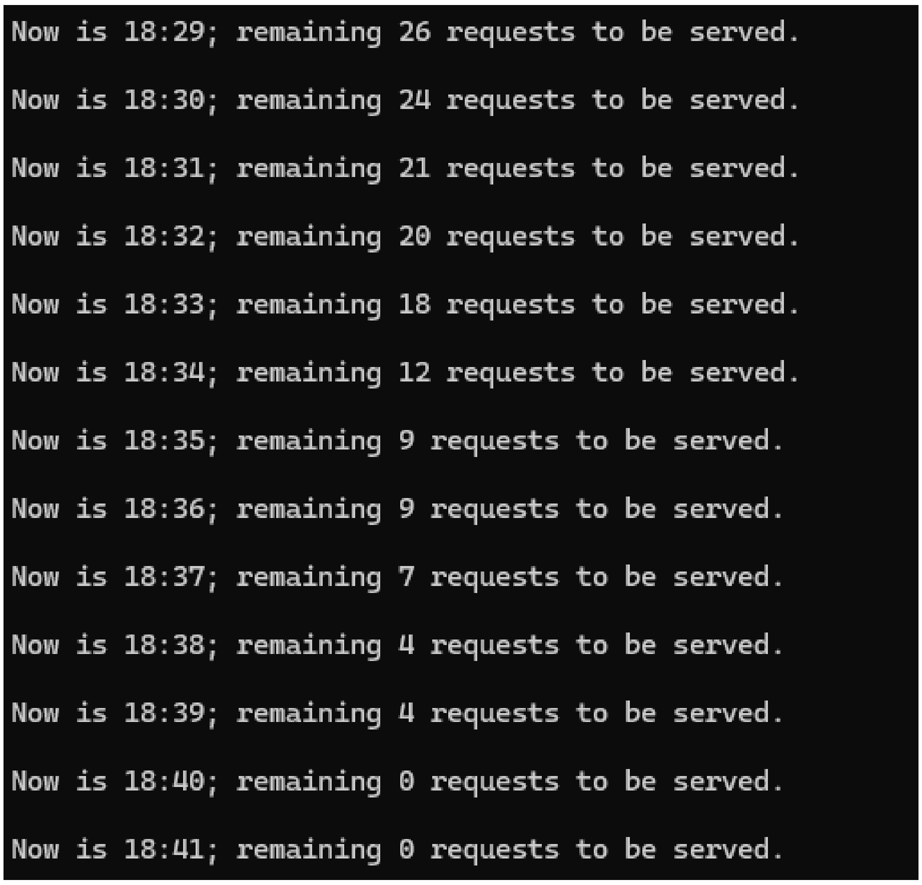
```
Car 8: { Time: 11:32; Node 19; D: Norm, 0P } --> { Time: 11:35; Node 10; D: Norm, 0P } --> {
10; D: DARP, 0P } --> { Time: 15:19; Node 21; D: DARP, 1P } --> { Time: 15:21; Node 15; D: DA
{ Time: 15:23; Node 10; D: DARP, 0P } --> { Time: 18:10; Node 10; D: DARP, 0P } --> { Time: 1
DARP, 1P } --> { Time: 18:16; Node 29; D: DARP, 0P } --> { Time: 18:20; Node 10; D: DARP, 0P

Car 9: { Time: 11:38; Node 12; D: Norm, 0P } --> { Time: 11:40; Node 3; D: Norm, 0P } --> { T
3; D: DARP, 0P } --> { Time: 15:47; Node 21; D: DARP, 1P } --> { Time: 15:51; Node 25; D: DAF
{ Time: 15:55; Node 3; D: DARP, 0P } -->


Highest single request earliness waiting time: 0 minutes

Total earliness waiting time for all participant: 0 minutes

Total lateness waiting time for all participant: 0 minutes

Total travel expenses for all participant: RM100.24

Total number of darp-like cases: 19
```

Figure 4.3 Output of The Solution

The “D: Norm” indicated a normal case, where the driver is also serving the driver’s own request when driving the car. The “D: DARP” indicated a special driver assignment case, where the driver is assigned solely to serve some participants’ requests. The “xP” indicates the number of passengers excluding the driver on the car when the car reaches a node, where x is the number of passengers.

4.2.2 Simulation of the Schedule

To simulate the run of the schedule, first, the program will run a simulation clock starting from 8 am to 6.40 pm. If the end time of any request is reached, which means its end time is the same as the current time on the simulation clock, the request will be removed from the car serving it. This indicates that the request was served. The number of requests left to be served will be outputted every minute so that we can see the changes in the number of requests, and make sure that the simulation is running fine. When the simulation clock ends, the program will calculate the total number of requests again to make sure all requests are served. Figure 4.5 shows the screenshot of the simulation process.



```
Now is 18:29; remaining 26 requests to be served.

Now is 18:30; remaining 24 requests to be served.

Now is 18:31; remaining 21 requests to be served.

Now is 18:32; remaining 20 requests to be served.

Now is 18:33; remaining 18 requests to be served.

Now is 18:34; remaining 12 requests to be served.

Now is 18:35; remaining 9 requests to be served.

Now is 18:36; remaining 9 requests to be served.

Now is 18:37; remaining 7 requests to be served.

Now is 18:38; remaining 4 requests to be served.

Now is 18:39; remaining 4 requests to be served.

Now is 18:40; remaining 0 requests to be served.

Now is 18:41; remaining 0 requests to be served.
```

Figure 4.4 Simulation of the Best Solution From the Static Scenario

This simulation will only cover the static case. The simulation that includes dynamic cases will be done in Chapter 5 to evaluate the ability of the program to handle dynamic scenarios.

4.3 Experiment Configuration

Parameter of the solution searching algorithm in the static scenario:

- Number of loops: 5
- Number of iterations of not updated solution quality to converge, per loop: 35
- Starting temperature: 7
- Temperature increment rate: increment by 1 per 15 iterations

Parameter of the solution searching algorithm in the dynamic scenario:

- Number of loops: 1
- Number of iterations of not updated solution quality to converge, per loop: 35
- Starting temperature: 7
- Temperature increment rate: increment by 1 per 15 iterations
- Optimization objectives are the same as static scenarios
- Same constraint as the static scenario

Planning horizon:

From 8 am to 6.40 pm. The earliest request cannot start before 8 am, and the latest request cannot end after 6.40 pm.

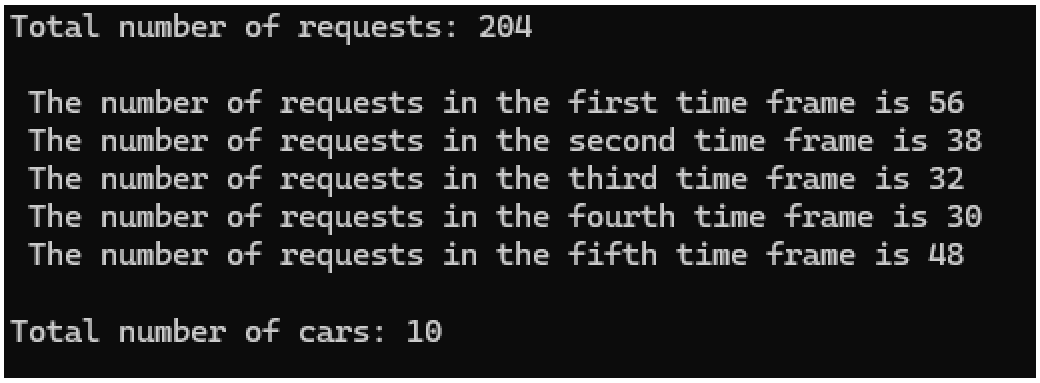
Test cases:

To better represent the university scenario, the requests will be generated at a different probability in the different time frames within the planning horizon. The total number of requests will be set before the generating of requests, so the probability means the likelihood of the generated requests will fall in that time frame.

The probability of generating requests for different time frames is as follows:

- 8 am to 9 am: 25%
- 10 am to 11 am: 17.5%
- 12 pm to 1 pm: 10%
- 2 pm to 3 pm: 17.5%
- 4 pm to 6 pm: 25%

This means that if now we have 200 requests, the probable outcome will be: 50 requests in 8 – 9 am, 35 requests in 10 – 11 am, 20 requests in 12 – 1 pm, 35 requests in 2 – 3 pm, 50 requests in 4 – 6 pm. The sample outcome of this request-generating mechanism is shown in Figure 4.5.



```
Total number of requests: 204

 The number of requests in the first time frame is 56
 The number of requests in the second time frame is 38
 The number of requests in the third time frame is 32
 The number of requests in the fourth time frame is 30
 The number of requests in the fifth time frame is 48

Total number of cars: 10
```

Figure 4.5 Different Probability of Request Generating in Different Time Frame

Likewise, the occurrence of the dynamic requests is also in a different probability in different time frames across the planning horizon. But this probability is purely the likelihood of the occurrence of the dynamic requests, as the total number of dynamic requests is not set. **The probabilities are as such:**

- 8 am – 10 am: 50%
- 11 am – 1 pm: 40%
- 2 pm – 4 pm: 30%
- 5 pm – 6 pm: 20%

The probabilistic requests generation mechanism will be run once every three minutes of the simulation clock. If the mechanism hits the probability of generating dynamic requests, 1 dynamic request will be generated.

The test cases also include cars and maps. The number of cars will be fixed to 10, and each car will be assumed to be at the starting point of the first requests assigned to them before the schedule starts to run. The maps used in the experiment are randomly generated.

The generated maps have the following characteristics:

- Consist of 30 nodes
- Each node has 4 to 8 adjacent nodes
- The distance between each node with each other is between 0.35 km to 3 km

4.3.1 Objective of Experiment

1. To prove that the solution searching and optimization algorithm is working fine
2. Stress test the program
3. To tune the parameters of the algorithm
4. To show that the dynamic scenario handling mechanism is working fine

4.3.2 Experiment Method

To meet experiment objective 1, the algorithm will be run with two different weight configurations. The metrics of the top three ranked solutions from both configurations will be recorded and tabulated. Both configurations will be run with test cases consisting of 200 to 300 requests.

Next, to fulfill experiment objective 2, the program will be tested with test cases consisting of various ranges of number of requests to determine the highest range of number of requests the program can take. For example, the test cases will have 0 to 100 requests and 100 to 200 requests, and they will keep on increasing until the program cannot handle them.

From the result of the test on objective 2, we will then try out different weight configurations on the earliness weight (as the travel expenses weight had been fixed to 1) to find the best weight configuration. Each configuration will be tested 500 times with different ranges of request sizes the program can handle, and the result will be presented on charts. We will use shell scripts to help run the test in parallel to save time. The scripts will be run on Microsoft PowerShell.

For dynamic scenario handling, we will show that the program can reject and accept dynamic requests properly. The rejected requests should not be inserted into the schedule, and the accepted requests must be inserted into the schedule and be served. Other than that, we will show that the request-locking mechanism is working fine. In our scenario, there is at least one hour of reaction time, so the speed of re-optimization may not be crucial. But we would still show the average time the new solution can be re-optimized to ensure enough reaction time.

4.4 Implementation Issues and Challenges

There are three main issues and challenges faced when implementing this model.

First, as there is a QOS constraint in our program, sometimes the algorithm may not be able to find a better solution that meets the QOS requirement. The algorithm will then output the initial solution as the only feasible solution. This problem is likely to happen when the number of requests is high. From our observations, this will happen when the total number of requests is more than 400, and the frequency of happening will increase when the number of requests increases.

Second, the dynamic scenario can only be simulated using test cases with at most 150 requests. This is because there is a memory leaking problem when the algorithm runs, so a higher number of requests may result in memory allocation error. The memory leaking problem happens most likely because the objects in the program are not being defined or constructed properly when we start building the program. The source of leaking is hard to detect at this stage, and changes may need the whole revamp of the program, so we will only use test cases with 150 requests or less to conduct the dynamic scenario simulation.

Chapter 5

System Evaluation

5.1 Static Scenario

The results of the three experiments done in the static scenario to test the ability of the algorithms and tune the weight of the objective function will be displayed and discussed in the below sub-sections.

5.1.1 The Working of the Solution Searching and Optimization Algorithm

We conduct the test with earliness weight = 5 and earliness weight = 4. Table 5.1 and Table 5.2 record the metrics of the top three tanked and initial solutions for both weight configurations.

Table 5.1 Test Result for Weight Configuration 1

Weight of earliness: 5					
Solution rank	**Inconvenience cost**	**Total earliness waiting time**	**Total DARP-like cases**	**Maximum single request waiting time**	**Total traveling expenses**
1	1199.99	54 minutes	161	5 minutes	RM929.99
2	1209.81	57 minutes	160	5 minutes	RM924.81
3	1234.33	65 minutes	162	7 minutes	RM909.33
Initial	1464.47	66 minutes	235	8 minutes	RM1134.47
Number of requests: 277					

Table 5.2 Test Result for Weight Configuration 2

Weight of earliness: 4					
Solution rank	**Inconvenience cost**	**Total earliness waiting time**	**Total DARP-like cases**	**Maximum single request waiting time**	**Total traveling expenses**
1	1349.11	86 minutes	160	8 minutes	RM945.11
2	1379.79	101 minutes	159	8 minutes	RM939.79
3	1401.48	110 minutes	174	8 minutes	RM969.48
Initial	1560.02	108 minutes	248	8 minutes	RM1216.02
Number of requests: 300					

From the experiment result, we can see that across different weight configurations and different numbers of requests, the solution searching and optimization algorithm can tell which solution is better in both scenarios, as the best solution is better than the initial solution, and the algorithm can rank the solutions according to their quality. The QOS constraints are also met in every solution. These show that the algorithm is working fine.

5.1.2 Stress Test

The stress test will use five request size ranges: 0-100, 100-200, 200-300, 300-400, and 400-500 requests. The maximum number of requests is set at 500 because the algorithm's ability to find optimal solutions visibly degrades when the request size exceeds 400. To determine whether weight configurations affect the algorithm's ability to handle larger request volumes, we will test the cases using three different weight configurations. We will then analyze the trends in travel expenses and earliness waiting time across the test cases with varying request sizes to evaluate how the algorithm's performance changes as the number of requests increases. As comparison across different request size ranges will be made in this test, all the metrics will be per request rather than per solution to ensure fairness in comparison, as it is normal for the solution with the higher number of requests to have a higher value in each metric.

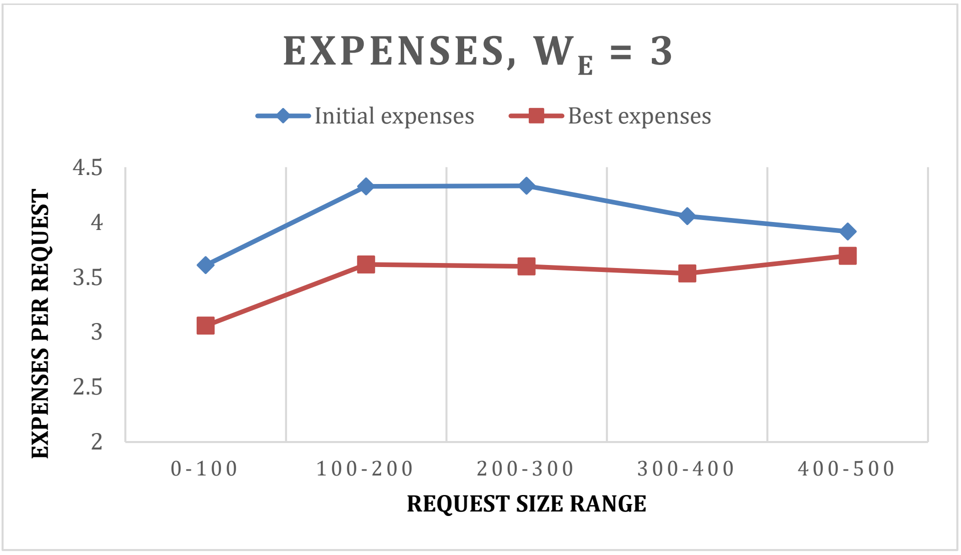


Figure 5.1 Average Travel Expenses per Request, $W_E = 3$

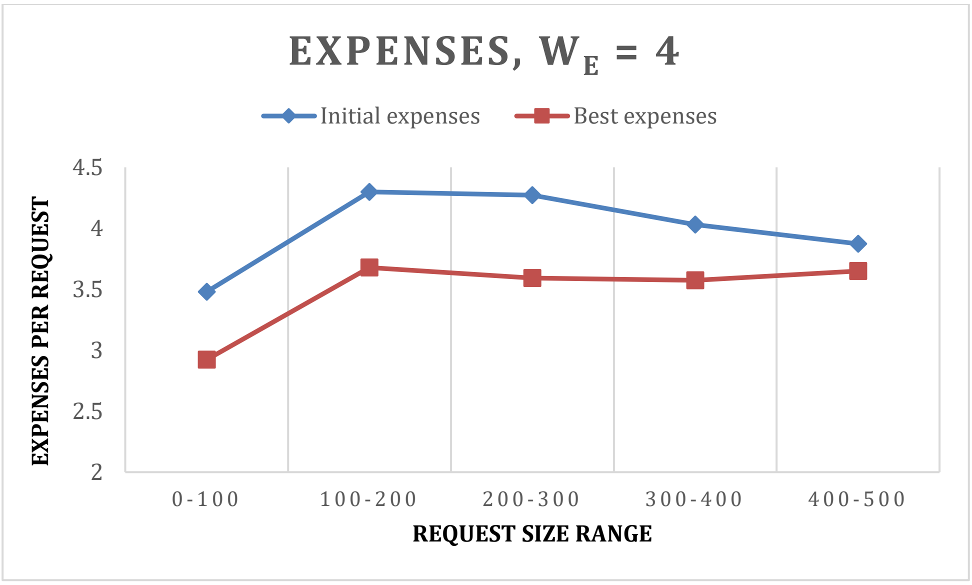


Figure 5.2 Average Travel Expenses per Request, $W_E = 4$

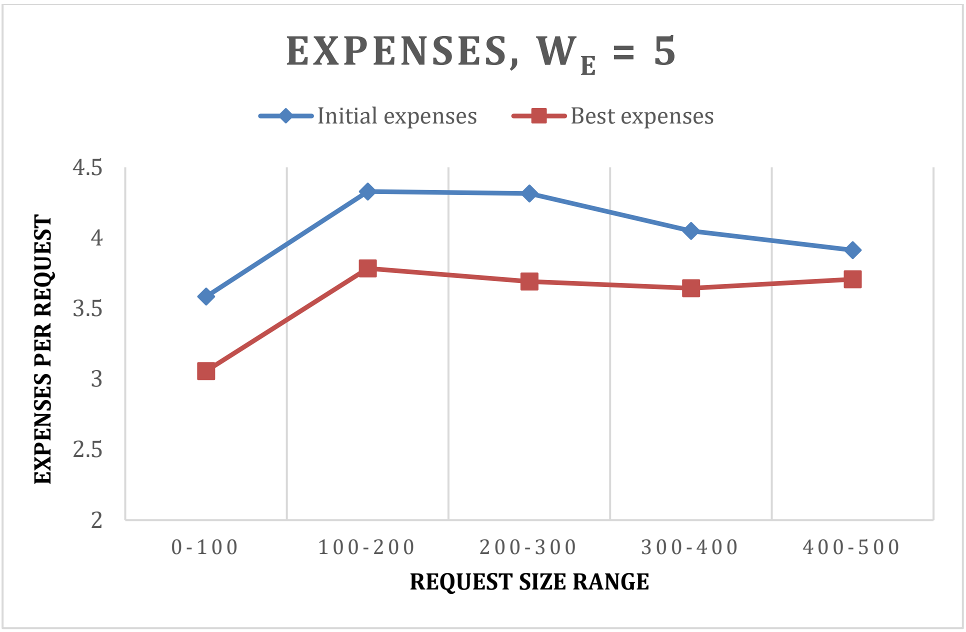


Figure 5.3 Average Travel Expenses per Request, $W_E = 5$

Figures 5.1 to 5.3 show that the traveling expenses per request for both the initial and the best solutions will increase when the request size range increases from 0-100 to 100-200 but will slightly decrease as the number of requests increases thereafter. The gap between the initial and best solutions also increases as the number of requests increases to 300 but decreases thereafter until there is only a little gap left when the request size range is 400-500.

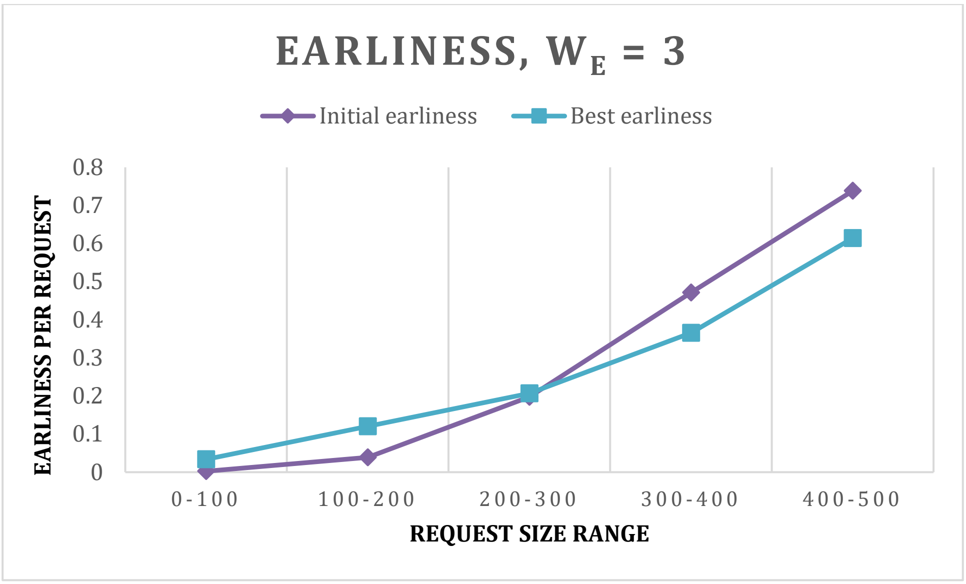


Figure 5.4 Average Earliness Waiting Time per Request, $W_E = 3$

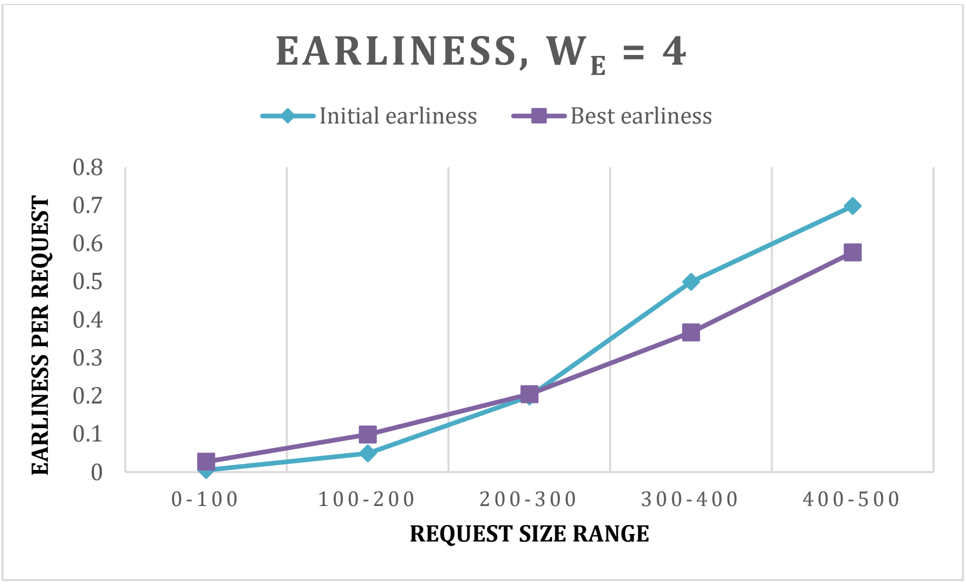


Figure 5.5 Average Earliness Waiting Time per Request, $W_E = 4$

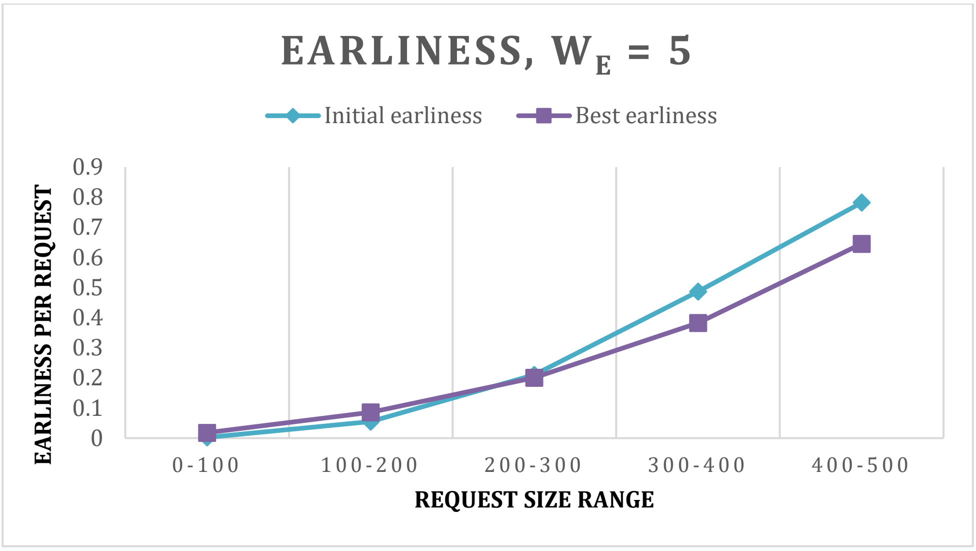


Figure 5.6 Average Earliness Waiting Time per Request, $W_E = 5$

Figures 5.4 to 5.6 show that the earliness waiting time per request for both the initial and the best solutions will increase as the number of requests increases. The earliness waiting time of the best requests is higher than the initial solutions when the request size range is less than 200. It will become almost the same for the request size range of 200-300, and the best solutions will have a lower earliness waiting time than the initial solutions when the request size range is more than 300, and the gap between them will increase as the number of requests increases.

Explanation:

The increase in traveling expenses when the number of requests increases to 200 may be caused by the increasing complexity of the problem as there are more requests. After that, the traveling expenses decrease as the number of requests increases. This may be because when there are more requests, the algorithm can cluster requests with nearby starting or ending points together more efficiently, thus reducing the distances of detours and the number of detours, so the traveling cost will decrease.

The earliness waiting time increases as the number of requests increases because the problem will be more complex when there are more requests. As for the gap between the initial and the best solutions, when the request size is less than 300, the optimization space for the traveling expenses may be larger than the earliness waiting time, so it may be easier for the algorithm to sacrifice the earliness waiting time to find the best solution with significantly less traveling expenses than the initial solution to achieve a lower inconvenience cost. Another reason will be the total traveling expenses per solution will be several hundred or more than a thousand, while the earliness waiting time at most will only exceed one hundred. So even if the weight on the earliness waiting time is much higher, the solution may still find it easier to find a solution with a lower cost by just focusing on reducing the traveling expenses.

When the number of requests is increased further, and the optimization space for the traveling expenses decreases, to achieve a lower inconvenience cost, the algorithm will need to shift focus to the earliness waiting time with a higher weight on it. This may explain the trend of the gap between the initial and the best solutions for both traveling expenses and earliness waiting time when the number of requests increases.

5.1.3 Weight Tuning

For a university student or staff, wanting them to wait at their starting point earlier than intended will be more intolerable than a slightly higher cost. In the weight tuning test, we intend to find the weight configuration that ensures the best optimization strength on the earliness waiting time while also optimizing the expenses.

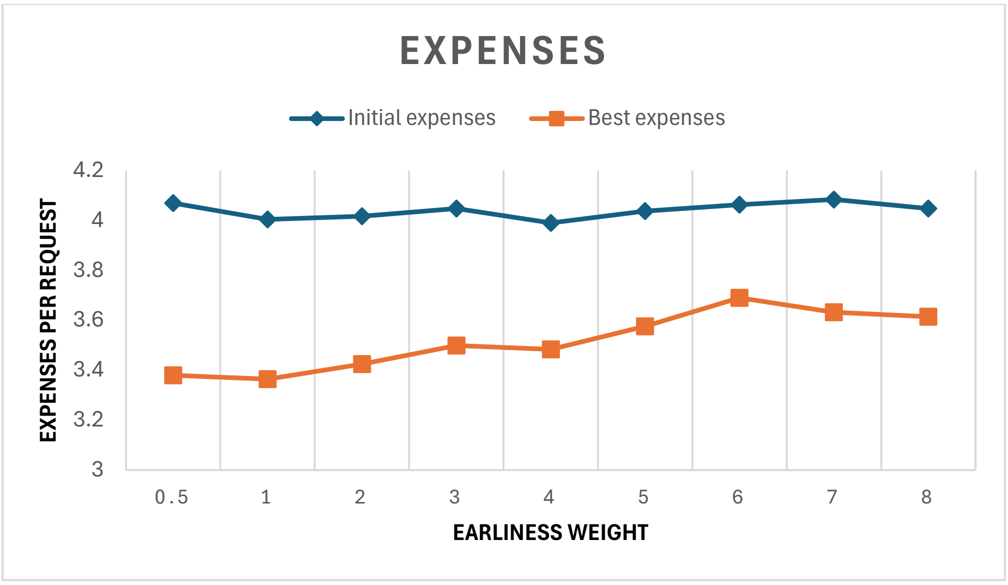


Figure 5.7 Average Travel Expenses Per Request in Different Weight Configurations

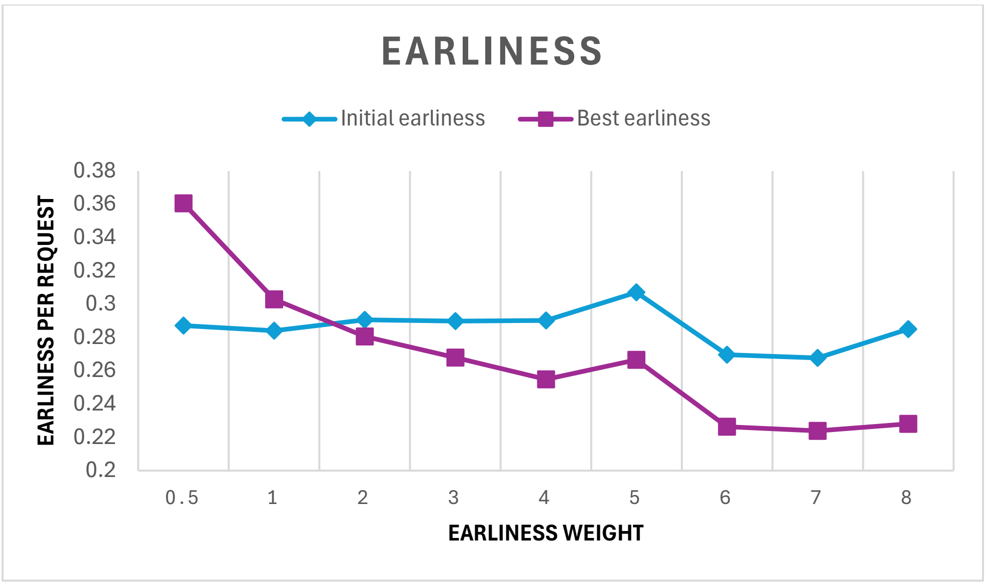


Figure 5.8 Average Earliness Waiting Time Per Request in Different Weight Configurations

The optimization strength on the expenses decreases slightly as the weight ratio increases. This is because as the weight of the earliness waiting time increases, the focus of optimization will shift toward the earliness waiting time.

The optimization strength on the earliness waiting time is increasing as the weight of earliness increases. As we can see from Figure 5.8, the earliness waiting time of the best solution spikes high when the weight is 0.5, which means when the priority is given to the expenses. After that when the priority is given to the earliness waiting time, the earliness of the best solutions starts to decrease as more priority is given, typified by the higher weight.

The expenses of the best solutions, when prioritizing earliness above traveling expenses, are still being optimized properly, albeit the optimization strength is not that strong compared to when expenses and earliness are given equal priority. The reason we make the weight of earliness changeable is also because more priority will be given to the lower earliness waiting time. Thus, we can focus on the optimization strength of the earliness to determine the weight to be chosen.

Table 5.3 Comparison of the Reduction Percentage of Two Weight Settings on the Earliness Waiting Time

Weight	**Initial earliness**	**Best earliness**	**Reduction %**
4	0.2902	0.2548	12.2
5	0.3071	0.2666	13.19
6	0.2697	0.2263	16.09
7	0.2677	0.2239	16.36
8	0.2851	0.2282	19.96

As shown in Table 5.3, the reduction percentage of the best earliness compared to the initial earliness of weight 8 is higher, showing a stronger optimization strength. So, the weight of the earliness waiting time is set to 8.

The number of iterations will be determined by observation. Throughout the experiments done in this project, if the solution quality is not updated for more than 30 iterations, the algorithm will not find a better solution even though it is being run 100 iterations after that. The algorithm's temperature will reach 10 after 45 iterations, regardless of the solution quality. Therefore, setting the parameter too high is not ideal, as the algorithm will primarily perform local searches after the 45th iteration. At that point, the search space becomes quite limited, reducing the chances of finding a better solution. This parameter setting will prevent the algorithm from spending unnecessary time and computing resources.

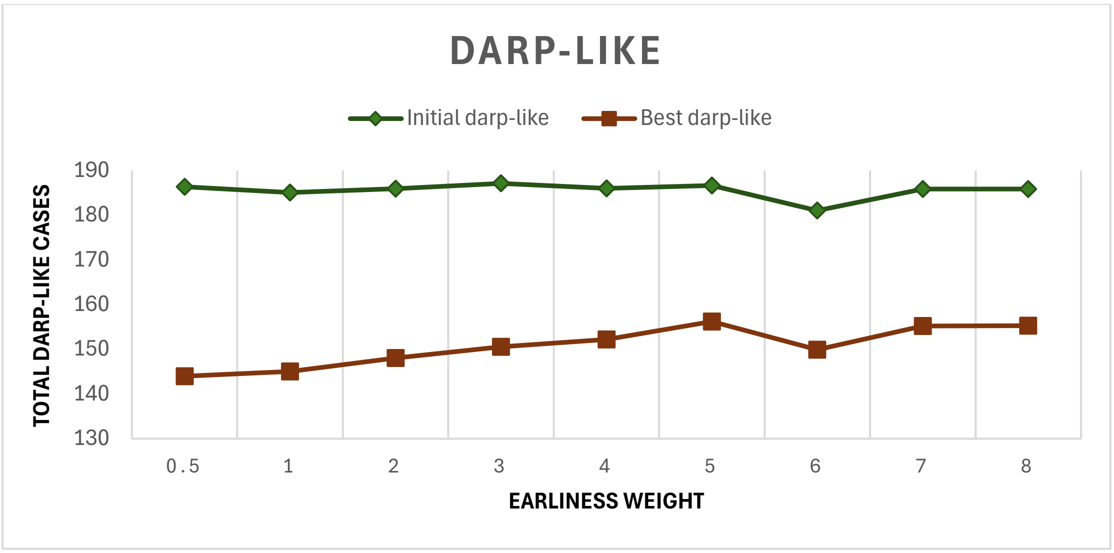


Figure 5.9 Total DARP-like Cases in Different Weight Configurations

Figure 5.9 shows that the DARP-like cases, which means special driver assignment cases will be optimized even if not set as an individual objective function. By optimizing the traveling expenses, the number of DARP-like cases will also be optimized.

5.2 Dynamic Scenario

For the dynamic scenario, we are interested in finding out the handling mechanism's ability to reject and accept requests based on predefined time constraints, the time needed to re-optimize the solution, and the EDOD of the dynamic scenario.

As seen in the pseudocode for the dynamic scenario handling in Chapter 3, the dynamic requests that do not meet the time constraints will be directly rejected, and the accepted requests will be inserted into the service list of a car. This mechanism works fine as throughout our tests to determine the average re-optimization time and the EDOD, the requests that should be rejected are all being rejected, and the requests that should be accepted are all being accepted. Using the dynamic requests generating setting in Chapter 4, the average number of dynamic requests the algorithm accepts for each run is 50.25 requests.

For the re-optimization time, we will use test cases with 100-150 requests and will run the dynamic scenario simulation for 100 times. The average time needed is 27.0125 seconds. The re-optimization can be done quickly in our experiments because the total number of requests is low, as according to our observation of the algorithms, they tend to use more time to complete their search when the number of requests increases.

The average EDOD throughout the test is 0.1867. As it is lower than 0.3, the dynamic scenario will be classified as a weak dynamic scenario. The EDOD is low because the probability of the dynamic requests generating will be lower as the virtual time goes nearer to the end of the planning horizon, and requests that come in 1 hour before the planning horizon ends will be rejected. This gives the algorithm enough time to handle the requests.

5.3 Concluding Remarks

The working effectiveness of the algorithm:

- Working fine as the algorithm can rank the solutions it found based on their quality, and the best solution found is better than the initial solution.

The final parameter value of the algorithm:

- Weight of earliness: 8
- Number of iterations of not updated solution quality to converge, per loop: 35

The findings from the stress test:

- The ability of the algorithm to find the optimal solution will visibly degrade when the number of requests exceeds 400.

Dynamic scenario:

- An average of 50.25 dynamic requests are accepted per test
- The algorithm can accept and reject dynamic requests accurately according to whether they meet the time constraints
- EDOD: 0.1867, indicating the dynamic scenario studied in this project is a weak dynamic scenario

Chapter 6
Conclusion and Recommendation

6.1 Conclusion

Nowadays in the university, as most students and staff have cars, congestion can happen in the university. The timetable for students or staff is not fully full throughout the day, so they might leave the university if there is still some time before their class. Therefore, the congestion in the university will not only happen in classic peak hours, but it can happen anytime. Even though public transport will be provided, usually it will not stop at every block of the university, thus is not the most convenient for students and staff. Moreover, the increasing number of cars will cause the parking space in the university cannot accommodate all cars. Some will have no place to park, and some will need to park far away from their destination. They will need to walk under the hot sun to their destination, and the situation will be worse when it is raining. Other than that, due to the bad driving attitude of drivers, it is quite high risk for bikers to ride in the university, and sometimes accidents may happen to them. The university scenario consists of the static part and the dynamic part, where the timetable of the students and staff will generate static requests, and the sudden need to go to the university due to various reasons such as additional classes and urgent meetings will generate dynamic requests.

This project aims to formulate the problems above and develop a model that will come out with a vehicle pickup and drop-off schedule. A solution searching and optimization algorithm will be built to optimize the schedule to better meet the passengers' and drivers' objectives. A dynamic scenario handling mechanism will be built to handle dynamic requests. We attempt to combine some aspects of DARP and CPP in a hybrid way to formulate the problems that best suit the university setting of our problems. The outcome of this project will be a runnable program to generate the best solution from the given test cases.

Based on literature reviews on past works, the exact way to formulate the problems and the exact algorithm to use for solution searching and optimization is determined. The problems will be formulated as DARP-based, meaning the schedule is only for one day. Unlike the classic DARP setting, the participants can both be passengers or drivers, meaning that university students and staff can serve themselves. No additional full-time drivers are needed; thus, this formulation approach is more feasible in the university setting. The objective of the model is to reduce the total costs, to reduce the total earliness waiting time, and to reduce the number of cases where drivers are specially assigned to serve some requests. Drivers will be specially assigned when there is currently no driver on the car and the car is not at the starting point of its next assigned request. This will be a multi-objective, many-to-many, static and dynamic model.

In this project, we will take care of the satisfaction of drivers by implementing an expense function that drivers will not need to pay the travel expenses, and if drivers are specially assigned, passengers will need to compensate the driver. Unserved requests will be penalized, and lateness will be removed by the lateness waiting time rollback mechanism, as these are intolerable for the participants. These penalties will also be applied to the driver's own request. This ensured the fairness of between the drivers and the passengers. There will be 15 constraints the model must comply with when generating a schedule. If earliness waiting time is present, it should not be longer than 10 minutes per request. This is implemented as the QOS constraint. The quality of the solution is determined by its inconvenience cost, which **inconvenience cost = total earliness waiting time * weight + total costs + penalty**. Only the weight of the earliness waiting time is adjusted, as the weight for earliness and travel expenses can be simplified to a ratio of **weight : 1**, and the lower earliness waiting time will be given more priority. The number of DARP-like cases is not being optimized individually as it is already included in the travel expenses.

Next, to start the solution searching and optimization process, metaheuristics search algorithms will be built. We choose to use the simulated annealing (SA) algorithm for its faster convergence speed, lower memory usage, and lower processing power needed. We tweaked the SA algorithm to improve its performance. The algorithm used in this project is called SA-based multi-direction iterative local search. The algorithm consists of three parts, the initial solution-generating algorithm, the local search algorithm, and the optimization algorithm. For the local search algorithm, the idea of [11] is referred to, but we improved the search rate of the algorithm by reducing the search step distance to avoid missing out on high-quality solutions.

Moving on to the optimization algorithms, we created a vector of 5 solutions and performed an SA-based local search for all 5 solutions every iteration. The algorithm will have multiple loops and each loop has multiple iterations. At the beginning of each loop, the probability of accepting worse requests will be lowered to encourage global search. This enables the algorithm to explore the solution space more diversely, reducing the possibility of converging with a worse solution. To prevent premature convergence, the algorithm will only converge when the best solution is not updated for a certain number of iterations.

For the handling of the dynamic requests, we will lock the requests that start within 1 hour from the current simulation virtual time. The locked request will not be touched in re-optimization, so their pickup and drop off time will remain the same throughout the simulation. This is to ensure enough reaction time for the participants. Requests that come in 1 hour from the current virtual time will be rejected, and requests that come in within 1 hour before the planning horizon ends will also be rejected. The dynamic request will be inserted into the service list of the car with the lowest inconvenient cost. If the new solution is better than the current solution, no re-optimization will be made. Or else the new solution will go through re-optimization using the same algorithms from the static scenario, with the same constraints and some parameters tweaked to reduce the time needed for re-optimization. The new solution will then become the current solution. Schedule simulation will be needed for dynamic scenarios, and we will run a virtual simulation clock to simulate the schedule. The requests that were served will be removed.

Experiments were done on the algorithm to fine-tune the parameters of the algorithm, and to verify that the algorithm works fine. This also shows that the algorithms can handle the university scenario well. Stress tests have been done to find out how the algorithms act when the number of requests increases, and the maximum number of requests the algorithms can handle while optimizing the solution effectively. For each test objective, the algorithms are tested with different configurations of test cases, with each configuration consisting of 100 test cases. Experiments on the dynamic part are mainly to ensure that the dynamic scenario handling mechanism is working fine. The main objectives will be ensuring the correct rejection and acceptance of requests, the average time needed for re-optimization, and the dynamism of the dynamic scenario, measured in EDOD. The time needed for re-optimization is just for reference as the reaction time in our scenario is long.

The algorithms are proven to be working fine as they can effectively identify the quality of the solutions and rank them according to their quality. The quality of the best solution found is also better than the initial solution. For the stress test, we find out that the algorithms' ability to find the optimal solution will visibly degrade when the number of requests exceeds 400. The trend we observed from the test results is caused by the shift of priority level of the algorithms on either the travel expenses or the earliness waiting time. The weight of the earliness waiting time is 8 as this is the configuration that gives the algorithms the strongest optimization strength out of our tested configurations on the earliness waiting time. Apart from that, based on our observation, the number of iterations of not updated solution quality to converge per loop is set to 35 iterations to prevent unnecessary use of resources. Other than that, from our tests, we saw that the DARP-like cases will be optimized even if it is not being defined as an individual objective function.

Lastly, the experiment results on the dynamic scenario show that the dynamic scenario handling mechanism works fine. The requests are being accepted or rejected correctly. Due to some limitations, we will only use test cases with 100-150 requests for the dynamic scenario. The average re-optimization time is 27.0125 seconds, due to the low number of requests. The average EDOD is 0.1867, indicating a weak dynamic scenario as the reaction time for the handling mechanism is long enough. The average number of dynamic requests accepted in each test is 50.25 requests.

6.2 Recommendation

Recommendations for future improvements can be made mainly on the algorithms and the dynamic scenario handling mechanism.

To speed up the solution search and optimization process, parallel programming can be integrated into the algorithms, allowing multiple operations to run simultaneously. For even greater performance gains, the algorithms can be developed using Compute Unified Device Architecture (CUDA), which enables efficient use of the GPU's extensive parallel processing power. By utilizing the GPU's ability to handle thousands of threads concurrently, this approach significantly boosts computational efficiency and reduces the overall time required for optimization tasks, especially for large-scale problems.

Other than that, to handle more complex and dynamic scenarios, the implementation of deep reinforcement learning (DRL) algorithms as the solution searching and optimization algorithm can be studied. The DRL algorithms have a more stochastic nature, and newer ones even can take the context of all requests when searching for solutions as they are utilizing the transformer model [25]. These enable them to have the capability to produce high-quality solutions even in a very complex scenario. In dynamic scenarios, their ability to handle highly dynamic scenarios with many urgent requests should be further studied as this may be useful in developing a fully autonomous vehicle system.

REFERENCES

[1] S. Su, F. Zhou, H. Yu, "An artificial bee colony algorithm with variable neighborhood search and tabu list for long-term carpooling problem with time window," Applied Soft Computing, vol. 85, pp. 105814–105814, 2019.

[2] J. F. Cordeau, "A Branch-and-Cut Algorithm for the Dial-a-Ride Problem," *Operations Research,* vol. vol. 54, no. 3, p. pp. 573–586, 2006.

[3] Douglas O. Santos, Eduardo C. Xavier, "Taxi and Ride Sharing: A Dynamic Dial-a-Ride Problem with Money as an Incentive," *Expert Systems with Applications,* vol. 42, no. 19, pp. 6728-6737, 2015.

[4] S. Yan, C. Y. Chen and Y. F. Lin, "A Model With a Heuristic Algorithm for Solving the Long-Term Many-to-Many Car Pooling Problem," *IEEE Transactions on Intelligent Transportation Systems,* vol. 12, no. 4, pp. pp. 1362-1373, 2011.

[5] H. A. Taha, Operations Research, 10th ed., Fayetteville: Pearson Education Limited, 2017.

[6] Brenner Humberto Ojeda Rios, Eduardo C. Xavier, Flávio K. Miyazawa, Pedro Amorim, Eduardo Curcio, Maria João Santos, "Recent dynamic vehicle routing problems: A survey," *Computers & Industrial Engineering,* vol. 160, p. 107604, October 2021.

[7] A. Larsen, The dynamic vehicle routing problem, Department of MathematicalModelling, Technical University of Denmark, 2000.

[8] Luca Coslovich, Raffaele Pesenti, Walter Ukovich, "A two-phase insertion technique of unexpected customers for a dynamic dial-a-ride problem," *European Journal of Operational Research,* vol. 175, no. 3, pp. 1605-1615, 2006.

[9] "A tabu search heuristic for the static multi-vehicle dial-a-ride problem," *Transportation Research Part B,* vol. 37, pp. 579-594, 2003.

[10] G. R. Mauri, L. Antonio and N. Lorena, "Customers' satisfaction in a dial-a-ride problem," *IEEE Intelligent Transportation Systems Magazine,* vol. 1, no. 3, pp. 6-14, 2009.

[11] S. Ouasaid and M. Saddoune, "A dial-a-ride problem with driver preferences," *2021 7th International Conference on Optimization and Applications (ICOA),* pp. 1-6, 2021.

[12] M. Posada, H. Andersson, and C. H. Häll, "The integrated dial-a-ride problem with timetabled fixed route service," *Public Transport,* vol. 9, no. 1-2, pp. 217-241, 2016.

[13] A. Ham, "Dial-a-Ride Problem With Meeting Point Feature Known-as Express-Pool," *IEEE Access,* vol. 9, pp. 86404-86411, 2021.

[14] S. Su, D. Xiong, H. Yu, and X. Dong, "A multiple leaders particle swarm optimization algorithm with variable neighborhood search for multiobjective fixed crowd carpooling problem," *Swarm and Evolutionary Computation,* vol. 72, p. 101103, 2022.

[15] Jalel Euchi, Habib Chabchoub, "A hybrid tabu search to solve the heterogeneous fixed fleet vehicle routing problem," *Logistic Research,* vol. 2, pp. 3-11, 2010.

[16] S. Ho, C. Nagavarapu, R. Pandi, and J. Dauwels, "An Improved Tabu Search Heuristic for Static Dial-A-Ride Problem," 2018.

[17] Ali Asghar Rahmani Hosseinabadi, Fataneh Alavipour, Shahab S. Band, Valentina Emilia Balas, "A Novel Meta-Heuristic Combinatory Method for Solving Capacitated Vehicle Location-Routing Problem with Hard Time Windows," *Information Technology and Intelligent Transportation Systems,* vol. 1, pp. 707-728, 2016.

[18] Philippe Marin, Jean-Claude Bignon, Hervé Lequay, "A Genetic Algorithm for use in Creative Design Processes," *Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA),* October 2008.

[19] J. Li, K. Tomita and A. Kamimura, "A Novel Genetic Algorithm for a Multi-Vehicle Dial-a-Ride Problem," *2022 International Conference on Advanced Robotics and Mechatronics (ICARM),* pp. 682-689, 2022.

[20] S. Ouasaid and M. Saddoune, "Construction heuristic for the dial-a-Ride problem with driver preferences," *2021 International Conference on Decision Aid Sciences and Application (DASA),* pp. 192-196, 2021.

[21] Seyed Alireza Fayazi, Ardalan Vahidi, "Mixed-Integer Linear Programming for Optimal Scheduling of Autonomous Vehicle Intersection Crossing," *IEEE*

TRANSACTIONS ON INTELLIGENT VEHICLES, vol. 3, no. 3, pp. 287-299, 2018.

[22] Victor Pillac, Michel Gendreau, Christelle Guéret, Andrés Medaglia, "A review of dynamic vehicle routing problems," *European Journal of Operational Research,* vol. 225, no. 1, pp. 1-11, 2013.

[23] Gerardo Berbeglia, Jean-François Cordeau, Gilbert Laporte, "A Hybrid Tabu Search and Constraint Programming Algorithm for the Dynamic Dial-a-Ride Problem," *INFORMS Journal on Computing,* vol. 24, no. 3, pp. 343-355, 2011.

[24] C. Blum, "Ant colony optimization: Introduction and recent trends," *Physics of Life Reviews,* vol. 2, no. 4, pp. 353-373, 2005.

[25] Yang Zou, Hecheng Wu, Yunqiang Yin, Lalitha Dhamotharan, Daqiang Chen, Aviral Kumar Tiwari, "An improved transformer model with multi-head attention and attention to attention for low-carbon multi-depot vehicle routing problem," *Annals of Operations Research,* vol. 339, pp. 517-536, 2024.

APPENDIX

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S2	**Study week no.:** 2-3
Student Name & ID: Teo Chun Kit, 22ACB00091	
Supervisor: Prof Liew Soung Yue	
Project Title: Vehicle Pick-up and Drop-off Schedule Optimization in a University Setting	

1. WORK DONE [Please write the details of the work done in the last fortnight.] Debug
2. WORK TO BE DONE Further debug
3. PROBLEMS ENCOUNTERED -
4. SELF EVALUATION OF THE PROGRESS On track.

[illegible]

Supervisor's signature

[illegible]

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S2	**Study week no.:** 4-5
Student Name & ID: Teo Chun Kit, 22ACB00091	
Supervisor: Prof Liew Soung Yue	
Project Title: Vehicle Pick-up and Drop-off Schedule Optimization in a University Setting	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

debug

2. WORK TO BE DONE

Experiment setup.

3. PROBLEMS ENCOUNTERED

-

4. SELF EVALUATION OF THE PROGRESS

On track.

[illegible] [illegible]

Supervisor's signature Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S2	**Study week no.:** 6-7
Student Name & ID: Teo Chun Kit, 22ACB00091	
Supervisor: Prof Liew Soung Yue	
Project Title: Vehicle Pick-up and Drop-off Schedule Optimization in a University Setting	

1. WORK DONE [Please write the details of the work done in the last fortnight.] Experiment setup.
2. WORK TO BE DONE Dynamic scenario building.
3. PROBLEMS ENCOUNTERED -
4. SELF EVALUATION OF THE PROGRESS On track.

[illegible]

Supervisor's signature

[illegible]

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S2	**Study week no.:** 8-9
Student Name & ID: Teo Chun Kit, 22ACB00091	
Supervisor: Prof Liew Soung Yue	
Project Title: Vehicle Pick-up and Drop-off Schedule Optimization in a University Setting	

1. WORK DONE [Please write the details of the work done in the last fortnight.] Dynamic scenario building.
2. WORK TO BE DONE Experiments.
3. PROBLEMS ENCOUNTERED -
4. SELF EVALUATION OF THE PROGRESS On track.

[illegible]

Supervisor's signature

[illegible]

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S2	**Study week no.:** 10-11
Student Name & ID: Teo Chun Kit, 22ACB00091	
Supervisor: Prof Liew Soung Yue	
Project Title: Vehicle Pick-up and Drop-off Schedule Optimization in a University Setting	

1. WORK DONE
[Please write the details of the work done in the last fortnight.]

Experiments.

2. WORK TO BE DONE

Report writing.

3. PROBLEMS ENCOUNTERED

-

4. SELF EVALUATION OF THE PROGRESS

On track.

[illegible]

Supervisor's signature

[illegible]

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S2	**Study week no.:** 12-13
Student Name & ID: Teo Chun Kit, 22ACB00091	
Supervisor: Prof Liew Soung Yue	
Project Title: Vehicle Pick-up and Drop-off Schedule Optimization in a University Setting	

1. WORK DONE
[Please write the details of the work done in the last fortnight.]

Report writing.

2. WORK TO BE DONE

-

3. PROBLEMS ENCOUNTERED

-

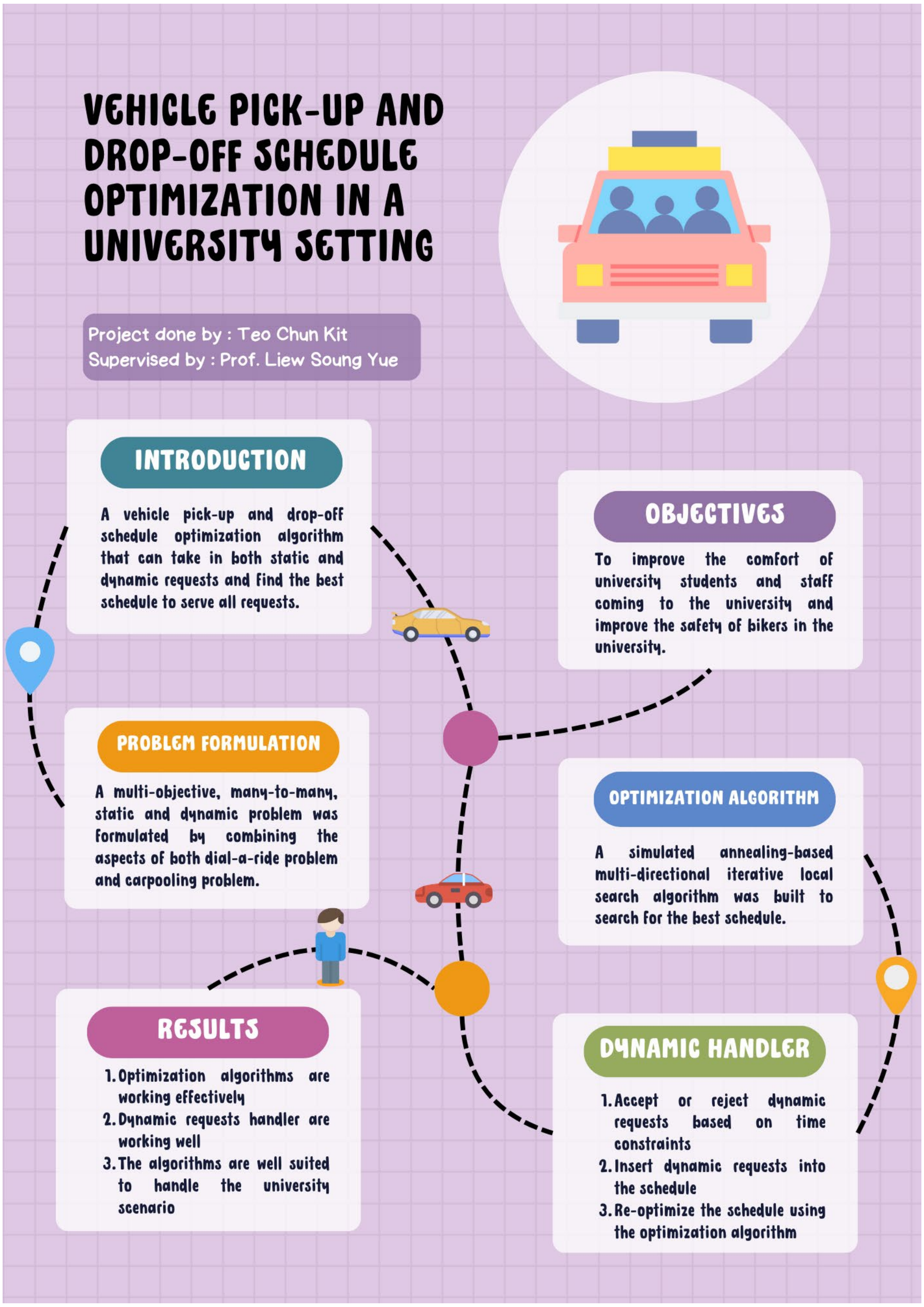
4. SELF EVALUATION OF THE PROGRESS

On track.

[illegible] [illegible]

Supervisor's signature Student's signature

POSTER



VEHICLE PICK-UP AND DROP-OFF SCHEDULE OPTIMIZATION IN A UNIVERSITY SETTING

Project done by : Teo Chun Kit
Supervised by : Prof. Liew Soung Yue

INTRODUCTION

A vehicle pick-up and drop-off schedule optimization algorithm that can take in both static and dynamic requests and find the best schedule to serve all requests.

OBJECTIVES

To improve the comfort of university students and staff coming to the university and improve the safety of bikers in the university.

PROBLEM FORMULATION

A multi-objective, many-to-many, static and dynamic problem was formulated by combining the aspects of both dial-a-ride problem and carpooling problem.

OPTIMIZATION ALGORITHM

A simulated annealing-based multi-directional iterative local search algorithm was built to search for the best schedule.

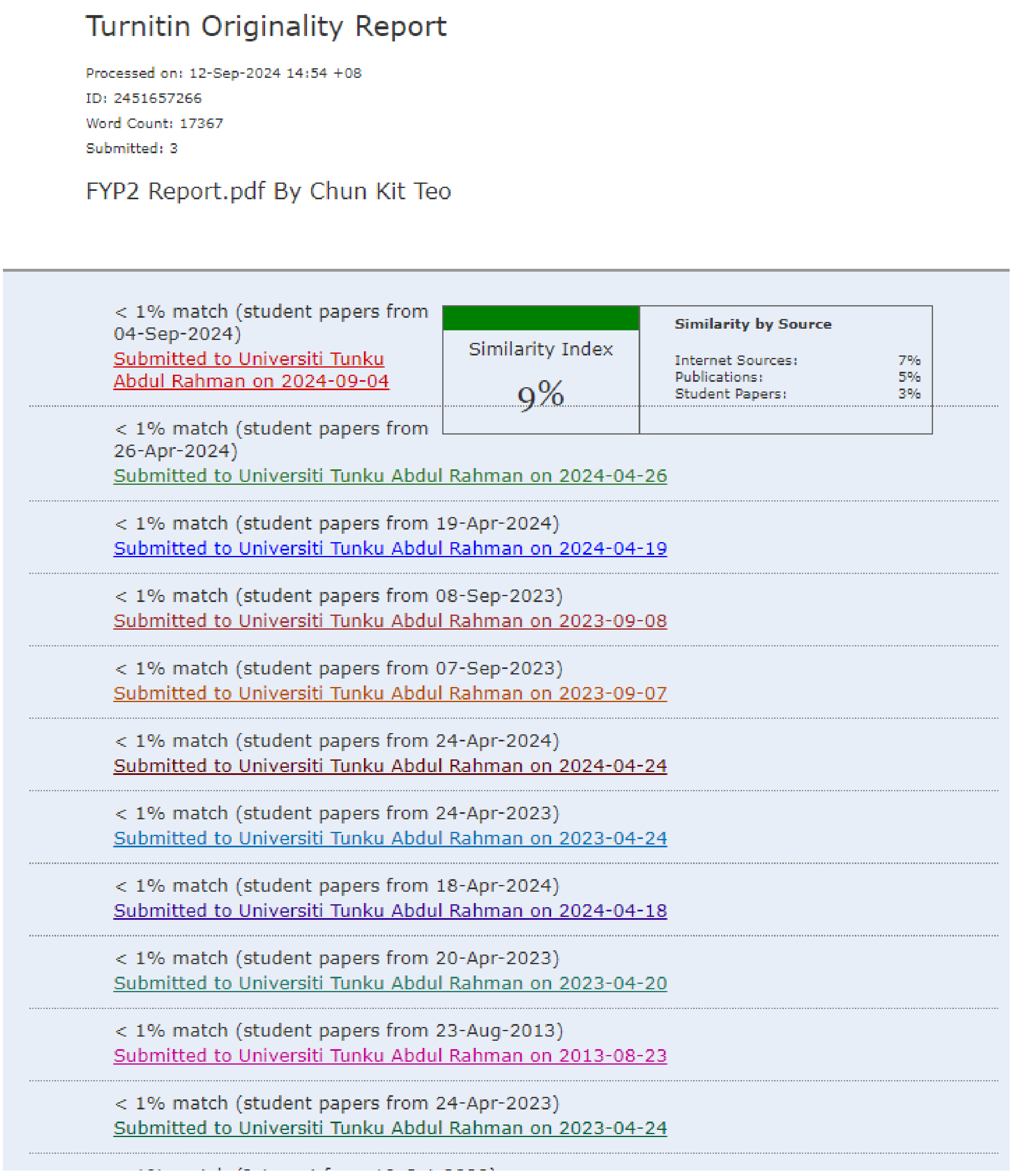
RESULTS

1. Optimization algorithms are working effectively
2. Dynamic requests handler are working well
3. The algorithms are well suited to handle the university scenario

DYNAMIC HANDLER

1. Accept or reject dynamic requests based on time constraints
2. Insert dynamic requests into the schedule
3. Re-optimize the schedule using the optimization algorithm

PLAGIARISM CHECK RESULT



Turnitin Originality Report

Processed on: 12-Sep-2024 14:54 +08
ID: 2451657266
Word Count: 17367
Submitted: 3

FYP2 Report.pdf By Chun Kit Teo

Similarity Index	Similarity by Source	
9%	Internet Sources:	7%
	Publications:	5%
	Student Papers:	3%

< 1% match (student papers from 04-Sep-2024)
Submitted to Universiti Tunku Abdul Rahman on 2024-09-04

< 1% match (student papers from 26-Apr-2024)
Submitted to Universiti Tunku Abdul Rahman on 2024-04-26

< 1% match (student papers from 19-Apr-2024)
Submitted to Universiti Tunku Abdul Rahman on 2024-04-19

< 1% match (student papers from 08-Sep-2023)
Submitted to Universiti Tunku Abdul Rahman on 2023-09-08

< 1% match (student papers from 07-Sep-2023)
Submitted to Universiti Tunku Abdul Rahman on 2023-09-07

< 1% match (student papers from 24-Apr-2024)
Submitted to Universiti Tunku Abdul Rahman on 2024-04-24

< 1% match (student papers from 24-Apr-2023)
Submitted to Universiti Tunku Abdul Rahman on 2023-04-24

< 1% match (student papers from 18-Apr-2024)
Submitted to Universiti Tunku Abdul Rahman on 2024-04-18

< 1% match (student papers from 20-Apr-2023)
Submitted to Universiti Tunku Abdul Rahman on 2023-04-20

< 1% match (student papers from 23-Aug-2013)
Submitted to Universiti Tunku Abdul Rahman on 2013-08-23

< 1% match (student papers from 24-Apr-2023)
Submitted to Universiti Tunku Abdul Rahman on 2023-04-24

Universiti Tunku Abdul Rahman			
Form Title : Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1of 1

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Full Name(s) of Candidate(s)	Teo Chun Kit
ID Number(s)	22ACB00091
Programme / Course	Bachelor of Computer Science (Honours)
Title of Final Year Project	Vehicle Pick-up and Drop-off Schedule Optimization in a University Setting

Similarity	**Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)**
Overall similarity index: 9 % **Similarity by source** Internet Sources: 7 % Publications: 5 % Student Papers: 3 %	Within the required range.
Number of individual sources listed of more than 3% similarity: 0	Within the required range.

Parameters of originality required and limits approved by UTAR are as Follows:
- **(i) Overall similarity index is 20% and below, and**
- **(ii) Matching of individual sources listed must be less than 3% each, and**
- **(iii) Matching texts in continuous block must not exceed 8 words**

Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.

Note Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.

[illegible]

Signature of Supervisor

Name: Liew Soung Yue

Date: 12/9/2024

Signature of Co-Supervisor

Name: ____________

Date: ____________

UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY (KAMPAR CAMPUS)

CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	22ACB00091
Student Name	Teo Chun Kit
Supervisor Name	Prof Liew Soung Yue

TICK (√)	DOCUMENT ITEMS Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
√	Title Page
√	Signed Report Status Declaration Form
√	Signed FYP Thesis Submission Form
√	Signed form of the Declaration of Originality
√	Acknowledgement
√	Abstract
√	Table of Contents
√	List of Figures (if applicable)
√	List of Tables (if applicable)
√	List of Symbols (if applicable)
√	List of Abbreviations (if applicable)
√	Chapters / Content
√	Bibliography (or References)
√	All references in bibliography are cited in the thesis, especially in the chapter of literature review
√	Appendices (if applicable)
√	Weekly Log
√	Poster
√	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)
√	I agree 5 marks will be deducted due to incorrect format, declare wrongly the ticked of these items, and/or any dispute happening for these items in this report.

*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report.

[illegible]

(Signature of Student)
Date: 12/9/2024