ulm university universität
uulm

University of Ulm
Faculty of Engineering and Computer Science
Institute of Media Informatics

Solving the Ridematching Problem in Dynamic Ridesharing

Dissertation zur Erlangung des Doktorgrades Dr. rer. nat.
der Fakulät für Ingenieurwissenschaften und Informatik
der Universität Ulm

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Doctoral Degree Dr. rer. nat. from the Faculty of Engineering and Computer Science at the University of Ulm

Wesam M. A. Herbawi
aus Hebron, Palestine

2012

Amtierender Dekan:	Prof. Dr.-Ing. Klaus Dietmayer
Gutachter:	Prof. Dr.-Ing. Michael Weber
Gutachter:	Prof. Dr. Günther Palm
Gutachter:	Prof. Hussein Abbass

Tag der Promotion: 23.01.2013

Abstract

Among the transportation demand management strategies that can be used to reduce the power consumption, air pollution and traffic congestion is the ridesharing. A flexible form of ridesharing in which a rideshare can be arranged on a very short notice is called dynamic ridesharing. Recently there are many active research areas in dynamic ridesharing including but not limited to security, privacy protection, payment and ridematching. This work is mainly focused on solving the ridematching problem.

The ridematching problem is to match riders and drivers, who wish to take part in ridesharing, in the best possible way taking into account the timing and end points of their trips and other constraints that can be imposed by the ridesharing participants. The computerized ridematching of drivers and riders that requires the minimal effort from the ridesharing participants is the heart of any dynamic ridesharing system. The ridematching problem in realistic dynamic ridesharing scenarios could be an extremely complex optimization problem that becomes intractable for larger problem instances. To attack the ridematching problem, we consider different variants of it and attack them by going from the more restricted to the more general variants. For each ridematching variant, we provide a model for the ridematching problem and propose an algorithm to solve it. All of the proposed algorithms are approximate algorithms, more specifically heuristics, as even the restricted variants of the ridematching problem are hard optimization problems in their own.

The proposed algorithms are tested on artificial and realistic ridematching instances extracted from realistic trip traces. Our experimentation indicates that our proposed algorithms can efficiently solve the ridematching problem by providing good quality solutions in reasonable time. In addition, the results of the algorithms on the realistic ridematching instances indicate that a reasonable number of ridematches can be found in real world scenarios and a considerable amount of travel distances can be saved with ridesharing which in turn could reduce power consumption, air pollution and traffic congestion.

[This page intentionally left blank]

[This page intentionally left blank]

[This page intentionally left blank]

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Transportation sector is a major power consumer and pollution producer. It accounts for the consumption of 51 percent of the liquids made from petroleum and biomass and this share is expected to rise up to 56 percent in 2030 [83]. In addition, transportation sector accounts for 27 percent of the total U.S. greenhouse gas emission in 2010 [29] and there is a growing consensus that increasing anthropogenic greenhouse gases participates in the global warming [62].

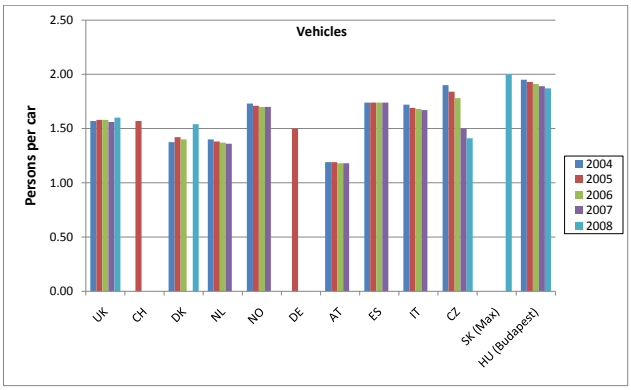


Figure 1.1: The average vehicle occupancies in Europe (From [27]).

Many studies have been directed to evaluate how effectively the transportation sector is operated. According to [27], the average car occupancies in Europe ranges from 1.1 to 2.0 for commuters and leisure trips respectively as in Figure 1.1. The U.S. has no better occupancy rates [8]. This inefficient utilization of the means of transportation leads to increase the air pollution, power consumption and traffic congestion . The estimated cost of lost hours and wasted fuel, resulted from congestion in the U.S. was 78 billion dollars in 2007 [77]. Jacobson and King [50] have showed that adding 1 passenger to every 10 traveling cars would potentially save 7.54-7.74 billion gallons of fuel per year.

Researchers in the field of transportation are interested in new mobility concepts to better utilize the resources in this sector and to increase the comfort level of its users. Among these mobility concepts are the multimodal transportation, door to door transportation services and ridesharing. Ridesharing is the shared use of a vehicle by its driver and one or more passengers called riders. A special type of ridesharing that is enabled by the growing ubiquity of the internet enabled mobile devices is called dynamic ridesharing [37]. Dynamic ridesharing is defined as a ridesharing system with the process of matching the riders and drivers to form ridesharing is done on very short notice or even en-route [1].

1.1 The Ridematching Problem

The definition of dynamic ridesharing indicates that the automated matching of drivers and riders that requires the minimal effort from the participants is the core of any dynamic ridesharing system. The process of matching riders and drivers to form rideshares is called ridematching. Given a set of drivers' offers and a set of riders' requests, the ridematching problem is to match riders' requests and drivers' offers in the best possible way. The best possible way of ridematching depends on an objective function which could include, but not limited to, the maximization of: the number of matches, social connections between matched riders and drivers and travel distance and cost savings. According to Casey et al. [11], dynamic ridesharing is characterized by single non-recurring trips as compared with vanpooling and carpooling which require long term commitment between participants on regular trips (e.g. between home and work). It is characterized also by enabling ridesharing requests and offers to appear close to the desired trip time. Therefore the ridematching in dynamic ridesharing has two major differences from the ridematching in traditional carpooling/vanpooling [20]. The first difference is the handling of the participants' schedules. In carpool/vanpool ridematching, participants' schedules are supposed to be fixed and regular and travel points are also supposed to be fixed. However in dynamic ridesharing the ridematching should consider matching trips with arbitrary end points at arbitrary times. The second difference is that the ridematching in dynamic ridesharing should provide quick answers to participants.

There are different variants of the ridematching problem. The first variant is the single driver and single rider ridematching. In this variant a rider can be matched with at most one driver and a driver can be matched with at most one rider. The second ridematching variant is the multiple drivers and single rider ridematching where a driver can be matched with at most one rider and a rider can be matched with multiple drivers at different times of his trip. The third variant is the single driver and multiple riders ridematching. In this variant, a driver can be matched with multiple riders and a rider can be matched at most with one driver. The last variant of ridematching is the multiple drivers and multiple riders ridematching where a driver can be matched with multiple riders and a rider can be matched with multiple drivers. The different ridematching variants will be discussed in more details in section 2.3.4.

1.2 Contribution

The main goal of this work is to model and solve the ridematching problem in its different variants going from the more specific to the more general variants. Different challenging problems exist in the different ridematching problem variants where all of the variants are hard optimization problems except the single driver and single rider ridematching that is solved to optimality in its basic form as we discuss in the related work in the next chapter. To the best of our knowledge, there exist no previous work that models and solves any of the other three variants of the ridematching problem and satisfies its basic requirements as discussed in section 2.4. Following is a summary of our contributions grouped by the variant of the ridematching problem.

Multiple Drivers and Single Rider Ridematching

We discuss different modeling approaches for this ridematching variant, and then we use an off-the-shelf exact algorithm to solve it. Because of the bad scalability of the used exact algorithm, we propose approximate methods to solve this ridematching variant. Different improvements for the proposed methods are also discussed and tested.

Single Driver and Multiple Riders Ridematching

We provide a model for this variant of the ridematching problem. We take into consideration the basic requirements of this ridematching variant in addition to a new requirement (time windows requirement) as discussed in section 2.3.4. After modeling this variant of the ridematching problem, we propose a metaheuristic to solve it. Afterward, we propose an algorithm that alternates between the proposed metaheuristic and a heuristic algorithm to solve this variant of the ridematching problem in more realistic dynamic ridesharing scenarios. Solving this variant of the ridematching problem with time windows includes solving the single driver and single rider ridematching problem considering the time windows.

Multiple Drivers and Multiple Riders Ridematching

We extend the model and the metaheuristic proposed for the single driver and multiple riders ridematching with time windows to be used for the multiple drivers and multiple riders ridematching with time windows. We experimentally show that the latter formulation of the ridematching problem could result in larger number of ridematches between drivers and riders.

1.3 Organization

This work consists of eight chapter as shown in Figure 1.2. We start by providing some background information in Chapter 2. First we describe the model of the dynamic ridesharing system considered in this work. Then we provide a quick overview of some

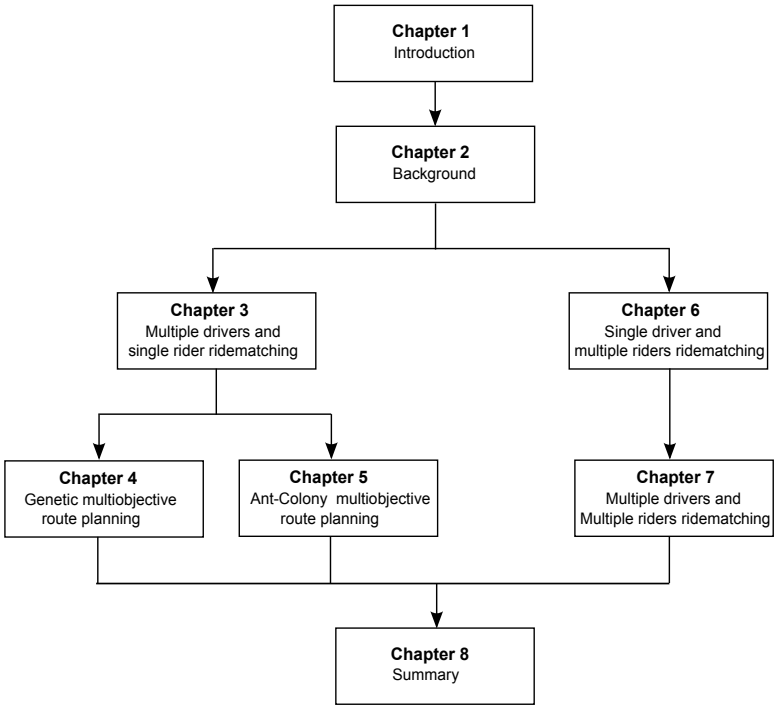


Figure 1.2: Organization of the work and dependencies between the chapters.

active research areas in dynamic ridesharing with more emphasis on the ridematching problem followed by a review of the related work on each variant of the ridematching problem. Afterwards, we discuss why the requirements of each variant of the ridematching problem have not been satisfied in the literature.

In Chapter 3, we consider the multiple drivers and single rider ridematching problem. The major task in this ridematching variant is considered to perform multiobjective route planning for a given riders' request. Different modeling approaches for the multiobjective route planning problem have been explored and an off-the-shelf exact algorithm is used to solve it on the selected model.

Chapter 4 introduces a genetic algorithm metaheuristic for solving the multiobjective route planning problem introduced in Chapter 3 as the used exact algorithm in Chapter 3 suffers from bad scalability. We start by giving a quick overview of the genetic algorithms metaheuristic followed by an introduction for using metaheuristics in solving multiobjective optimization problems. Since metaheuristics are approximate methods, we introduce some widely used quality indicators to judge the quality of the results of the multiobjective metaheuristics. Afterwards, we describe our first version of the genetic algorithm and study its performance. After that we try to improve the performance of the genetic algorithm. Chapter 5 talks about an ant-colony optimization (ACO) metaheuristic for solving the multiobjective route planning problem and compares its performance with the performance of the genetic algorithm.

Chapter 6 considers the single driver and multiple riders ridematching. First a mathematical model for this ridematching variant is provided, and then a genetic algorithm metaheuristic is introduced to solve it. Afterwards, an algorithm that alternates between the genetic algorithm and a heuristic algorithm is introduced to solve the problem in more realistic scenarios. In Chapter 7 we extend the work done in Chapter 6 to solve the multiple drivers and multiple riders ridematching. We try to show that the multiple drivers and multiple riders ridematching can provide more ridematches and mitigate the effects of some of the constraints imposed by the ridesharing participants.

Chapter 8 concludes our work and spots the light on the directions for future work.

Chapter 2

Background

In this chapter, we give a quick overview of the dynamic ridesharing system model considered in this work and the active research areas in dynamic ridesharing. We discuss the ridematching problem and discuss its different variants. For each variant of the ridematching problem we discuss the state of the art work on that variant. Finally we highlight the improvement opportunities in each ridematching variant where our work is going to contribute.

2.1 Dynamic Ridesharing System Model

The dynamic ridesharing system considered in this work is a centralized system. It consists of a centralized server to store and process drivers' offers and riders' requests. Participants (drivers and riders) use their ubiquitous devices to communicate with the server using the internet. Drivers declare their willingness to participate in ridesharing by offering trips to be stored in the server. Drivers' offers consist of their source and destination locations, departure times (as points in time or time windows) and possibly other information such as maximum detour, free seats, preferences, etc. Similar to drivers' offers, riders issue ridesharing requests by specifying their source and destination locations, departure times and possibly other information.

The server is responsible for matching the drivers and riders and providing route plans for them. The server can access social networks data for preferences matching. After sending the the matching results and the route plans for the participants by the server, the participants can communicate directly to monitor the progress of each other towards the planned meeting points. Figure 2.1 shows a general view of the considered dynamic ridesharing system.

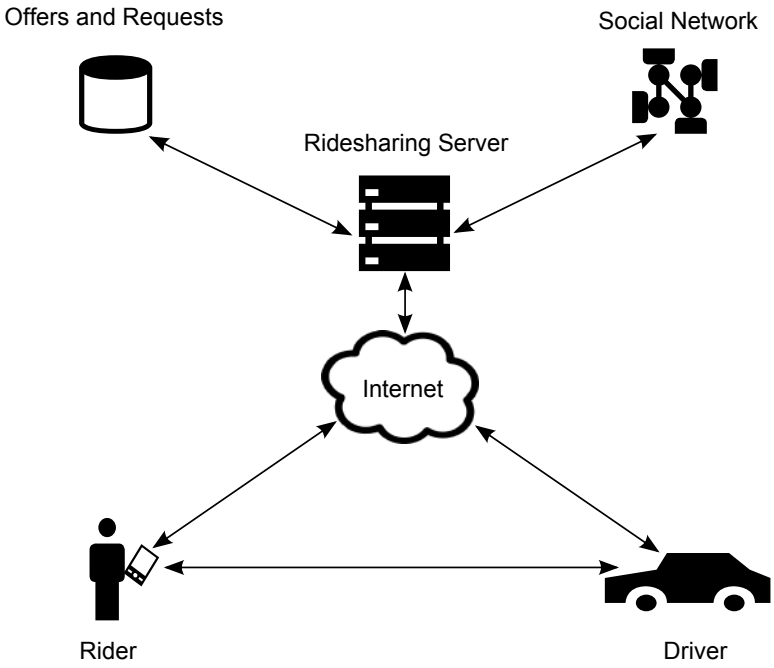


Figure 2.1: Centralized Dynamic Ridesharing System

2.2 Existing Ridesharing Services

There are many existing ridesharing services in the market even few of them are dynamic. Avego.com in Ireland is one of the most famous dynamic ridesharing services where a rider can ask for a trip in real-time. Drivers are supposed to have fixed routes and a rider can use his mobile phone to find a trip with a driver who shares him his route. If a match between a rider and a driver is found, then the system notifies the rider and driver and sends them the arranged meeting point. Another dynamic ridesharing service is flinc.org in Germany which works in a very similar way to avego.com.

Among the most successful non-dynamic ridesharing services is the mitfahrzentrale.de. The rider uses the service website to search for a trip where the service searches the drivers' offers database for the trips that share the rider the same route and return a list of the matching drivers' trips. It is the responsibility of the rider to contact the matching driver per telephone or email to arrange for the trip. Similar services are zimride.com, alternetrides.com, gishigo.com and carpooling.co.uk.

The previously discussed dynamic and non-dynamic ridesharing services provide only simple ridematching. A rider in these services cannot be matched with multiple drivers at different times during his trip to reach his destination hop by hop even though such matching could increase the chance of finding a trip for the rider. In addition, drivers' routes are supposed to be fixed. Enabling the drivers to detour, with maximum detour distances, could also increase the matching chances for the ridesharing participants. Another interesting idea which is not considered in the discussed ridesharing services is

the consideration of time windows for riders' requests and drivers' offers instead of points in time as we discuss in section 2.5. In the following section, we describe the ridematching problem in more details and identify four different variants of ridematching. The adopted ridematching variant in these ridesharing services is the simplest and most restrictive variant of the four variants; namely the single driver and single rider ridematching.

2.3 Active Research Areas in Dynamic Ridesharing

There are many active research areas in dynamic ridesharing. Among these research areas are the trust and security, payment, privacy protection and ridematching. Following we provide an overview of the aforementioned active research areas in dynamic ridesharing. We are mainly interested in the ridematching problem and therefore we discuss it in more details.

2.3.1 Trust and Security

Rideshares are usually coordinated between participants who do not know each other. This raises the problem of security and trust. In order to have comfortable ridesharing, the participants should feel secure with strangers and they should trust each other. Among the approaches to improve trust and security is the utilization of reputation system (like the one used in eBay), limiting the ridesharing to close community and utilizing the information provided by social networks [12, 13, 85].

2.3.2 Payment

The question of how much shall the rider pay for the driver is another research area in ridesharing. On one hand, the payment shall be large enough to encourage drivers to participate in ridesharing. On the other hand, to be attractive for the rider to participate in ridesharing, the payment should be less than what a rider would pay for a taxi. The situation could be more complicated with the presence of multiple choices of ridesharing for the driver which require some detour. The payment could be considered as a motivation for the driver to accept one rideshare and reject the other in a way similar to auctions. On a path with few number of drivers' offers and many riders requests, the payment should be large while it should be small otherwise [16, 48, 56, 95].

2.3.3 Privacy

Recently, a lot of attention has been directed to privacy protection in general especially with the presence of ubiquitous computing devices. People do not want to be tracked. In ridesharing, participants are supposed to provide their travel sources and destinations and possibly their calendar. In addition, more private data might be required such as the age, gender, living place, etc. This information might be necessary for successful ridesharing. A driver might be interested in a rider of a specific age, gender or any other

attribute and vice versa. If privacy is not protected, this might hinder the participation in ridesharing [28, 33, 74, 91].

2.3.4 Ridematching

Automated matching of drivers and riders that requires the minimal effort from the participants is the core of any dynamic ridesharing system. Agatz et al. [1] described different variants of ridematching in dynamic ridesharing. Given a set of ridesharing participants as riders and drivers, a driver might be willing to share the ride with a single rider or with multiple riders during his trip. Analogously, a rider might be ready to make the rideshare with only one driver or might allow the rideshare to be with multiple drivers such that the rider joins different drivers at different times of the same trip to reach his destination. Hence, four variants of the ridematching can be defined; single driver and single rider, single driver and multiple riders, multiple drivers and single rider and multiple drivers and multiple riders. Figure 2.2 shows the ridematching variants and the basic problems in each variant.

	Single Rider	**Multiple Riders**
Single driver	• Matching	• Matching • Routing Drivers
Multiple drivers	• Matching • Routing riders	• Matching • Routing drivers • Routing riders

Figure 2.2: Ridematching variants and the basic problems in each variant (Basic Requirements).

Single Driver and Single Rider (SDSR) Ridematching

The easiest ridematching variant to solve is the single driver and single rider ridematching. In single driver and single rider ridematching, a rider can be matched at most with one driver and each driver can be matched at most with one rider. The basic problem in this variant is to find the best matching of pairs of drivers and riders considering some objective function. If a driver is matched with some rider, then the routing of the driver is straight forward. The driver departs from his source to the source place of the rider. The driver picks up the rider and takes him to his destination place and then the driver travels to his destination as shown in Figure 2.3.

Figure 2.3: Driver routing in single driver and single rider ridematching. Gray nodes represent the source (+) and destination (-) of the driver.

Single Driver and Multiple Riders (SDMR) Ridematching

In the single driver and multiple riders ridematching, a driver can be matched with multiple riders whereas a rider can be matched at most with one driver. This variant is harder to solve than the previous one. The basic two problems in this variant are the matching of riders and drivers and the routing of drivers. The ridematching should find the best match among riders and drivers considering some objective function such that a driver can be matched with multiple riders and a rider can be matched with one driver only. In addition, the ridematching should route the drivers by providing the order of visiting the riders' sources and destinations. It is not necessary to visit a rider's source and destination consecutively which makes the problem even harder as shown in Figure 2.4.

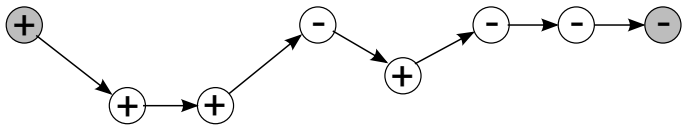


Figure 2.4: Driver routing in single driver and multiple riders ridematching. Gray nodes represent the source (+) and destination (-) of the driver.

Multiple Drivers and Single Rider (MDSR) Ridematching

The multiple drivers and single rider ridematching variant enables the rider to share a ride with multiple drivers at different times whereas a driver can be matched at most with one rider. The basic two problems in this variant are the matching and routing of riders. The matching should consider finding the best matches between drivers and riders considering an objective function such that a rider can be matched with multiple drivers and a driver can be matched with only one rider. The routing of drivers is easy. The same as in single driver and single rider ridematching, the driver travels from his source to the source of the rider to take him to his destination and then the driver travels to his own destination. The more complex task is the routing of the rider. If the rider is matched with multiple drivers, then he should be routed between the drivers such that the spatial and temporal connectivity should be maintained as shown in Figure 2.5. A driver shall deliver the rider at the place where the next driver is supposed to pick him up. In addition the driver shall deliver the rider no later than the time at which the next driver in the route of the rider is supposed to depart.

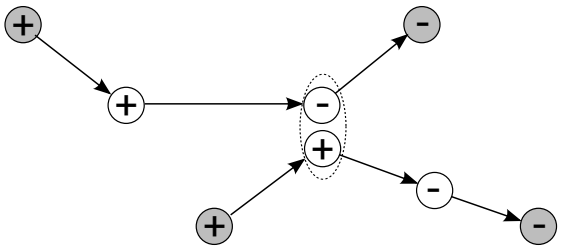


Figure 2.5: Rider routing in multiple drivers and single rider ridematching. The first driver delivers the rider to a local destination, (-) in the dashed ellipse, which is considered the source of the rider , (+) in the dashed ellipse, for the second driver. Gray nodes represent the sources (+) and destinations (-) of the drivers.

Multiple Drivers and Multiple Riders (MDMR) Ridematching

In the variant multiple drivers and multiple riders ridematching, a driver can be matched with multiple riders and a rider can be matched with multiple drivers. This variant combines the complexities of the variant single driver and multiple riders ridematching and the variant multiple drivers and single rider ridematching. The basic problems in this variant are the matching of riders and drivers and the routing of both drivers and riders. A driver has to be routed between the sources and destinations of the riders and a rider has to be routed between the different drivers as shown in Figure 2.6.

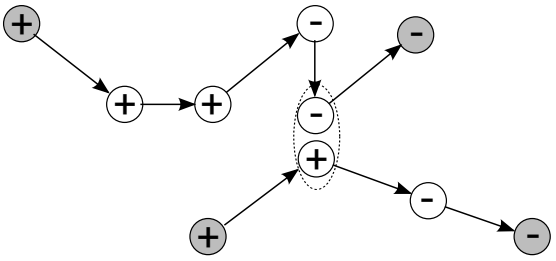


Figure 2.6: Driver and rider routing in multiple drivers and multiple riders ridematching. The first driver delivers a rider to a local destination, (-) in the dashed ellipse, which is considered the source of the rider, (+) in the dashed ellipse, for the second driver. Gray nodes represent the sources (+) and destinations (-) of the drivers.

2.4 Ridematching Related Work

Despite the fact that the ridematching problem is a central concept to dynamic rideshar-ing, not much effort has been invested in solving it [1]. Following we review state of the art work in ridematching organized by the variant of the ridematching.

2.4.1 Single Driver and Single Rider Ridematching

Agatz et al. [2] have compared two approaches for solving the single driver and single rider ridematching. The utility function to be minimized in their work is the total system-wide vehicle miles. The first approach uses simple greedy heuristic and the second approach uses more sophisticated optimization method. Given two sets of drivers offers and riders requests, following we briefly describe the two approaches.

The greedy heuristic approach works as follows. It assigns for each rider's request the driver's offer (if any) with the maximum possible travel distance saving and considers it as a match. The travel distance saving for a given match between a rider's request and a driver's offer is defined as the difference between the travel distance of the driver including serving the rider and the travel distance of the driver and rider if each one travels alone without ridesharing. Among all matches, the method chooses the one with the maximum travel distance saving and fix it. The driver's offer and the rider's request of the fixed match are removed and the process repeats for the remaining offers and requests until no match with positive distance saving is found. In the second approach, they have represented the problem as a maximum weight bipartite matching model. They have created a node for each request o and offer r and inserted an arc (o, r) from o to r if it is feasible to make a ridematch between them with positive travel distance saving. The weight of the arc (o, r) is the travel distance saving of the match. Then the bipartite matching problem is transformed into a network flow maximum cost circulation problem and solved by the commercial optimization tool CPLEX. Simulation results based on the 2008 travel demand data from metropolitan Atlanta indicated that using sophisticated optimization methods can considerably improve the performance of ridesharing systems by increasing the ridematching rate and the travel distance saving.

Geisberger et al. [31] proposed a ridematching method that finds for a single rider's request the best matching driver's offer. Given a set $o_i = (s_i, t_i)$, $i \in \{1...k\}$, of drivers' offers and a rider's request $q = (s^{'}, t^{'})$, the problem statement is to find the driver's offer that can be used to serve the request q with the minimal detour such that the original driver's path is affected only with a small percent. The detour distance of offer o_i is given by $distance(s_i, s^{'}) + distance(s^{'}, t^{'}) + distance(t^{'}, t_i)$ - $distance(s_i, t_i)$ where $distance(a, b)$ returns the shortest path distance between the points a and b. To find the offer with the minimal detour distance, they compute the $2k + 1$ shortest path distances $distance(s_i, s^{'})$, $distance(s^{'}, t^{'})$ and $distance(t^{'}, t_i)$. The main contribution of this work is a fast detour computation method. They have adapted a distance table computation algorithm to compute the distances. The algorithm is based on bidirectional search in a graph preprocessed using contraction hierarchies.

A work that combines the concept of payment and ridematching is proposed by Kleiner et al. [56]. They proposed an auction based ridematching algorithm. The main goal of the proposed algorithm is to increase the probability of finding matches between drivers' offers and riders' requests. Riders use bidding during their search for a match on a specific path. Higher bidding values increase the riders' chances to be matched with drivers. Drivers accept requests based on the bidding value and the required travel distance. The higher is the bid, is the more the detour distance that can be travelled

by the drivers to serve a rider. However, the driver's travel distance is not allowed to exceed the sum of the shortest path travel distances of the driver and rider as if they are traveling alone. The proposed algorithm is compared with an exact algorithm that finds the optimal match based on the travel distance savings. To provide an optimal ridematch, the problem is reduced to finding the maximum weighted bipartite matching which is solved to optimality by the Hungarian algorithm [58]. The proposed auction based algorithm showed to provide higher ridematching rate (higher bidding motivates the drivers to offer longer detours) and provides a balance between the travel distance savings and the matching rate.

The previously discussed works on this variant of ridematching successfully solved it in its basic form. Actually, this variant is the easiest to solve among all other variants. This variant becomes more complex when considering the time windows requirement besides the basic requirements as we discuss in section 2.5. We indirectly solve this variant considering the basic and time windows requirements by solving the more general variant SDMR ridematching considering its basic and time windows requirements.

2.4.2 Single Driver and Multiple Riders Ridematching

A ridesharing system that utilizes GPS enabled devices and social networks to make matches between drivers and riders considering single driver and multiple riders ridematching has been proposed by Gidfalvi et al. [34]. The objectives of the ridematching are to minimize the detour distance that should be made by the drivers to accommodate riders' requests and to maximize the social connections among the participants of ridesharing to increase the comfort level in ridesharing systems. They have modeled the ridematching problem as a bipartite matching graph of offers and requests. The set of edges in the graph represent matches between offers and requests. The weight of an edge is the weighted sum of the two objective functions for the match specified by the edge. To solve the matching problem, they have proposed a greedy algorithm that works on the bipartite graph. The proposed algorithm is very similar to the one proposed by Agatz et al [2] for single driver and single rider ridematching with the major difference that an offer can be considered in more than one match as drivers can be matched with multiple riders.

This work introduces a very important idea to be considered during ridematching which is the use of social networks data. However, a basic requirement for single driver and multiple riders ridematching is missing in this work. The authors do not consider the routing of the drivers between the sources and destinations of the multiple riders in this work. It is hard to tell the quality of the matching without considering the routing of the drivers.

Teodorovi and Orco [89] considered the matching and routing problems in the single driver and multiple riders ridematching. They have proposed a bee colony optimization metaheuristic for solving the ridematching. The trips of the drivers and riders are considered recurring trips between home and work with the weekly travel time schedules being known in advance. The problem statement is to make routing and scheduling of vehicles (drivers) and riders in the best possible way for the whole week. The work states

that the possible objective functions are the minimization of the total travel distance for all participants, the minimization of the total delay and/or to make vehicles utilization relatively equal. However, it is not clear which objective functions are actually considered in their work and how they are modeled and computed. The routing of the driver is done by listing the requests to be served by the driver in sequential order. The driver has to visit the source and destination locations of a request consecutively. The algorithm is tested on a ridesharing demand in the city of Trani in Italy.

This work is an abstract consideration of the problem. First, it does not take the vehicle capacity and the travel distance and time into consideration as constraints while matching the riders and drivers. Second, the adopted routing approach for drivers is inefficient. While a routing approach that simply list the requests to be served in sequential order makes the problem easier to solve, it results in inefficient routes. Sometimes it is much more efficient for example to visit the source locations of two or more riders consecutively before visiting any of their destination locations especially if their source locations are close to each other.

A work that indirectly deals with the single driver and multiple riders ridematching but provides a better routing approach than the previous one is proposed by Kamar and Horvitz [54]. They have developed computational methods for guiding collaboration that show how shared plans can be built in ridesharing. In their work, they consider a set of participants where each one owns a vehicle. The problem is to group the participants based on their sources and destinations to make rideshares (matching) and to route the driver of each group. All participants in one group have their sources close to each other and their destinations close to each other (within a given radius). In each group, one participant is assigned a driver role and the rest as riders. The driver is routed between the sources and destinations of the riders in the group. The objective to be minimized is the participants' inconvenience which combines the increase of the trips' durations and the shifting in travel times with the gain of fuel savings and reducing the other costs of driving a vehicle. The problem is considered as a set cover problem and solved using a greedy set-cover algorithm proposed in [59]. They considered subsets with a maximum subset size of k, where k is the capacity of the vehicle and is considered equal for all vehicles.

This work provides a step forward to solve the routing problem of the driver in the single driver and multiple riders ridematching. It provides a better driver routing approach than the one provided by Teodorovi and Orco (previously discussed) in the sense that the work of Teodorovi and Orco limits the driver to visiting the riders' sources and destinations successively but this work enables interleaving the visiting order of the different riders' sources and destinations. However, all source locations of the riders have to be visited before their destination locations. This is because in this work the sources of participants in each group have to be close to each other (can be justified for carpooling). While this approach might make the routing problem easier to solve, it could result in some limitations for the ridesharing system. One of the limitations of this approach is limiting the ridesharing group size to k (capacity of the vehicle) as the driver visits the source locations of the riders successively. Limiting the ridesharing group size to k might result in some participants being left without ridesharing. The authors assume that the

participants in each group have to share close sources and destinations locations (e.g. home and work) as the application in mind is more like carpooling. This approach does not allow a driver to share a ride with a rider whose end points are on the path of the driver but not close to the endpoints of the driver which is a typical scenario for dynamic ridesharing.

Milica et al. [79] considered a problem similar to the problem addressed previously by Kamar and Horvitz. The problem is to group (match) a set of drivers to form ridesharing groups based on their living and working places and their working hours. The problem is considered as clustering problem. To group the drivers, a modified version of k-means clustering algorithm is used. The authors do not consider further work after the clustering of the drivers. The routing of the driver of each group is not considered in this work.

2.4.3 Multiple Drivers and Single Rider Ridematching

Xing et al. [94] have proposed a method to find a ridematch for a single rider's request given a set of drivers' offers. A request can be matched with two drivers' offers at different times if a single matching offer, that can take the rider from his source to his destination, is not found. The method first determines the set of feasible offers that share the request the same source and destination and depart after the departure time of the request. Among the set of feasible offers, the one with the minimal travel time is selected to serve the request. If a single offer cannot be utilized to serve the request, the method tries to find the set of offers that can cooperate to serve the request with a maximum of two offers' cooperation. The set of cooperating offers contains the set A of offers that share the request its source and the set B of offers whose paths intersect the path of any offer in A and shares the same destination with the request. The method selects the two offers with the minimum accumulated travel time. The authors state that they have limited the number of offers that can cooperate for serving a request because of complexity issues despite the fact that it could be very helpful.

The first problem in this method is the selection criterion of offers to match a request. The method selects the offer with the minimal travel time. This is over simplification as it does not take the waiting times into consideration. A rider can take an offer with a bit longer travel time but arrives earlier to his destination as he needs to wait shorter time for the driver of the offer with the longer travel time. In addition, the adopted selection criteria in the case of two cooperating offers can result in long waiting times for the rider and results in missing attractive combinations of offers. Second, another important criterion that every rider cares about is not considered. The criterion is the cost of the trip. Third, the considered strategy in this method is to match the request with a single offer first and if no single offer is found then consider two cooperating offers. The match with two cooperating offers is given a lower priority and is done only if the matching with single offer does not succeed. While matching the request with more than one offer might introduce inconvenience to the rider, it could reduce the travel time and cost and provides more alternatives to the rider. Instead of long trip with one driver, it would be better to have shorter trip even if it requires traveling with two drivers.

Finally, the number of cooperating offers to serve a request is limited to two at most. This makes the problem easier to solve but reduces the matching possibilities.

2.4.4 Multiple Drivers and Multiple Riders Ridematching

A system that is supposed to match drivers and riders on the basis of multiple drivers and multiple riders ridematching is described by Grueble [36]. The author uses the term multihop (networking concept) ridesharing to refer to a system where the rider can reach his destination hop by hop by sharing rides with multiple drivers. For the rider, each offer in his trip is a hop. This work talks about many advantages of multihop ridesharing (many drivers and many riders ridematching) and highly motivates its implementation. Many options have been discussed for such system to improve its service quality such as the use of historical and/or current system state and external data to optimize one or more route metrics including the cost, number of hops, individual hop and total hop waiting times, overall travel time, overall travel distance and personal preferences. Besides the ridematching problem, the work talks about other interesting problems in ridesharing; namely, security and payment. Although the work motivates the use of the variant multiple drivers and multiple riders ridematching, no algorithmic solution or modeling for the problem has been provided. To the best of our knowledge, there is no attempt to model and solve this variant of ridematching.

2.5 Overall Discussion

We have discussed the related work regarding each variant of the ridematching problem. Following we discuss for each ridematching variant the extent to which the basic requirements have been satisfied in the literature. Then we discuss an interesting requirement which might make all ridematching variants harder to solve but also makes the idea of ridesharing more practical. The requirement is the consideration of time windows that define upper and lower bounds for the departure and arrival times as we explain soon.

Figure 2.7 shows the requirements satisfaction of the different variants of the ridematching problem. The variant single driver and single rider ridematching is reasonably considered in the literature. The basic requirement of this variant (matching pairs of drivers and riders) has been properly solved. This variant is easy to solve and an optimal solution can be reached in polynomial time. This variant becomes harder to solve with the introduction of time windows as we will see.

The basic problem of routing the drivers in the single driver and multiple riders ridematching is not properly considered in the related work. Most of the work on this problem concentrates on the matching of the riders and drivers and do not consider the routing of the drivers between the riders. The two works that considered routing the drivers considered simplified versions of the routing problem. The work of Teodorovi and Orco [89] considers a routing approach where the sources and destinations of riders have to be visited consequently and the work of Kamar and Horvitz [54] considers that the sources of all riders have to be visited before their destinations. As already discussed, the simplified versions of the routing could be easier to solve but do not properly solve

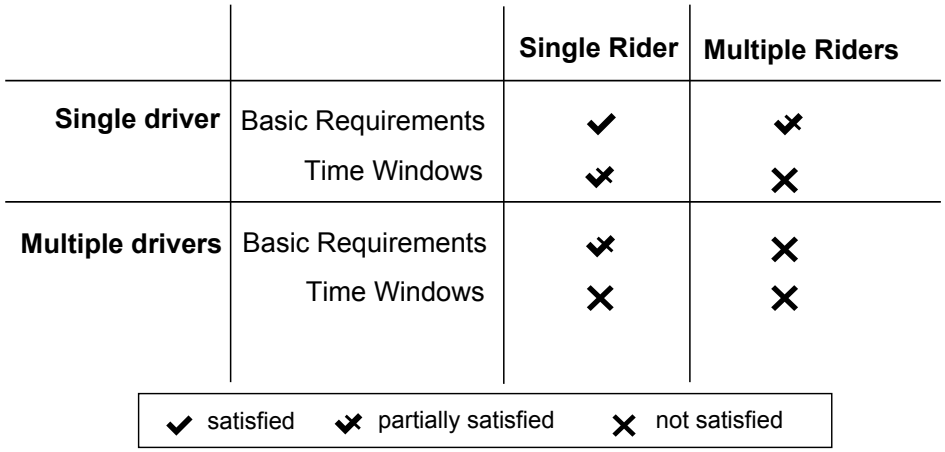


Figure 2.7: Requirements satisfaction of the ridematching variants

the basic drivers routing problem in this variant. Actually the concepts of matching and routing are hard to separate. The routing plays a major role in defining the quality of the match.

Regarding the multiple drivers and single rider ridematching, little work has been proposed to solve it. To the best of our knowledge, there exists no work that considered the problem in its general form to match a rider with multiple drivers and to route him between the drivers. The existing work handles only a limited variant of the problem. A rider is matched with at most two drivers if he cannot be matched with a single driver to take him from his source to his destination. The work on this problem is over simplified and does not consider realistic scenarios. The waiting times are not taken into consideration during the routing of the rider which might result in non preferable routes.

To the best of our knowledge, no previous work has been proposed to solve the multiple riders and multiple drivers ridematching. This variant is the hardest to solve. This variant has been considered as an attractive approach to increase the number of matches and to increase the chance of a rider to find a ridesharing trip. It also increases the alternative matches for both drivers and riders. However, a work that goes further than appreciating this variant to solving it does not exist.

An interesting requirement to be considered during ridematching is the satisfaction of participants' time windows. Ridesharing participants might specify time windows at which they are ready to share their rides. A time window specifies the earliest and latest possible departure time for the participant and hence the earliest and latest arrival time. The concept of time windows in ridesharing is very interesting. For example, an employee participant might specify that at earliest he can depart at 07:30 and at latest at 8:00 to be able to reach his work place on time. A rideshare earlier than the earliest departure time is uncomfortable and a rideshare after the latest departure time is not acceptable. Participants might also specify how much extra time/distance they are willing to travel to participate in ridesharing.

Including the time windows in the ridematching problem makes it harder to solve especially if the total participants travel time is considered in the objective function and if there is a constraint on the participants maximum travel time. Delaying/advancing the departure of a participant might increase the chance of increasing the number of ridematches or decreasing the total travel time/distance. The idea of time windows has not been considered in the related work. The only exception in the related work is the work done by Agatz et al [2] on the single driver and single rider ridematching. They have considered a very similar concept to time windows where the participants specify their earliest possible departure times and their latest arrival times. These two parameters form a time window at the source and destination locations of each participant by considering the time of the shortest path between the source and destination locations.

2.6 Summary

The ridematching problem is a core problem in dynamic ridesharing systems. We talked about four different variants of the ridematching problem and outlined the basic requirements to be addressed in each variant. Literature review showed that the basic requirements of only the first variant, single driver and single rider ridematching, have been satisfied. The basic requirements of the other three variants are either partially satisfied or not considered at all. Furthermore, we have talked about the time windows satisfaction during the ridematching process which is partially considered in the first variant and has not been considered at all in the other three variants.

To help pushing the wheel forward in this research area, we try to fill the blocks of the variants of ridematching. First we start by solving the variant multiple drivers and single rider ridematching considering the basic requirement of proper routing of a rider between drivers. Then we solve the variant single driver and multiple riders ridematching considering the basic requirements and the time windows. Providing a solution to the single driver and multiple riders ridematching with time windows gives a solution to the variant single driver and single rider ridematching with time windows. Finally, we solve the variant multiple drivers and multiple riders ridematching considering the basic requirements and time windows. However, for this variant, we limit ourselves for a maximum of two drivers' cooperation (matching a rider with two drivers at most).

Chapter 3

Multiple Drivers and Single Rider Ridematching

In this chapter we consider the multiple drivers and single rider ridematching problem. The problem is considered in this chapter in its basic form without time windows. The main task in solving this problem is to find a route plan for a rider given a set of drivers' offers. The route plan includes the matching and routing of a rider. The quality of the route plan in this work is defined through the following objectives to be minimized:

- Cost of the route: The total money cost of the route in the plan.
- Time of the route: The total route time in the plan including the waiting time.
- Number of drivers: The number of drivers considered in the route plan.

These objectives are considered in a conceptual work for a ridematching system proposed by Grueble [36]. We consider the drivers' routes to be fixed (no detour and no waiting for riders) for this variant of ridematching. Therefore, we do not consider minimizing drivers' travel time and distance which will be considered for minimization for the other variants of ridematching in Chapters 6 and 7 as drivers will be allowed to detour and wait for riders.

We start by defining the problem and describing the drivers' offers and the riders' requests. Then we describe the multiobjective route planning problem as the main task to be solved. Then we explore different modeling for the drivers' offers to solve the multiobjective route planning problem in multiple drivers and single rider ridematching. After that we discuss an exact algorithm for solving the multiobjective route planning problem on the selected model and examine its performance on a set of problem instances.

3.1 Problem Definition

We consider a dynamic ridesharing system with a predefined set of stations where riders and drivers can meet to share their rides. Let S be the set of all possible stations in

the system and let the set D be the set of drivers participating in the ridesharing. A driver who is willing to participate in ridesharing declares an offer. Each offer $\mathbf{o} \in O$ is a 6-tuple $(s_s, s_t, t_d, t_a, d, c)$ consisting of a starting station s_s, destination station s_t, departure time t_d, arrival time t_a, driver identification d and cost c such that: $s_s, s_t \in S$, $s_s \neq s_d$, $d \in D$, $t_d, t_a \in \mathbb{N}$ and $t_d < t_a$. The whole driver's trip could be split into many offers between successive stations in the driver's route. A rider issues a request $\mathbf{q} = (s_s, s_t, t_d)$, for sharing a ride, that consists of a source station s_s, destination station s_t and departure time t_d such that: $s_s, s_t \in S$ and $s_s \neq s_t$ and $t_d \in \mathbb{N}$. Elements in requests and offers are referenced by functions by their names (e.g. $s_s(q), s_s(o), ...$).

The multiple drivers and single rider ridematching problem is to match the rider with the set of drivers who can take him from his source to his destination and to find the ordering and timing of the matches with the drivers. The process of ridematching a rider with a set of drivers is a process of finding a route plan for the rider among the different drivers. The route plan shall provide the set of drivers with whom the rider will share the ride and also it shall define the ordering and timing of the rideshares with the drivers. The quality of the route plan depends on the objectives cost, time and the number of drivers in the route plan.

3.2 Multiobjective Route Planning

Given a set of drivers' offers and a rider's request $\mathbf{q}$, the rider's route plan is an n-tuple $\mathbf{p} = (p_1, p_2, ..., p_n)$ of drivers' offers that are spatially and temporarily continuous and sufficient to serve a request $\mathbf{q}$ such that: $s_t(\mathbf{p}_i) = s_s(\mathbf{p}_{i+1})$ (spatial continuous) and $t_a(\mathbf{p}_i) \leq t_d(\mathbf{p}_{i+1})$ (temporal continuous) and $s_s(\mathbf{p}_1) = s_s(\mathbf{q})$ and $s_t(\mathbf{p}_n) = s_t(\mathbf{q})$ and $t_d(\mathbf{q}) \leq t_d(\mathbf{p}_1)$ where $1 \leq i < n$ and $\mathbf{p}_i$ is the driver's offer at index i in route plan $\mathbf{p}$.

The quality of a route plan $\mathbf{p}$ is defined through three objectives to be minimized: The cost (money), time and the number of drivers in the route plan. The calculation of each objective for the route plan $\mathbf{p}$ is shown in Equations 3.1-3.3.

$$cost(\mathbf{p}) = \sum_{i=1}^{n} c(\mathbf{p}_i) \tag{3.1}$$

$$time(\mathbf{p}) = t_a(\mathbf{p}_n) - t_d(\mathbf{q}) \tag{3.2}$$

$$\begin{aligned} drivers(\mathbf{p}) &= 1 + \sum_{i=2}^{n} f(i), \\ f(i) &= \begin{cases} 0 & \text{if } d(\mathbf{p}_{i-1}) = d(\mathbf{p}_i) \\ 1 & \text{Otherwise} \end{cases} \end{aligned} \tag{3.3}$$

One approach to solve the multiobjective route planning is to formulate a single objective function consisting of a weighted linear combination of the different objectives and to

find the route plan that minimizes it. This approach makes the problem easier to solve, however it is not a convenient way for the following reasons. The weights need to be carefully set and the riders need to have high experience in setting the different weights to catch good route plans. In addition, there are some good route plans (non-dominated as we see soon) which could not be reached given many weight combinations [63].

In this work we consider minimizing each objective independently. Let P be the set of all feasible route plans and let $F : P \rightarrow [\mathbb{R}^+]^k$ be the function which assigns for each route plan $p \in P$ a k dimensional objective vector where k is the number of objectives. The goal is to find the route plan optimizing all objectives together. Unfortunately, a single route plan that optimizes all objectives together usually does not exist especially with the presence of conflicting objectives. One route plan could have a good value for one objective and bad value for the other and vice versa for another solution. To tell which route plan is better than the other, we use the concept of Pareto-dominance as an ordering relation between the route plans.

Definition 1 (Pareto-dominance) *Given two route plans p with objective vector $f(p) = (y_1, y_2, ..., y_k)$ and p^* with objective vector $f(p^*) = (y_1^*, y_2^*, ..., y_k^*)$, we say that p dominates p^*, written $p \prec p^*$, if and only if $\forall i \in \{1, 2, ..., k\}, y_i \leq y_i^* \wedge \exists i \in \{1, 2, ..., k\} : y_i < y_i^*$.*

In other words $p \prec p^*$ if p has at least one objective value less than the corresponding value of the same objective in p^* and the other objective values are not more than their corresponding values in p^*. Figure 3.1 shows a pictorial explanation of the idea of Pareto-dominance.

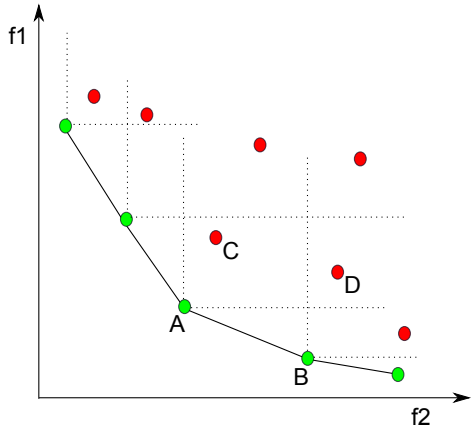


Figure 3.1: Pareto-dominance (minimization). The points in the figure represent the objective vectors of a set of solutions in the objective space. Red solutions are dominated by green solutions. Solution C is dominated by solution A and solution D is dominated by solutions A and B.

Based on the concept of Pareto-dominance, we use the concept of Pareto-optimal [26] [71]. A Pareto-optimal route plan is a route plan for which we cannot improve

one objective function without deteriorating another objective function. The goal of multiobjective route planning is to find the set of all Pareto-optimal route plans which is called the Pareto-optimal set. The rider can select the most preferable solution from the Pareto-optimal set of route plans based on his preferences.

Definition 2 (Pareto-optimal) *For route plan p, if* $\nexists x \in P, x \prec p$ *then the route plan p is called Pareto-optimal.*

Definition 3 (Pareto-optimal Set) *The set of all Pareto-optimal route plans is called Pareto-optimal Set. The Pareto-optimal set of route plans* $= \{p : p \in P \text{ and } \nexists x \in P, x \prec p\}$.

3.3 Modeling Drivers' Offers for Multiobjective Route Planning

We explore different modeling approaches for the drivers' offers to solve the multiobjective routing problem in dynamic ridesharing. Then we discuss the suitability of each model to our problem and we finally select the most suitable model.

3.3.1 Modeling alternatives

Given the multiobjective route planning problem defined, we discuss different ways to model the drivers' offers to solve the multiobjective route planning problem. An example set of drivers' offers will be modeled in different ways to make the different models more clear. The example drivers' offers are: $o_1 = (s_1, s_2, t_1, t_3, d_1, c_1)$, $o_2 = (s_1, s_2, t_4, t_8, d_2, c_2)$, $o_3 = (s_1, s_2, t_7, t_{14}, d_3, c_3)$, $o_4 = (s_2, s_3, t_5, t_7, d_1, c_4)$, $o_5 = (s_2, s_4, t_{10}, t_{14}, d_2, c_5)$, $o_6 = (s_2, s_3, t_{15}, t_{17}, d_3, c_6)$. We start by modeling the example drivers' offers for single objective route planning and then we show the extension for multiple objectives. We consider the objective "time of the route plan" first.

The simplest model is the time-independent condensed model. The drivers' offers are modeled as a digraph $G = (V, A)$ where V is the set of stations S and $A \subseteq V \times V$ is the set of arcs (directed edges) connecting them. An arc $e = (u, v) \in A$ is inserted between two vertices u, v if and only if there is at least an offer $o \in O$ that departs from u to v. The weight of the arc $w(e) = min\{t_a(o) - t_d(o) : o \in O \text{ and } s_s(o) = u \text{ and } s_t(o) = v\}$. In other words, the weight of the arc e is the smallest travel time among all offers that depart from u to v.

This model does not model the problem properly. A query for the shortest path (which represents a route plan as we will see) from a source station to a destination station will always result in the lower bound path in terms of time. This model is not realistic and does not consider the departure time.

In the literature, other models could be used to model the offers properly and mainly they fall in two categories: time-expanded and time-dependent graph models. We start with the time-expanded graph models. There are two types of the time-expanded graph models. The first type is the simple time-expanded graph $G = (V, A)$ [78]. The vertices in the graph refer to the departure and arrival events. For each offer o, two vertices

$u, v \in V$ are introduced. The first vertex represents the departure event from station s_s at time t_d and the second represents the arrival event to station s_t at time t_a. Each vertex is annotated with its station s and event time t (timestamp). An arc $e = (u, v) \in A$, is inserted between each two departure and arrival vertices to represent the travel from s_s to s_t with weight $w(e) = \Delta(t_d(o), t_a(o))$.

At each station s, the set of vertices that represent either departure or arrival events are sorted in ascending order according to their timestamps. An arc $e = (v_i, v_{i+1})$ is inserted between each two consecutive vertices (v_i and v_{i+1} after sorting) that belong to the same station s to model the waiting time at station s with $w(e) = \Delta(t(v_i), t(v_{i+1}))$. Figure 3.2 shows the simple time-expanded graph representation of the example offers. At each station, the gray vertices represent departure events to their corresponding arrival events, white vertices, at the arrival station. The gray vertex at station s_1 with timestamp t_1 represents a departure event from s_1 at time t_1 to station s_2 at time t_5 with the travel time associated with the arc that connects them.

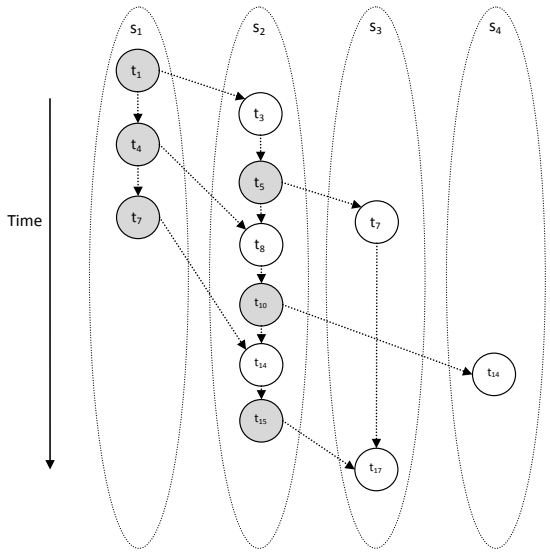


Figure 3.2: Simple time-expanded graph representation of the example offers. Departure vertices v_{dep}: Gray, Arrival vertices v_{arr}: white.

The problem of this model is that it does not model the realistic transfer times. The weight of a wait arc $e = (v_i, v_{i+1})$ might be too small to represent the actual required time to switch between different offers.

To solve the realistic transfer time problem in the simple time-expanded graph, the realistic time-expanded graph [73] is introduced. For each offer $o \in O$, three vertices are introduced: departure, arrival and transfer vertices as shown in Figure 3.3. Each vertex is associated with its station and timestamp as in the simple time-expanded graph where the departure and transfer vertices are assigned the same values. Instead of sorting all vertices that belong to the same station according to their timestamps, only transfer vertices are sorted.

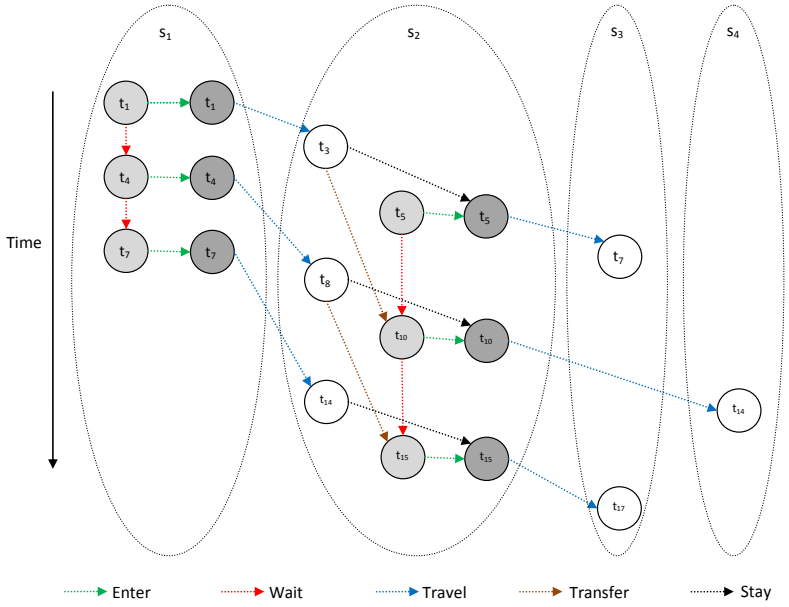


Figure 3.3: Realistic time-expanded graph representation of the example offers. Departure vertices v_{dep}: Gray, Transfer vertices v_{tran}:light gray, Arrival vertices v_{arr}: white.

In this model, five different types of arcs are used as shown in Figure 3.3. The first type, enter arcs, is introduced between the transfer vertices and their corresponding departure vertices with weight 0. The second type, wait arcs, models the waiting at each station and connects the consecutive transfer vertices that belong to the same station with weights $w(e) = \Delta(t(v_i), t(v_{i+1}))$. Another type of arcs, transfer arcs, connects the arrival vertices v_{arr} to the earliest possible transfer vertices v_{tran} that belong to the same station such that $\Delta(t(v_{arr}), t(v_{tran})) \leq Transfer(s)$. The function $Transfer(s)$ returns the minimum transfer time at station s.

To model the continuity with the same driver for two consecutive offers, a new type of arcs, stay arcs, $e = (v_{arr}, v_{dep})$ is introduced that connects the arrival vertices v_{arr} with the departure vertices v_{dep} that belong to the same driver with $w(e) = \Delta(t(v_{arr}), t(v_{dep}))$. Finally, to model the travel from one station to the other, the last type of arcs, travel arcs, is introduced. Travel arcs connect the departure vertices with the arrival vertices that belong to the same offer with $w(e) = \Delta(t(v_{dep}), t(v_{arr}))$.

The second type of graphs is the time-dependent graph models [72]. This type models the drivers' offers the same way as in the condensed model but keeps correctness with the penalty of introducing time dependency in the graph. Like time-expanded models, there are two types of time-dependent models; simple and realistic time-dependent graph models.

In the simple time-dependent model, the graph $G = (V, A)$ consists of a set of vertices V, that represents the set of stations S, and a set of arcs $A \subseteq V \times V$. An arc $e = (u, v)$

is inserted between two vertices u, v if and only if there is an offer o with $s_s(o) = u$ and $s_t(o) = v$ with $w(e)$ being time dependent. Let T be a set denoting time, then the weight of an arc e at time t is $f_t(e)$ where t is the departure time at vertex u and the function $f_t(e) : T \to T$ returns the travel time on arc e at time t. Figure 3.4 shows a simple time-dependent graph modeling the example offers. Like the simple time-expanded graph, the simple time-dependent graph suffers from being unable to model realistic transfer times.

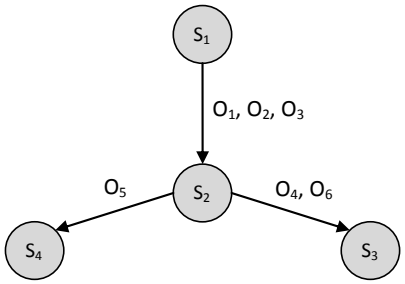


Figure 3.4: Simple time-dependent graph representation of the example offers.

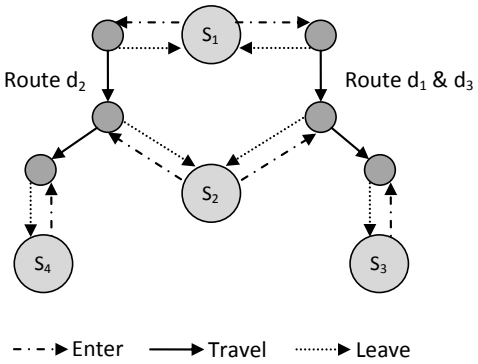


Figure 3.5: Realistic time-dependent graph representation of the example offers. Route vertices: Gray, Station vertices: Light gray.

The realistic time-dependent graph is introduced to avoid the weakness of the simple time-dependent graph in modeling realistic transfer times. In this graph model, stations' vertices are not connected with each other by the presence of an offer that connects them, but instead a new type of vertices called route vertices is introduced. Stations $s_1, s_2, ..., s_n$ form a driver route R if there is a driver visiting them in the same order (offers from the same driver visiting the consecutive stations). All drivers visiting the same set of stations in the same order belong to the same driver route R. Route vertices are inserted to represent the stations visited by each driver route. A route vertex is inserted against each station vertex visited by each driver route. Therefore, for each station, there will be one station vertex and as much route vertices as there are different driver routes that

visit the station.

Route vertices that belong to the same driver route are connected with time-dependent arcs weighted the same way as in the simple time-dependent graph model. Between each station vertex and each route vertex that belongs to the same station, two types of arcs are introduced. A leave arc $e = (r, s)$ from the route vertex r to the station vertex s is inserted with $w(e) = 0$ to model leaving a vehicle where leaving the vehicle accounts for 0 time. The second type is the enter arc $e = (s, r)$ from the station vertex s to the route vertex r to model getting into vehicle with $w(s, r) = c$ (c is a constant value). Figure 3.5 shows the different types of vertices and edges in the realistic time-dependent graph model of the example offers.

3.3.2 Deciding on the Model

Given the five different graph models, following we decide on the suitable model for modeling the drivers' offers to solve the multiobjective route planning problem. While the time-independent condensed model results in a small graph, it fails in modeling the real problem at hand and simply will be ignored. The competition exists between the time-dependent and time-expanded models. To compare the two broad categories; time-expanded and time-dependent, we will use the following criteria: The size of the resulting graph, necessity to follow the FIFO property and the ability to model the multiple objectives.

In general, the size of the time-expanded graph is larger than the time-dependent graph [66]. However, according to [69], solving the earliest arrival problem on a time-dependent graph that does not follow the FIFO property is NP-hard. FIFO property means that there are no two offers o_1 and o_2 that traverse the same edge e at different times t_1 and t_2 respectively, $t_2 > t_1$ and $f_{t_1}(e) + t_1 \geq f_{t_2}(e) + t_2$. In other words, overtaking is prohibited. The time-expanded graph models do not imply this restriction.

For a single objective route planning, the time-dependent graph model seems to be more attractive especially because of the smaller resulting graph. However, when considering realistic scenarios with multiple objectives, the time-expanded graph model becomes more attractive. According to [66], the time-dependent graph model becomes very complex when used to model realistic multiobjective scenarios and its size significantly increases. However, the extension of the time-expanded graph model to model multiple objectives is straight forward as we see soon.

Based on the above discussion, it seems that the time-expanded graph model is more suitable for modeling the drivers' offers for multiobjective route planning. It is not restricted to the FIFO property and it can easily model multiple objectives. Therefore, in this work we utilize the time-expanded graph model to model the multiobjective route planning problem.

Now we compare the simple and realistic time-expanded graph models. For single objective route planning, the main difference between them is the ability to model realistic transfer times. While this point is very important in railway networks where these models are often used, it is less important in our problem because the driver in ridesharing can tolerate zero transfer time for the rider and even the driver can wait for the rider.

Hence for single objective route planning we would utilize the simple time-expanded graph because of its smaller size. However for multiobjective route planning, we have to test the ability of each model to model the different objectives. We consider the time, cost and number of drivers in the route plan to be the set of objectives.

The simple time-expanded graph can easily model the time and cost objectives where the travel arcs are annotated with a weight vector representing the travel time and cost. The wait arcs are annotated with a weight vector representing the waiting time and zero cost. This model fails to model the number of drivers. The realistic time-expanded graph models the three objectives as follows. All arcs are annotated with a weight vector of the form cost, time and number of drivers. The value of the cost is zero for all arc types except the travel arc. The value of the time objective is set to zero for the enter arc and to the time difference between the events connected by the other types of arcs. The value of the number of drivers is zero for all arc types except for the enter arc is set to one.

Because of its ability to model the different objectives and being not restricted to the FIFO property, we decided to utilize the realistic time-expanded graph to model the drivers' offers for multiobjective route planning.

3.4 An Exact Algorithm for Multiobjective Route Planning

Given that the multiobjective route planning problem is modeled as a realistic time-expanded graph, the multiobjective route planning problem is reduced to solving the multiobjective shortest path problem (MSPP) on the realistic time-expanded graph. Finding a path from a source station to a destination station for a specific rider's requests includes matching the rider with a set of drivers and providing the order and timing of the matches with the drivers. Each shortest path represents a Pareto-optimal route plan. To solve the multiobjective shortest path problem, we utilize a generalized label correcting algorithm (GLC) [76] [81].

Algorithm 1 provides the basic steps of the generalized label correcting algorithm. Each vertex in the realistic time-expanded graph is annotated with a set of non-dominated labels (regarding the objective vectors). Each label represents a path P (list of vertices from the source vertex s at the source station to the labeled vertex) and the objective vector $L(P)$ of the path P. Given a request for a rideshare from station s_s at time t_x, we ascendingly search the transfer vertices at station s_s until we reach a vertex s with timestamp greater than or equal t_x. We consider s to be the source vertex for the search (alg. 1 line 4). We create a label for s with an objective vector of zeros and we consider this label to be the first label in the priority queue.

While the priority queue is not empty, we keep removing labels from it and relaxing the edges of the last vertices of the paths in the labels. The set of labels annotating the incident vertices are updated such that at the time of adding a new label to the set of labels annotating some vertex, we add it only if it is not dominated by any label in the set of labels annotating the vertex. If the new label dominates any label in the set of vertex labels, then the dominated label is removed from the set and from the priority

Algorithm 1: Generalized Label Correcting Algorithm

```
1  begin
2      Labels(v) ← Φ, ∀v ∈ V :
3      PriorityQueue : Q
4      s ← getStartVertex(start station, departure time);
5      Q.insert((s), (0, 0, 0))
6      while Q ! = Φ do
7          (P, L(P)) ← Q.dequeue()
8          v ← P_length(P)
9          foreach edge (v,w) ∈ E do
10             P' ← P ‖ (w)              // ‖ is a list concatenation operator
11             L(P') ← L(P) + weight_vector(v, w)
12             updateLabels(w, (P', L(P')))
```

Procedure updateLabels(w,(P',L(P')))

```
1  begin
2      foreach label (P,L(P)) ∈ Labels(w) do
3          if L(P) dominates L(P') then
4              return
5          if L(P') dominates L(P) then
6              delete (P, L(P)) from Labels(w) and Q
7      insert (P', L(P')) into Labels(w) and Q
```

queue as in Procedure updateLabels.

Note that the organization of the labels in the priority queue Q is not straight forward as the labels are partially ordered. In fact, the extraction order of the labels from the priority queue does not affect the correctness of the algorithm, however it affects its efficiency [76]. In this work we utilize the Pareto-dominance to organize the elements in the priority queue such that dominating labels first. We hope that dominating labels are parts of the optimal solutions and hence reduce the number of labels updates in line 12, Algorithm 1.

The multiobjective shortest path problem is a hard optimization problem where any algorithm for solving it has at least exponential runtime complexity in the worst case [88]. The computational complexity of the GLC algorithm depends on the number of labels (representing different paths) annotating the vertices in the graph. Müller-Hanneman and Weihe [67] proved that even for two objectives shortest path problem there could be an exponential number of Pareto-optimal paths at vertex v. Following we briefly show their proof. Given a graph $G = (V, A)$ with two alternative paths between the vertices v_{2i} and v_{2i+2} as in Figure 3.6. A is the set of arcs (v_{2i}, v_{2i+1}), (v_{2i+1}, v_{2i+2})

and (v_{2i}, v_{2i+2}). The arcs (v_{2i}, v_{2i+1}), (v_{2i+1}, v_{2i+2}) are annotated with two dimensional objective vectors of the form $(2^{i+1}, 2^i)$, and the arcs (v_{2i}, v_{2i+2}) are annotated with two dimensional objective vectors of the form $(2^{i+1}, 2^{i+2})$. From G, we can prove by induction that there are 2^i Pareto-optimal paths at vertex $2i$ with objective values of the form $(2^{i+1} - 2 + 2j, 2^{i+2} - 4 - 2j)$ for $j = 0, 1, ..., 2^i - 1$ and $i \geq 1$. We believe that the problem is hard even if there is only one Pareto-optimal path to the destination vertex, still not easy to find it.

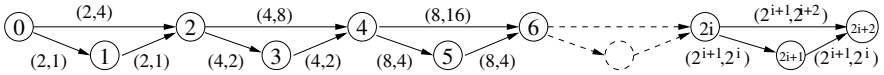


Figure 3.6: Example exponential number of Pareto-optimal paths (From [67])

3.5 Experimentation

To study the behavior of the GLC algorithm, first we have constructed a network of 41 nodes with a density of 0.038 to represent an example stations network with nodes number 0 and 40 being at the opposite ends of the network and all other nodes being in between. The nodes in the network represent the ridesharing stations. Arcs are introduced between nodes such that alternative paths between nodes are possible. Then different number of random trips with length $x, 2 \leq x \leq 40$, have been generated on this network.

A random trip is generated by randomly selecting a trip start station, trip length and trip starting time. Trip length represents how many stations to visit. The trip time and cost between each two stations are set randomly to represent different behaviors for different drivers and each trip is related for a unique driver. The numbers of generated trips, that represent our four instances, were 300, 600, 900 and 1200 trips. We call the four instances DMR300, DMR600, DMR900 and DMR1200.

Experiments are performed on a computer with 4G RAM and 2.4GHz AMD 64bit dual core processor running Windows XP x64. The algorithms are implemented in Java with JRE 1.6.0_22 being the runtime environment.

Figure 3.7 shows the runtime of the GLC on the four instances for finding the set of Pareto-optimal route plans from node (station) number zero to node number forty. We notice the nonlinear increase in the runtime of the GLC as the size of the problem increases. The algorithm failed to give results for a larger instance with 1400 trips (DMR1400).

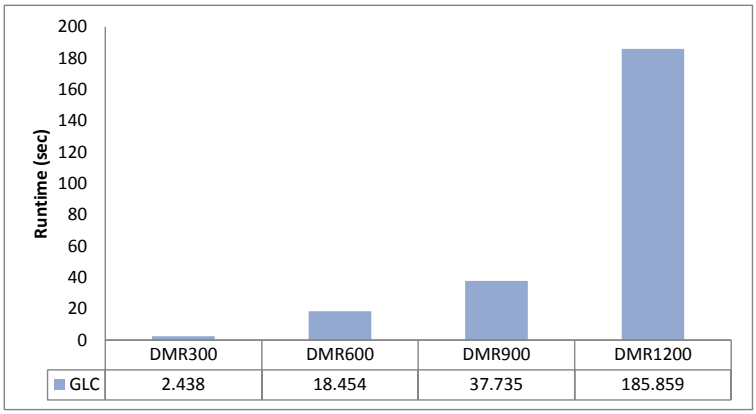


Figure 3.7: Runtime of the GLC algorithm for the different instances

3.6 Summary

In this chapter we have considered the multiple drivers and single rider ridematching problem. The major task of riders route planning is highlighted and considered as a multiobjective optimization problem. Then, different graph models have been investigated to model the drivers' offers for multiobjective route planning. We utilized GLC algorithm as off the shelf exact algorithm to solve the multiobjective route planning problem on the selected graph model. Experimentation results showed a large increase in the runtime of the GLC algorithm when the size of the problem instance increases.

The utilized exact algorithm (GLC) for solving the multiobjective route planning problem suffers from bad scalability. For higher ridesharing demand, the problem becomes intractable using this exact method. In the coming two chapters, we propose approximate methods (more precisely metaheuristics) that are characterized by good scalability for solving the multiobjective route planning problem. The first metaheuristic is a genetic algorithm which is proposed in Chapter 4. In Chapter 5, we propose the second metaheuristic which is an ant colony algorithm and compare its performance with the performance of the proposed genetic algorithm.

Chapter 4

Genetic Multiobjective Route Planning

In this chapter, we propose a genetic algorithm for solving the multiobjective route planning problem introduced in the previous chapter. We start by giving a quick overview of the genetic algorithms metaheuristic followed by an introduction for using metaheuristics in solving multiobjective optimization problems. Then we discuss commonly used quality indicators to assess the quality of the results of multiobjective metaheuristics. The first version of our proposed genetic algorithm for solving the multiobjective route planning is introduced next. After testing our proposed algorithm, we compare the behavior of different evolutionary algorithms (equipped with our solution representation and genetic operators) for solving the multiobjective route planning problem. Then we consider comparing the effect of different solution representations and finally we try to improve our algorithm by equipping it with a local search facility[1].

4.1 Genetic Algorithms Metaheuristic

Genetic algorithms (GAs) are subclass of the wider class of evolutionary algorithms that utilize techniques inspired by natural evolution such as inheritance, selection, crossover and mutation to solve computational problems. In the 1970s, John Holland and his colleagues and students at the University of Michigan have developed the genetic algorithms. The main goal of the invention of the GAs was not to solve specific problems, but to mimic the idea of natural adaptation to be used in computer systems [47]. GAs are considered as metaheuristics. Metaheuristics are randomized algorithms that are used to find optimal (or close to optimal) solutions to hard optimization problems [60]. They are characterized by being general to apply for many classes of problems, computationally efficient and simple to understand and implement [51].

GAs mimic the idea of natural selection by using selection operators and they use genetic-inspired operators to move from a set of solutions (chromosomes) called population to a new population. The genetic-inspired operators include the crossover and

[1]The findings in this chapter are published in [40–43]

mutation operators. The selection operator selects the solutions, from a given population, that are allowed to reproduce. The better (fitter) solutions have more probability to reproduce. The crossover operator combines subparts of two solutions (called parent solutions) to produce new solutions (Child solutions). The hope is to produce child solutions with better quality than the parent solutions. The mutation operator is used to make small changes in the solutions at random locations. The mutation operator helps in exploring the unvisited parts of the search space [64]. Algorithm 2 shows the general skeleton of genetic algorithms. GAs showed success in many areas including but not limited to automotive design, engineering design, robotics, evolvable hardware, routing, gene expression profiling, image processing and scheduling [7, 14, 32, 61, 68].

Algorithm 2: Basic Genetic Algorithm

```
begin
    Initialize population
    Evaluate population
    while Stop criteria not met do
        select solutions from population to reproduce
        create offspring population by applying crossover and mutation operators
        on the selected solutions
        evaluate the offspring population
        replace the population by the offspring population
    return best solution in the population
```

4.2 Multiobjective Metaheuristics

Because of the computational complexity of most multiobjective optimization problems, it seems that one of the most promising approaches for solving them is the use of metaheuristics such as genetic algorithms, local search based algorithms (e.g. Tabu search and simulated annealing) and swarm intelligence (e.g. ant colony optimization and particle swarm optimization) [92].

Many metaheuristics have been proposed for solving multiobjective optimization problems which are mostly generalizations of the single objective metaheuristics. Deb et al. [21], Zitzler et al. [97], and Corne et al. [17] proposed genetic based multiobjective optimizers. Czyzźak and Jaszkiewicz [19] and Bandyopadhyay et al. [4] proposed simulated annealing based multiobjective optimizers and Alaya et al. [3] proposed ant colony based multiobjective optimizer.

The definition of the quality of a solution (fitness in genetic algorithms) plays a major role in guiding the metaheuristic algorithm during the search process. The quality of the solution in single objective optimization problems is defined as the value of the objective function. The result of applying a metaheuristic algorithm on a single objective problem is the solution with the best value of the objective function. However, the definition of

the solution's quality in multiobjective optimization problems is not straight forward. The result of using a metaheuristic algorithm to solve a multiobjective optimization problem is a set of nondominated solutions (Pareto-optimal set) as there is usually no single solution that optimizes all objectives. Therefore, the definition of the solution's quality for multiobjective optimization problems should help in producing high quality and diverse approximate Pareto-optimal set.

4.3 Quality Indicators for Multiobjective Metaheuristics

Following we describe the quality indicators used to assess the quality of an approximate Pareto-optimal set, generated by a metaheuristic, w.r.t the true Pareto-optimal set. The two sets Ω and $\breve{\Omega}$ are the images of the true and approximate Pareto-optimal sets in the objective space respectively. The image of the Pareto-optimal set in the objective space is called Pareto-front.

1. Hypervolume HV (to be maximized) [98, 99]: This quality indicator calculates the hypervolume of the part of the objective space that is dominated by the members of the approximate Pareto-front $\breve{\Omega}$ regarding a reference point as shown in Figure 4.1. The reference point is a vector that consists of the highest values of the objectives throughout all the elements in Ω. This indicator is originally proposed for maximization problems, therefore we invert the values of the elements of $\breve{\Omega}$ and the reference point before applying this indicator.

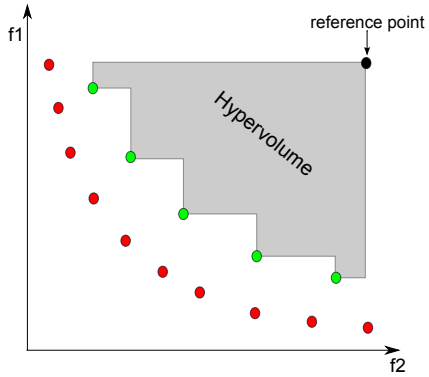


Figure 4.1: Hypervolume. True Pareto-front and approximate Pareto-front are shown in red and green respectively.

2. Generational Distance GD (to be minimized) [93]: This quality indicator is used to measure how far the elements in the approximate Pareto-front $\breve{\Omega}$ are from those in the true Pareto-front Ω as shown in Figure 4.2 and Equation 4.1.

$$GD = \frac{\sqrt{\sum_{i=1}^{n} {d_i}^2}}{n} \tag{4.1}$$

Where n is the number of the elements in $\breve{\Omega}$, d_i is the Euclidean distance between each element of $\breve{\Omega}$ and the nearest element in Ω. The best value of GD is 0, however this does not necessarily mean that the two sets are equal but means that $\breve{\Omega} \subseteq \Omega$. Therefore, we also consider the Inverted GD.

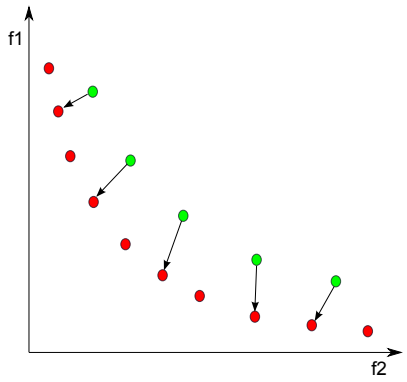


Figure 4.2: Generational Distance (GD). True Pareto-front and approximate Pareto-front are shown in red and green respectively.

3. Inverted Generational Distance IGD (to be minimized): This indicator measures how far the elements in the true Pareto-front Ω are from the elements in the approximate Pareto-front $\breve{\Omega}$ as shown in Figure 4.3.

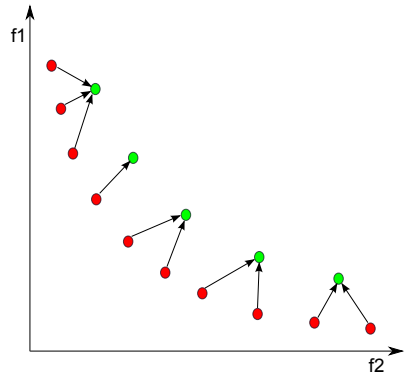


Figure 4.3: Inverted Generational Distance (IGD). True Pareto-front and approximate Pareto-front are shown in red and green respectively.

4.4 Related Work

As we explain in section 4.5, the multiobjective route planning problem is reduced to solving the multiobjective shortest path problem on a simple time-expanded graph representation of the drivers' offers, and then we propose a genetic algorithm metaheuristic

to solve it. In literature, a set of metaheuristics (mostly genetic algorithms) have been proposed to solve the multiobjective shortest path problem where in this section we discuss the state of the art of the work done in this direction.

Mooney and Winstanley [65] have proposed a multiobjective genetic algorithm for solving the MSPP. Their proposed genetic algorithm implements some sort of elitism preservation by keeping the best (nondominated) solutions through generations. However, the algorithm lacks a diversity preservation mechanism which is a necessary feature of multiobjective metaheuristics. A solution for finding a path between a source and destination vertices in a graph is represented as a list of vertices that starts with the source vertex and ends with the destination vertex. The initial population is generated using random walks from the source vertex to the destination vertex. The crossover operator performs a single point crossover between two parent solutions. A vertex is randomly selected from each parent and the two parents are crossed by connecting the first part (the list of vertices from the source vertex up to the selected vertex) of the first solution to the second part (the list of vertices from the selected vertex up to the destination vertex) of the second solution. The result of the crossover is accepted if there is an edge in the graph that connects the two randomly selected vertices from the two parent solutions. The mutation operator simply replaces a vertex in the solution with another vertex, from the graph, that is a adjacent to the predecessor and successor vertices of the replaced vertex in the solution. Their experimentation on different problem instances indicate that the problem can be efficiently solved using genetic algorithms.

Pangilinan and Janssens [70] have utilized an already existing multiobjective evolutionary algorithm and provided problem specific genetic operators. The utilized algorithm is called SPEA2 which will be described in more details in section 4.6. A solution is represented as a list of vertices the same as in [65]. The crossover is a single point crossover where a vertex is randomly selected from one solution and a search is performed for the vertex in the second solution. If the vertex is found in the two solutions then the two solutions are crossed at the positions of the vertex in the two solutions. A solution is mutated by randomly selecting a vertex in the solution and building a new path from the selected vertex to the destination vertex utilizing random walks. Almost the same approach is adopted later by Chitra and Subbaraj [15] (they use the same solution representation and crossover operator), however they utilize an algorithm called NSGA instead of SPEA2. It is not clear in their work how they mutate a solution.

He et al. [38] have proposed an evolutionary algorithm for solving the MSPP. They follow the same solution resperesntation as in [65] but utilizes the depth first search for solutions initialization instead of random walks. The crossover and mutation operators are the same as in [70] with the exception that depth first search is utilized in the mutation instead of random walks. An evolutionary algorithm that has the same solution representation and crossover operator as in [70] is proposed by Kanoh and Hara [55]. The main difference in this work is that they rely on Dijkstra algorithm for population initialization and they do not use mutation.

We notice that the same solution representation (list of vertices) is considered in all of discussed related work on the MSPP, however different methods for solution initialization are utilized in the different works. Single point crossover is always used with different

interpretations. The mutation is used to create random changes in the given paths. We utilize the same solution representation and adopt a single point crossover very similar to the one proposed in [70]. We use a different mutation operator from the previous discussed ones. The mutation operator discussed in [65] is not suitable for our problem as it tries to replace a vertex in a path with another vertex. As we see later on our simple time-expanded graph, it is impossible to replace a vertex with another one. The mutation operator proposed in [70] and used in the other works has the problem that it has a high probability of making large changes in the given solutions. A vertex is randomly selected in a path and a new path is created by creating a path from the selected vertex to the destination vertex. To reduce the probability of large changes in the paths, we do mutation by selecting two vertices from the path instead of one and we try find an alternative path connecting them as we discuss later. Afterwards, we discuss the limitations imposed by the adopted solution representation to solve our problem, and then we propose a new solution representation (we call it time-dependent) and new genetic operators based on the new representation.

4.5 A Genetic Algorithm for Multiobjective Route Planning

Following we describe our first version of the genetic algorithm proposed to find an approximate Pareto-optimal set for the multiobjective route planning problem. Good multiobjective metaheuristics shall provide good quality and diverse approximate Pareto-optimal set. Fitness definition and diversity preservation are active research areas in the design of multiobjective metaheuristics. We do not concentrate on finding new ways for fitness definition and diversity preservation but we concentrate on providing solution representation and genetic operators for the multiobjective route planning problem. Therefore, we use the core of the Nondominated Sorting Genetic Algorithm (NSGA-II) [21] that provides fitness definition in addition to a diversity preservation mechanism and we provide problem specific solution representation and crossover and mutation operators for the multiobjective route planning problem. NSGA-II is considered one of the best multiobjective optimizers. It defines the fitness of a solution based on the concept of non-domination sorting besides a diversity preservation concept as we explain soon. It uses the concept of elitism in order not to lose promising solutions during generations in searching for the best solutions. The basic operations of the algorithm are shown in Algorithm 3. We utilize the jMetal framework [25] implementation of the NSGA-II algorithm.

NSGA-II starts with an initial population P. Its uses the selection, crossover and mutation operators to generate offspring population Q from P (Alg. 3 line 5). The selection operator in NSGA-II is called binary tournament. To select a solution to reproduce, two solutions are randomly selected from the population and compared for dominance. If one solution dominates the other, then it is selected for reproduction. If no one dominates the other, then the one that lies in the less crowded area of the objective space is selected to increase the diversity. To implement the concept of elitism, the populations P and Q are joined in population R to select among them the solutions that

will survive for the next generation (Line 11). To select the solutions that will survive for the next generation, the concept of non-domination sort is used. The solutions in the population R are sorted and ranked according to their level of domination. Solutions that are not dominated by any solution are in rank one and solutions that are dominated by at most one solution are in rank two and so on as shown in Procedure non-dominationSort. The solutions with lower rank have the priority to survive (Line 15). If the number of required solutions to survive from a given rank is less than the number of solutions in that rank, then the crowding distance is used to discriminate between the solutions in the same rank. The crowding distance gives an indication about the density of the solutions in the objective space surrounding a specific solution. Solutions with higher crowding distance (less density) have priority to survive. The crowding distance of a solution is the sum of the distance between the two closest solutions, in the objective space, on either side of the solution along each of the objectives. Procedure assignCrowdingDistance shows the calculation of the crowding distance. Note that, to ensure diversity, the extreme solutions along each objective are assigned crowding distance of ∞.

Algorithm 3: NSGA-II Basic Operations

```
1  begin
2      P ⟵ initial population
3      N ⟵ |P|
4      while !Stop_Condition do
5          for i ← 1 to N/2 step 1 do
6              select parents p1, p2 from P
7              crossover p1 and p1 to get offsprings s1, s2
8              mutate s1 and s2
9              evaluate s1, s2
10             Q ⟵ Q ∪ {s1} ∪ {s2}
11         R ⟵ P ∪ Q
12         F ⟵ non-dominationSort(R)
13         P ⟵ ∅
14         i ⟵ 1
15         while |P| + |Fi| ≤ N do
16             assignCrowdingDistance(Fi, number of objectives)
17             P ⟵ P ∪ Fi
18             i ⟵ i + 1
19         Crowding_Distance_Sorting(Fi)
20         P ⟵ P ∪ Fi[1 : (N − |P|)]
```

Procedure non-dominationSort(set of solutions : P)

```
begin
    l = |P|
    i ⟵ 1
    while P ≠ φ do
        P' ⟵ {first solution in P}
        foreach p ∈ P and p ∉ P' do
            P' ⟵ P' ∪ {p}
            foreach q ∈ P' and q ≠ p do
                if p dominates q then
                    P' ⟵ P' \ {q}
                else if q dominates p then
                    P' ⟵ P' \ {p}
        F_i ⟵ P'
        P ⟵ P \ F_i
        i ⟵ i + 1
```

Procedure assignCrowdingDistance(set of solutions : P, number of objectives: m)

```
begin
    l = |P|
    Individuals[i].crowdingDistance = 0, 1 ≤ i ≤ l
    for i = 1 to m do
        sort_according_objective(P,i)
        P[1].crowdingDistance = ∞
        P[l].crowdingDistance = ∞
        for i = 2 to l-1 do
            Distance = P[i + 1].objective(i) − P[i − 1].objective(i)
            P[i].crowdingDistance = P[i].crowdingDistance + Distance
```

4.5.1 Solution Representation and Population Initialization

We model the drivers' offers as a simple time-expanded graph. A solution is represented as a path on the simple time-expanded graph between a vertex at the source station and a vertex at the destination station. Therefore, the task of the genetic algorithm is to find the set of multiobjective shortest paths between the source and destination stations. The simple-time expanded graph model was not the choice for the GLC algorithm in the previous chapter as it failed to model the number of drivers. Therefore, the GLC is

not guaranteed to give optimal solutions on the time-expanded graph model. However, the genetic algorithm is a randomized algorithm and it is not affected by the inability of the model to model the number of drivers. It computes the values of the objective functions for complete solutions as compared to the GLC which computes the values of the objective functions incrementally for partial solutions and depends on them to disfavor some partial solutions. We utilize an example time-expanded graph with six stations (A-F), Figure 4.4, to explain the solution representation and the genetic operators. As discussed in section 3.3, there are two types of arcs in the time-expanded graph. The travel arcs connect vertices at different stations and the wait arcs connect vertices at the same station. Each arc in the graph is annotated with a 3-dimensional vector representing the time and cost of traversing the arc and an identification of a driver with whom to traverse the arc. The travel time is set to the time difference between the timestamps of the vertices connected by the arc. The cost is set to zero for the wait arcs and to the cost specified by the drivers for the travel arcs. The identification of the driver is set to null for the wait arcs and is set to the identity of the driver for the travel arcs.

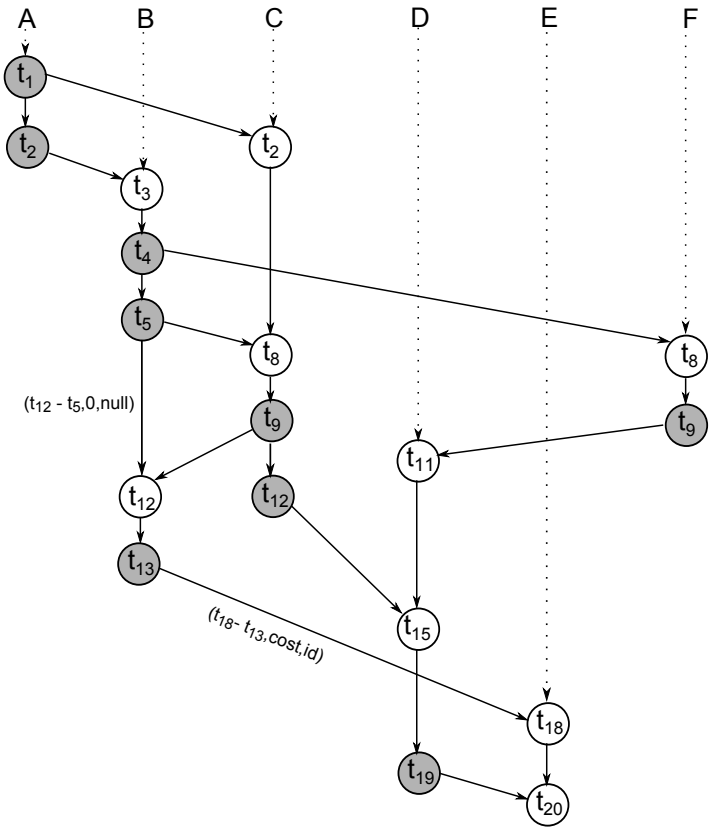


Figure 4.4: An example time-expanded graph with six stations (A-F). Gray vertices represent departure events and white vertices represent arrival events

A solution is represented as a list L of vertices forming a path on the simple time-expanded graph from a source vertex (at the source station) to a destination vertex (at the destination station). Given that different paths have different lengths, we followed the variable-length solution representation approach. For population initialization, we use the Random Walks (RW) [18]. We ascendingly search for a vertex, at the departure station of the rider, that has a timestamp greater than or equal to the departure time of the rider and consider it as the source vertex. Starting from the source vertex and at each vertex, if there are more than one outgoing arcs, we randomly select one to traverse. This process continues until we reach a vertex at the destination station or until we reach a vertex with no outgoing arcs. This method (random walks) is characterized by generating more diverse paths as compared with other approaches such as A* and depth-first and breadth-first search and hence it enables the algorithm to better explore the search space [65]. During population initialization and offspring creation, a station might be visited twice. We suppose that a path with a duplicate visit to the same station will have higher values for the objective functions and we simply consider it as a penalty. Figure 4.5 is an example solution representation for the genetic algorithm to travel from station A to station E.

A_{t2}	B_{t3}	B_{t4}	B_{t5}	C_{t8}	C_{t9}	C_{t12}	D_{t15}	D_{t19}	E_{t20}

Figure 4.5: Time-expanded solution representation

4.5.2 Genetic Operators

Given that the solution is represented as a list L of vertices, following we describe the crossover and mutation operators. The crossover operator between two solutions is a single point crossover. A vertex from one solution is randomly selected and a search for the same vertex in the second solution is made. If the vertex is found in the second solution, then the two solutions are crossed after the positions of the vertex in the two solutions as shown in Figure 4.6. The resulting two solutions of the crossover are always valid paths from a source vertex at the source station to a destination vertex at the destination station. Figure 4.7 shows how the paths on the time-expanded graph look like before and after the crossover. Notice that the crossover operator discovered two new paths from existing paths with simple operation. This efficient exploration of the search space is an advantage of the genetic algorithms. We noticed that the probability of finding the same vertex in the two selected solutions is low. Therefore, if a vertex is not found in both solutions, we select another vertex until finding a vertex that exists in both solutions or canceling the crossover operation otherwise.

The proposed mutation operator makes small changes in a given solutions L by creating an alternative subpath in L as shown in Figure 4.8. Two vertices from L are randomly selected, and then the random walk is utilized to find an alternative path connecting them. We call the two selected vertices from L the local source and local

destination vertices. The alternative subpath is created by randomly walking from the local source until reaching the local destination. During the random walk, if a vertex with a timestamp larger than the timestamp of the local destination is reached, then the mutation fails. Figure 4.9 shows the path on the time-expanded graph before and after mutation.

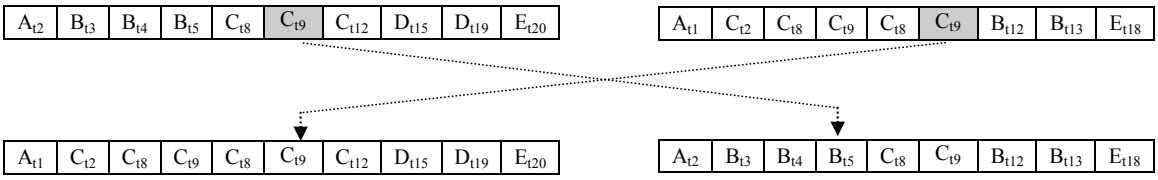


Figure 4.6: Single point crossover: Shaded vertex represent crossover vertex

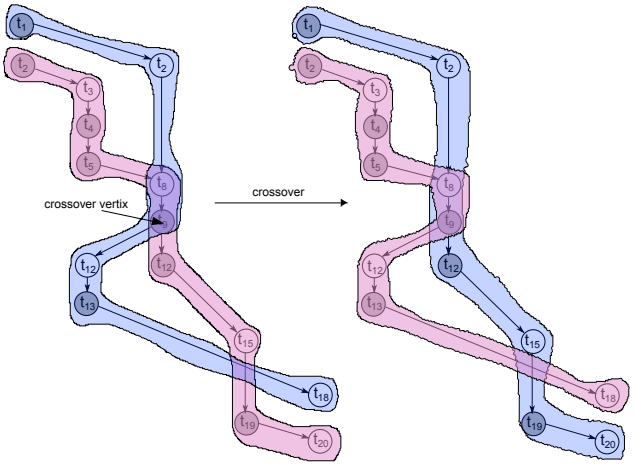


Figure 4.7: Visualization of the crossover operation on the simple time-expanded graph

Note that the application of the crossover operator does not allow the algorithm to discover paths with new stations or vertices that do not exist in the solutions to be crossed. In other words, the crossover operator is closed to the set of stations and vertices in the solutions to be crossed. Therefore the crossover is supposed to intensify the search. The mutation operator diversifies the search by enabling the algorithm to discover new stations and vertices not included in the solutions to be mutated.

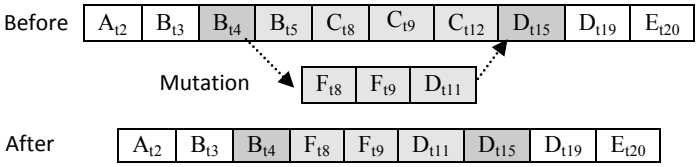


Figure 4.8: Mutation: Dark gray vertices represent the local source and destination vertices and the light gray vertices represent different paths connecting them

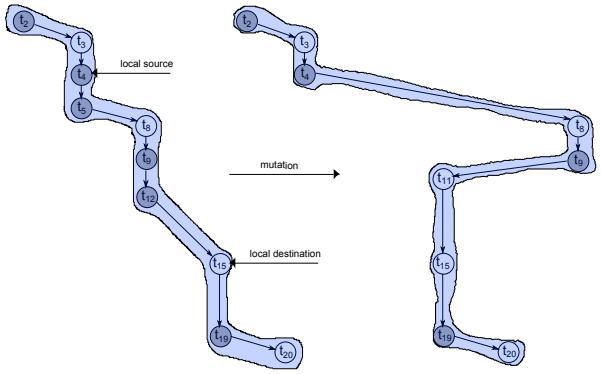


Figure 4.9: Visualization of the mutation operation on the simple time-expanded graph

4.5.3 Experiments

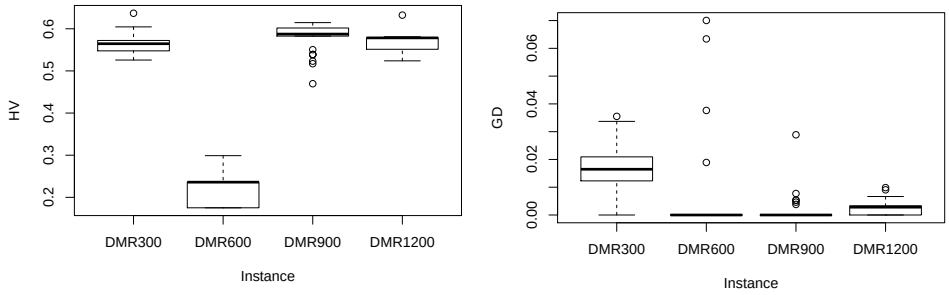
We use the instances DMR300, DMR600, DMR900 and DMR1200 described in Section 3.5 to test the proposed genetic algorithm. Four instances of the simple time-expanded graph are created, one for each instance. A trip of length x in an instance will generate $2 \times x$ vertices in the time-expanded graph as described in section 3.3. For each instance, the genetic algorithm is executed 30 times to find an approximation of the Pareto-optimal solutions set of route plans from station zero to station forty.

The algorithm settings are as follows: The crossover and mutation probabilities are 1.0 and 0.4 per solution (chromosome) respectively. The population size is 100 and the maximum generations is 100 with total of 10000 function evaluations. We noticed that the proposed algorithm is not very sensitive to the values of probabilities of the genetic operators. However, a value less than 0.4 for the mutation operator results in drop in solutions quality. The experimentation platform is the same platform described in Section 3.5.

Table 4.1 shows the mean and standard deviation of the quality indicators for the results obtained by the genetic algorithm on the different problem instances. Also, we provide boxplots for the same data in Figure 4.10. The hypervolumes of the true Pareto-fronts (the Pareto-fronts of the true Pareto-optimal set) of the instances DMR300, DMR600, DMR900 and DMR1200 are 0.747, 0.607, 0.794 and 0.780 respectively. The algorithm achieved more than 70% of the hypervolume of the true Pareto-front on all instances except DMR600 (0.375%). We consider this quality of solutions to be satis-

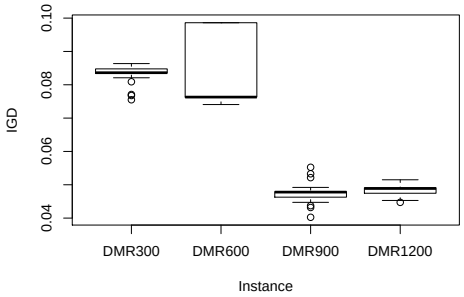
Table 4.1: The values of the quality indicators of the results of the algorithm on the different problem instances. Mean and standard deviation μ_σ.

	DMR 300	DMR 600	DMR 900	DMR 1200
HV	$5.66e-01_{2.48e-02}$	$2.28e-01_{4.34e-02}$	$5.79e-01_{3.35e-02}$	$5.67e-01_{2.47e-02}$
GD	$1.57e-02_{9.80e-03}$	$6.33e-03_{1.81e-02}$	$1.86e-03_{5.52e-03}$	$2.85e-03_{2.52e-03}$
IGD	$8.33e-02_{2.60e-03}$	$8.26e-02_{1.07e-02}$	$4.75e-02_{2.87e-03}$	$4.84e-02_{1.90e-03}$



(a) Hypervolume HV

(b) Generational Distance GD

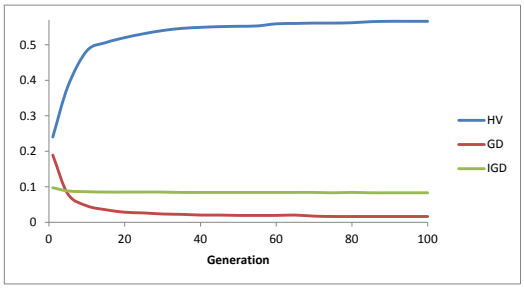


(c) Inverted Generational Distance IGD

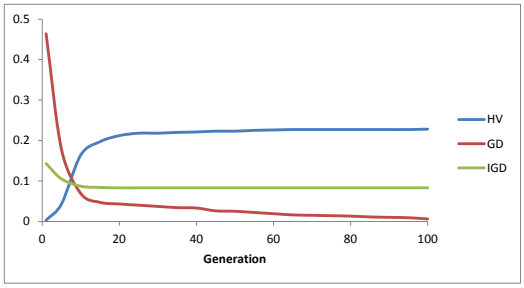
Figure 4.10: Boxplots of the values of the quality indicators of the results of the genetic algorithm on the different problem instances

factory especially if we consider the runtime as we discuss later. The small values of the GD indicator indicate that the solutions produced by the genetic algorithm are very close to the optimal ones.

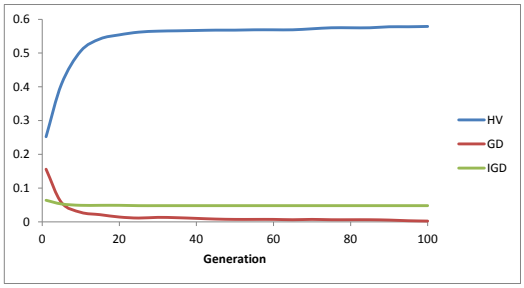
Figure 4.11 shows the values of the different quality indicators at different generations. The Figure indicates that the genetic algorithm is able to maximize the hypervolume and minimize the generational and inverted generational distances. From Table 4.1 and Figure 4.11 we notice that the IGD indicator has higher values than the GD which means that there are parts of the search space not well explored. This results in leaving some points in the true Pareto-front without having points in the approximate Pareto-front being in vicinity which leads to higher values of IGD. After investigating the results, we noticed that the part of the objective space with high values of the second objective



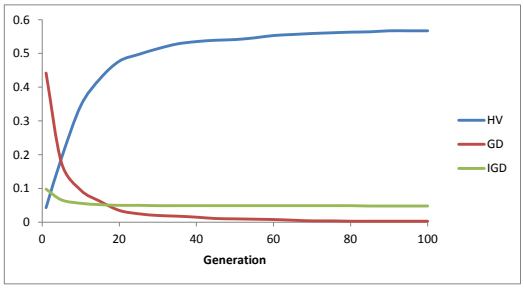
(a) DMR300



(b) DMR600



(c) DMR900



(d) DMR1200

Figure 4.11: The values of the quality indicators at different generations for the different instances

function (time) is not well explored. This is due to the time-expansion nature of the graph and the fact that we use the Random Walks approach for both population initialization and mutation which gives higher chance for solutions with smaller time to be found. The algorithm tends to cover the higher parts of the time-expanded graph which has lower values of time.

Figure 4.12 shows the average runtime of the genetic algorithm for the different instances. It takes the genetic algorithm less than half a second to finish for all instances. This is a great gain compared with the GLC algorithm. A little bit confusing result is that the algorithm took longer time on the instance DMR300 than it took on the larger instance DMR600. We explain this as follows. To provide good approximation, we take care of having diverse initial population. Therefore, at the initialization phase, we try to find undiscovered solutions and neglect already discovered solutions for a constant number of times (Tabu solutions). For smaller instances, the algorithm requires larger number of trials to find distinct solutions. This means that for smaller problem instances, the startup time is significant part of the total runtime of the algorithm.

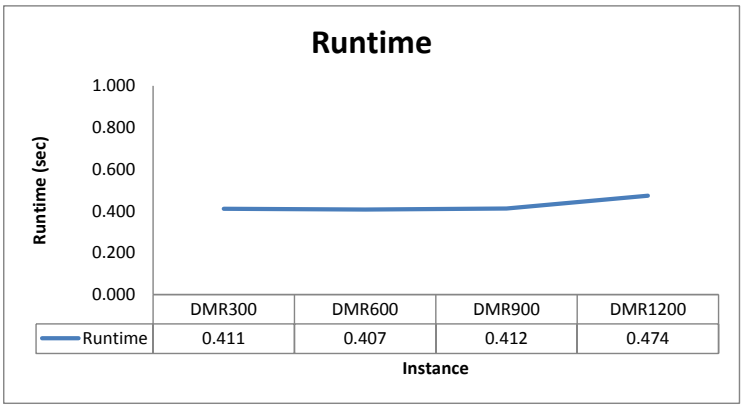


Figure 4.12: Average runtime of the genetic algorithm for the different instances

4.6 Comparing Different Evolutionary Algorithms for Multiobjective Route Planning

We utilized the core of NSGA-II algorithm to solve the multiobjective route planning problem. While NSGA-II is one of the best multiobjective evolutionary algorithms (MOEAs), there are other multiobjective evolutionary algorithms which are considered in the literature that consider different fitness definitions and different diversity preservation mechanisms. In addition, experience says that there is no best metaheuristic for all problem types [10,75]. Different algorithms have different behaviors on different problem types. Therefore we consider utilizing the cores of different state of the art MOEAs and equip them with our solution representation and genetic operators to study their behavior in solving the multiobjective route planning problem.

4.6.1 Algorithms Description

The considered MOEAs are all genetic algorithms except PAES which is an evolution strategy algorithm. The considered algorithms beside NSGA-II are: Improved Strength Pareto Evolutionary Algorithm (SPEA2) [97], Region-based Selection in evolutionary multiobjective optimization (PESA-II) [17], Indicator-Based Selection in Multiobjective Search (IBEA) [96], A Dynamic Population Sizing Approach for Solving Expensive Multiobjective Optimization Problems (FastPGA) [30] and Pareto Archived Evolution Strategy (PAES) [57]. The resulting algorithms from combining the cores of the different MOEAs and our solution representation and genetic operators are simply named after the MOEAs whose cores are being utilized.

Following with the help of Figure 4.13 we describe the general skeleton of the multiobjective genetic algorithms based on which we show the major differences between the different algorithms. Given a population that represents a specific generation (generation x in Figure 4.13), each solution is assigned a fitness value to indicate how good is that solution. Individuals from this population are selected to participate in the creation of the offspring using mating selection. To make sure that the better solutions are not lost through generations, i.e. to maintain elitism, the population and its offspring are mixed and among them an environmental selection process is made to determine the individuals to pass to the next generation (generation x+1 in Figure 4.13). Mostly mating selection is randomized where two parent solutions are randomly selected for the creation of child solutions. Environmental selection is mostly deterministic and depends on the fitness of the solution.

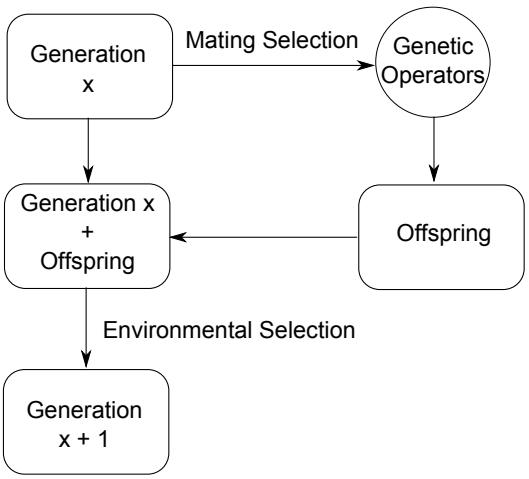


Figure 4.13: General Skeleton of The Genetic Multiobjective Evolutionary Algorithms.

The algorithms NSGA-II, SPEA2, PESA-II, IBEA and FastPGA share this general skeleton with major differences in the definition and calculation of the solution's fitness that plays a major role especially in the environmental selection. Regarding PAES, it is not a genetic algorithm, but an evolution strategy that combines the concept of local

search with Pareto-optimal archived population used to judge the selection process.

NSGA-II uses ranking and crowding distance for fitness definition and calculation as already discussed. It divides the mixture, generation(x) and offspring, into ranks. Within each rank, individuals are differentiated by the crowding distance. It uses the binary tournament method for mating selection as discussed in section 4.5. For environmental selection, NSGA-II selects the solutions with the lower rank as first priority and within each rank, it selects the solutions in the less crowded area in the objective space.

SPEA2 assigns fitness for individuals in the mixture as follows. The fitness of an individual is the sum of the strengths of the individuals dominating it plus its density. The strength of an individual is the number of solutions it dominates and the density is a decreasing function of the k-th nearest neighbor distance. SPEA2 utilizes binary tournament for mating selection that compares the two selected solutions using only the dominance comparison and randomly selects one if no solution dominates the other. Solutions are sorted in the mixture according to their fitness and the environmental selection selects the fittest solutions to proceed to the next generation.

PESA-II is slightly different in that the mating selection is region based instead of individual based and it keeps an external archive to maintain the set of nondominated solutions found so far. The archive is divided into hyperboxes or regions to which the fitness is assigned based on the number of solutions in each hyperbox. For mating selection, it randomly selects two occupied hyperboxes and randomly selects an individual from the less occupied one. The solutions of the offspring are added to the archive such that for environmental selection, simply keep the nondominated solutions in the archive. If the archive is full and a solution is to be inserted, then if it belongs to the most populated hyperbox, it is rejected. Otherwise, a solution from the most populated hyperbox is removed.

IBEA defines the fitness in terms of binary quality indicator. There are more than one binary quality indicators that can be used. In this study, we use a binary quality indicator based on the hypervolume concept. Each solution in the mixture is assigned a fitness value that indicates the loss in the hypervolume covered by the mixture if the solution is removed. For mating selection, two solutions are randomly selected and the one with the better fitness is selected. For environmental selection, it removes the worst solutions (solutions that if removed, then small loss in hypervolume will happen) from the mixture to match the required size.

FastPGA uses an approach similar to NSGA-II for fitness assignment. It divides the mixture into two ranks; the first rank is the set of all nondominated solutions and the second one contains all dominated solutions. For the first rank, crowding distance is used as fitness to distinguish between the nondominated solutions. The fitness of the individual in the second rank is the summation of the strengths (same as SPEA2 strength definition) of the solutions it dominates minus the strengths of the solutions dominating it. For mating selection, binary tournament is utilized where among two randomly selected solutions the one with the better rank is selected and if they have the same rank then the one with the better fitness or randomly otherwise. For environmental selection, the mixture is sorted according to the rank and within the rank according to the fitness and the better solutions are selected for the next generation. Another distinguishing

point for FastPGA is the variable population size such that the population size is not fixed throughout the generations and depends on the number of the solutions in the first rank.

PAES is an evolution strategy that utilizes the idea of the local search with the archiving process. Initially a random solution is generated and inserted into the archive. A copy of the solution, original solution, is then mutated and the resulting new solution is compared with the original one. If it dominates the original solution then it is inserted into the archive and used for the generation of new solution through mutation. If no one dominates the other, then the new solution is inserted into the archive and a comparison between the new solution and the original one is made using the archive to determine the solution that lies in the less crowded area to be selected for generating new solution.

4.6.2 Experimentation

We compare the behavior of the algorithms on the problem instances DMR300, DMR600, DMR900 and DMR1200 described in section 3.5. For each instance, each algorithm is executed 100 times to find an approximation of the Pareto-optimal solutions set of route plans from station zero to station forty. Algorithms settings are summarized in Table 4.2. Population size and number of generations are 100 and 250 respectively for all algorithms. For PAES to be equivalent to other algorithms, 25000 evaluations/generations were made. The experimentation platform is the same platform described in Section 3.5.

Table 4.2: Algorithms Settings

	NSGA-II	SPEA2	PESA-II	IBEA	FastPGA	MOCell	PAES
Archive Size	-	100	100	100	-	100	100
Population Size	100	100	100	100	100	100	1
Max Population Size	-	-	-	-	100	-	-
Number of generations	250	250	250	250	250	250	25000
Crossover Probability	1.0	1.0	1.0	1.0	1.0	1.0	-
Mutation Probability	0.4	0.4	0.4	0.4	0.4	0.4	1.0
a, b, c & d	-	-	-	-	20, 1, 20 & 0	-	-

Table 4.3 shows the mean and standard deviation of the different quality indicators for the different problem instances for each algorithm. Tables 4.4 - 4.15 provide pairwise Wilcoxon rank sum test comparison of the algorithms for each problem instance and quality indicator. The symbols $\triangledown$, ▲ and – mean row item is statistically significant worse than column item, row item is statistically significant better than column item and no significant difference respectively.

There is no significant difference between NSGA-II and SPEA2 on all problem instances regarding all quality indicators. PESA2 is beaten by NSGA-II on the instance DMR900 regarding the HV quality indicator. It is also beaten by NSGA-II regarding the GD quality indicator on the instance DMR600 and regarding the IGD on the instances DMR300 and DMR600. IBEA and FastBGA are beaten by NSGA-II, SPEA2 and PESA-II on most instances regarding most quality indicators. The results of PAES are fluctuating. However the results indicated that it is either beaten by NSGA-II and

Table 4.3: The values of the quality indicators of the results of the different algorithms on the different problem instances. Mean and standard deviation μ_σ.

		NSGAII	SPEA2	PESA-II	IBEA	FastPGA	PAES
DMR300	HV	$5.83e-01_{2.8e-02}$	$5.82e-01_{3.3e-02}$	$5.75e-01_{2.9e-02}$	$5.74e-01_{3.2e-02}$	$5.75e-01_{2.8e-02}$	$5.85e-01_{2.0e-02}$
	GD	$1.39e-02_{6.7e-03}$	$1.30e-02_{6.7e-03}$	$1.32e-02_{7.0e-03}$	$1.26e-02_{6.4e-03}$	$1.62e-02_{6.7e-03}$	$1.27e-02_{7.2e-03}$
	IGD	$8.28e-02_{3.3e-03}$	$8.34e-02_{2.2e-03}$	$8.39e-02_{1.9e-03}$	$8.42e-02_{3.2e-03}$	$8.37e-02_{2.3e-03}$	$8.42e-02_{2.2e-03}$
DMR600	HV	$2.22e-01_{8.5e-02}$	$2.37e-01_{5.5e-02}$	$2.16e-01_{4.3e-02}$	$2.10e-01_{7.8e-02}$	$2.03e-01_{6.9e-02}$	$2.12e-01_{3.6e-02}$
	GD	$1.60e-03_{8.5e-03}$	$1.64e-03_{9.6e-03}$	$0.00e+00_{0.0e+00}$	$1.64e-02_{3.8e-02}$	$1.20e-03_{6.9e-03}$	$1.34e-02_{2.5e-02}$
	IGD	$8.61e-02_{1.6e-02}$	$8.27e-02_{1.2e-02}$	$8.63e-02_{1.1e-02}$	$8.59e-02_{1.3e-02}$	$8.83e-02_{1.5e-02}$	$8.36e-02_{1.1e-02}$
DMR900	HV	$5.81e-01_{3.6e-02}$	$5.81e-01_{3.0e-02}$	$5.71e-01_{3.7e-02}$	$5.59e-01_{3.4e-02}$	$5.64e-01_{4.2e-02}$	$5.80e-01_{2.8e-02}$
	GD	$9.73e-04_{5.4e-03}$	$6.77e-04_{3.2e-03}$	$1.06e-03_{4.5e-03}$	$4.62e-04_{2.6e-03}$	$3.36e-04_{1.5e-03}$	$3.12e-03_{6.3e-03}$
	IGD	$4.72e-02_{2.7e-03}$	$4.77e-02_{1.9e-03}$	$4.82e-02_{2.4e-03}$	$4.86e-02_{2.2e-03}$	$4.88e-02_{3.4e-03}$	$4.76e-02_{2.2e-03}$
DMR1200	HV	$5.78e-01_{1.6e-02}$	$5.75e-01_{2.1e-02}$	$5.76e-01_{1.4e-02}$	$5.74e-01_{1.7e-02}$	$5.71e-01_{1.7e-02}$	$5.61e-01_{2.2e-02}$
	GD	$2.09e-03_{1.8e-03}$	$3.53e-03_{1.5e-02}$	$1.57e-03_{1.6e-03}$	$1.81e-03_{1.8e-03}$	$1.93e-03_{1.6e-03}$	$3.13e-03_{2.3e-03}$
	IGD	$4.78e-02_{1.7e-03}$	$4.79e-02_{1.7e-03}$	$4.80e-02_{1.5e-03}$	$4.84e-02_{1.1e-03}$	$4.85e-02_{1.4e-03}$	$4.81e-02_{1.8e-03}$

Table 4.4: DMR300. HV pairwise Wilcoxon rank sum test.

	SPEA2	PESA-II	IBEA	FastPGA	PAES
NSGAII	–	–	▲	▲	–
SPEA2		–	▲	–	–
PESA2			–	–	▽
IBEA				–	▽
FastPGA					▽

Table 4.5: DMR600. HV pairwise Wilcoxon rank sum test.

	SPEA2	PESA-II	IBEA	FastPGA	PAES
NSGAII	–	–	▲	–	–
SPEA2		▲	▲	▲	▲
PESA2			–	–	–
IBEA				–	–
FastPGA					–

Table 4.6: DMR900. HV pairwise Wilcoxon rank sum test.

	SPEA2	PESA-II	IBEA	FastPGA	PAES
NSGAII	–	▲	▲	▲	–
SPEA2		▲	▲	▲	–
PESA2			▲	–	–
IBEA				▽	▽
FastPGA					▽

Table 4.7: DMR1200. HV pairwise Wilcoxon rank sum test.

	SPEA2	PESA-II	IBEA	FastPGA	PAES
NSGAII	–	–	▲	▲	▲
SPEA2		–	▲	▲	▲
PESA2			▲	▲	▲
IBEA				▲	–
FastPGA					–

Table 4.8: DMR300. GD pairwise Wilcoxon rank sum test.

	SPEA2	PESA-II	IBEA	FastPGA	PAES
NSGAII	–	–	–	▲	–
SPEA2		–	–	▲	–
PESA2			–	▲	–
IBEA				▲	–
FastPGA					▽

Table 4.9: DMR600. GD pairwise Wilcoxon rank sum test.

	SPEA2	PESA-II	IBEA	FastPGA	PAES
NSGAII	–	▲	▲	–	▲
SPEA2		▲	▲	–	▲
PESA2			▲	–	▲
IBEA				▲	–
FastPGA					▲

Table 4.10: DMR900. GD pairwise Wilcoxon rank sum test.

	SPEA2	PESA-II	IBEA	FastPGA	PAES
NSGAII	–	–	–	–	▲
SPEA2		–	–	–	▲
PESA2			▲	–	▲
IBEA				–	▲
FastPGA					▲

Table 4.11: DMR1200. GD pairwise Wilcoxon rank sum test.

	SPEA2	PESA-II	IBEA	FastPGA	PAES
NSGAII	–	–	–	–	▽
SPEA2		–	–	–	▽
PESA2			–	–	▲
IBEA				–	▲
FastPGA					▽

Table 4.12: DMR300. IGD pairwise Wilcoxon rank sum test.

	SPEA2	PESA-II	IBEA	FastPGA	PAES
NSGAII	–	▲	▲	▲	▲
SPEA2		–	▲	–	▲
PESA2			▲	–	▲
IBEA				▽	–
FastPGA					▲

Table 4.13: DMR600. IGD pairwise Wilcoxon rank sum test.

	SPEA2	PESA-II	IBEA	FastPGA	PAES
NSGAII	–	–	–	▲	–
SPEA2		▲	▲	▲	▲
PESA2			–	–	–
IBEA				–	–
FastPGA					–

Table 4.14: DMR900. IGD pairwise Wilcoxon rank sum test.

	SPEA2	PESA-II	IBEA	FastPGA	PAES
NSGAII	–	▲	▲	▲	–
SPEA2		▲	▲	▲	–
PESA2			▲	–	–
IBEA				▽	▽
FastPGA					▽

Table 4.15: DMR1200. IGD pairwise Wilcoxon rank sum test.

	SPEA2	PESA-II	IBEA	FastPGA	PAES
NSGAII	–	–	▲	▲	–
SPEA2		–	▲	▲	–
PESA2			▲	▲	–
IBEA				–	▽
FastPGA					–

SPEA2 or at most as good as them on all instances regarding all quality indicators except for the GD indicator on the instance DMR1200 on which NSGA-II and SPEA2 were beaten by PAES.

Mainly the best two algorithms are NSGA-II and SPEA2 with no significant difference between them. Therefore, we make use of the runtime comparison to differentiate between them. Figure 4.14 shows the average runtime of the algorithms for the instance DMR1200 (The same pattern applies for the other instances). The figure shows that SPEA2 suffers from the worst runtime and NSGA-II achieved the second best runtime after PAES. Although NSGA-II and SPEA2 share the same proposed genetic operators, they use different methods for defining and calculating the fitness of the solution. NSGA-II is characterized by its fast non-domination sorting which is used for ranking and therefore it beats SPEA2 regarding the runtime. The comparison indicates that NSGA-II is at least as good as the other multiobjective optimizers that are used in the comparison. Therefore, we continue using the core of NSGA-II for solving the multiobjective route planning problem.

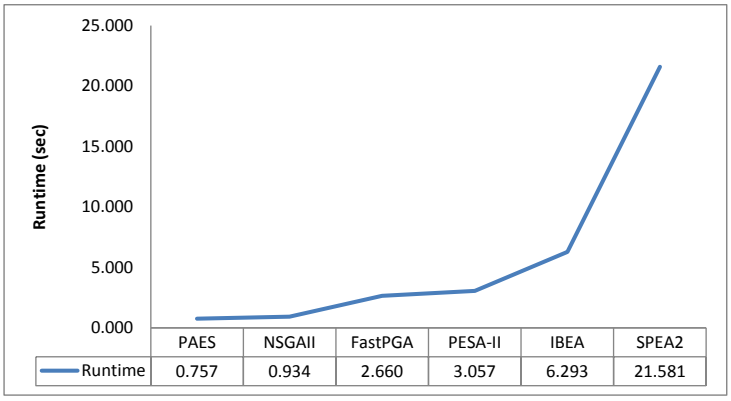


Figure 4.14: Average runtime of the different algorithms on the instance DMR1200

4.7 Alternative Solution Representation

Solution representation plays a significant role in the behavior of the genetic algorithm. We noticed that the time-expanded nature of the graph imposed some limitation on the genetic algorithm. The solution was represented as a path on the time-expanded graph representation of the offers and the algorithm tends to better explore the higher parts of the graph. This motivates us to consider the effect of a different solution representation.

4.7.1 Time dependent solution representation

We study the behavior of the genetic algorithm using a new solution representation. We call the new representation time-dependent representation as it is based on the

time-dependent representation of the drivers' offers. Very similar to the simple time-dependent model in section 3.3, the problem is modeled as a graph $G = (V, A)$. The set of vertices V represents the set of stations S and an arc $e = (u, v)$ is inserted between stations u and v if there exists at least one offer with source station u and destination station v. An arc $e = (u, v)$ is annotated with the set of offers that share the same source station u and destination station v. For performance issues and not for correctness, we keep the set of offers annotating each arc ascendingly sorted regarding their departure times.

Figure 4.15 shows the time-dependent graph representation of the following example offers: $o_1 = (s_1, s_2, t_1, t_5, d_1, c_1)$, $o_2 = (s_1, s_2, t_6, t_{11}, d_2, c_2)$, $o_3 = (s_2, s_3, t_7, t_{10}, d_4, c_3)$, $o_4 = (s_4, s_3, t_6, t_{13}, d_4, c_4)$. The arc connecting stations s_1 and s_2 is annotated with offers o_1 and o_2 which share the same source and destination stations.

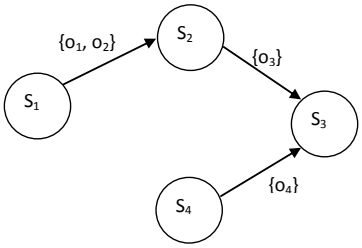


Figure 4.15: Time-dependent graph representation of the drivers' offers.

The solution is represented as a list p of offers that form a route plan from the source station $s_s(q)$ to the destination station $s_t(q)$ of the rider's request q as defined in Section 3.2. To create an initial solution, the Random Walks method is utilized to find a path, represented as a list of arcs without taking time-dependency into account, on the time-dependent graph between the source station $s_s(q)$ and the destination station $s_t(q)$. A route plan is created by randomly selecting offers, which satisfy the spatial and temporal connectivity, from the set of offers annotating the edges in the path. If no route plan can be created using the current path, another path is selected using RW and the process is repeated. Figure 4.16 shows an example time-dependent solution representation.

o_1	o_2	o_9	o_7	o_{11}	o_8	o_{10}	o_{13}

Figure 4.16: Time-dependent solution representation

The crossover operator, defined based on the time-dependent solution representation, is a single point crossover. Given two solutions, one offer is randomly selected from one solution and a search for the same offer is done in the second solution. If the offer is found in the second solution, then the two solutions are crossed after the positions of the offer in the solutions. If the offer is not found in the second solution, then another offer is randomly selected from the first solution and the process repeats until finding an offer in the two solutions or canceling the crossover operation otherwise. Figure 4.17

is an example crossover operation for two solutions with time-dependent representation. The crossover for the time-dependent solution representation has the same effect of the crossover operator for the time-expanded solution representation. It is closed to the set of offers and stations that are included in the set of solutions to be crossed.

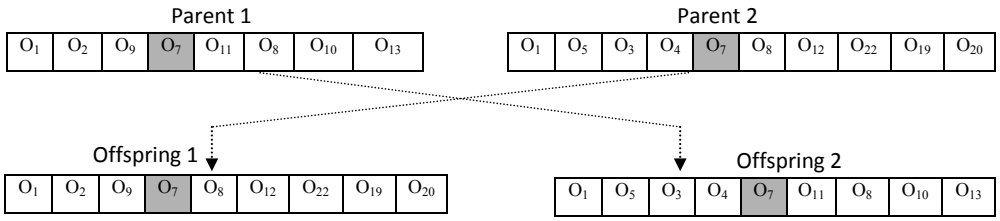


Figure 4.17: Single point crossover. Shaded offer represent crossover offer

Given a solution that consists of a list of offers on a specific path, the mutation operator should enable the algorithm to discover new route plans on the same path and also to explore new paths to build route plans upon them. This is important to avoid the problem of local optima and to make the whole search space reachable by the algorithm. For this operator to be more clear, a pseudo code in Algorithm 4 is provided.

Algorithm 4: Mutation Operator

input : Solution P, time_dependent graph G
output: mutation of P

```
1  begin
3      n ⟵ length(P)
5      i ⟵ Random[1, n]
7      j ⟵ i + Random[0, n]
9      if j > n then
11         j ⟵ n
13     if i = j then
14
16         offers ⟵ offers annotating the arc (S_s(P_i), S_t(P_i))
18         P_i ← select offer o from offers with t_d(o) ≥ t_a(P_{i−1}) and t_a(o) ≤ t_d(P_{i+1})
           // special cases when i = 0 or j = n are left for simplicity
20
21     else
22
24         subpath ⟵ random walk on G from s_s(P_i) to s_t(P_j)
26         subplan ⟵ create subplan on subpath
28         P ← P_[1:i−1] ‖ subplan ‖ P_[j+1:n]
           // ‖ is a list concatenation operator
```

Two offers p_i and p_j, $i \leq j$, are randomly selected from a given solution p. If the same offer p_i is selected twice, then an alternative offer, that satisfies the temporal connectivity, is randomly selected from the offers annotating the arc $e = (s_s(p_i), s_t(p_i))$ (Alg. 4 line 18). This results in a route plan on the same path, but with different offers. If the two selected offers are different, then the RW is utilized to find a subpath between the source station of the first offer $s_s(p_i)$ and the destination station of the second offer $s_t(p_j)$ (Alg. 4 line 24). A subplan is built on the resulted subpath between $s_s(p_i)$ and $s_t(p_j)$ such that the spatial and temporal connectivity for the whole route plan are maintained. This requires that the departure time of the first offer in the subplan to be no earlier than the arrival time of the offer before p_i in the original route plan and the arrival time of the last offer in the subplan to be no later than the departure time of the offer next to p_j in the original route plan.

4.7.2 Experimentation

We compare the behavior of the proposed genetic algorithm using the two different solution representations: The original representation which we call here time-expanded solution representation and the new time-dependent solution representation. The behavior of the algorithm using each solution representation is tested on the problem instances DMR300, DMR600, DMR900 and DMR1200 described in section 3.5. For each instance and solution representation, the algorithm is executed 30 times to find an approximation of the Pareto-optimal solutions set of route plans from station zero to station forty. The algorithm settings are all the same for the two solution representations except the mutation probability. Crossover probability, population size, and maximum generations are 1.0, 100, and 30000 respectively. Mutation probabilities are 0.4 and 0.1 per solution (chromosome) for the time-expanded and time-dependent representations respectively. The values of the parameters are set after ad hoc experiments. The value of the mutation operator for the time-dependent representation is set lower than its counterpart for the time-expanded representation because the mutation operator is computationally more expensive for the time-dependent representation. The experimentation platform is the same platform described in Section 3.5.

Table 4.16 shows the mean and standard deviation of the quality indicators of the results of the algorithm on the different problem instances using the time-expanded representation (TE) and the time-dependent representation (TD). In addition we provide boxplots for the same data in Figure 4.18.

From Table 4.16 and Figure 4.18 we notice the following. The algorithm achieved higher HV values using the time-dependent representation on all problem instances. Regarding the GD, the algorithm achieved smaller values on all problem instances, except DMR300, using the time-expanded representation. Better values of the IGD are achieved by the algorithm using the time-dependent representation on all problem instances.

Higher values of the HV mean that the algorithm covers larger part of the objective space using the time-dependent representation. This result is supported by the smaller values of the IGD using the time-dependent representation. The smaller values of the GD does not necessarily mean that the points in the approximate Pareto-front produced

Table 4.16: The values of the quality indicators of the results using the time-expanded TE and time-dependent TD solution representation on the different problem instances. Mean and standard deviation μ_σ.

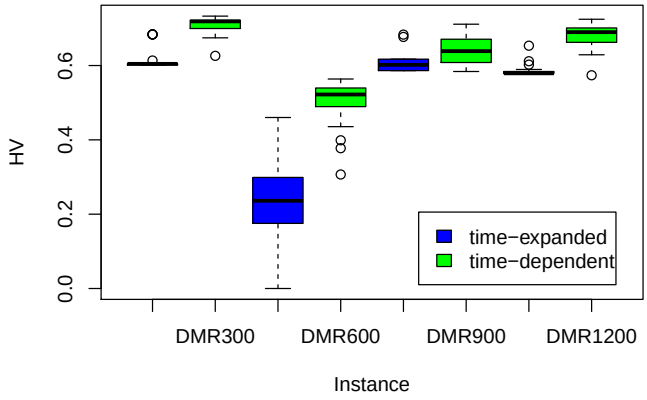
		DMR300	DMR600	DMR900	DMR1200
HV	TD	$7.08e-01_{2.1e-02}$	$5.02e-01_{5.7e-02}$	$6.42e-01_{3.5e-02}$	$6.81e-01_{2.9e-02}$
	TE	$6.14e-01_{2.8e-02}$	$2.20e-01_{1.1e-01}$	$6.04e-01_{2.4e-02}$	$5.85e-01_{1.5e-02}$
GD	TD	$2.63e-03_{2.6e-03}$	$1.07e-02_{4.8e-03}$	$7.72e-03_{2.3e-03}$	$8.27e-03_{2.9e-03}$
	TE	$6.46e-03_{6.5e-03}$	$0.00e+00_{0.0e+00}$	$0.00e+00_{0.0e+00}$	$3.44e-04_{9.2e-04}$
IGD	TD	$2.42e-02_{2.8e-03}$	$2.08e-02_{5.4e-03}$	$1.93e-02_{3.2e-03}$	$2.02e-02_{2.8e-03}$
	TE	$8.35e-02_{1.2e-03}$	$8.53e-02_{2.0e-02}$	$4.60e-02_{3.2e-03}$	$4.68e-02_{1.3e-03}$

using the time-expanded representation are closer to the points in the true Pareto-front than those produced using the time-dependent representation. To explain this, we visualize sample Pareto-fronts produced by the algorithm using the two representations in Figure 4.19. We notice from the figure that the number of elements in the Pareto-front produced using the time-dependent representation is larger than the number of elements in the Pareto-front produced using the time-expanded representation. Each element in the approximate Pareto-front using the time-expanded representation in Figure 4.19(b) exactly matches an element from the true Pareto-front with zero distance. Some of the elements in the approximate Pareto-front using the time-dependent representation in Figure 4.19(a) exactly matches some of the elements in the true-Pareto front with zero distance.

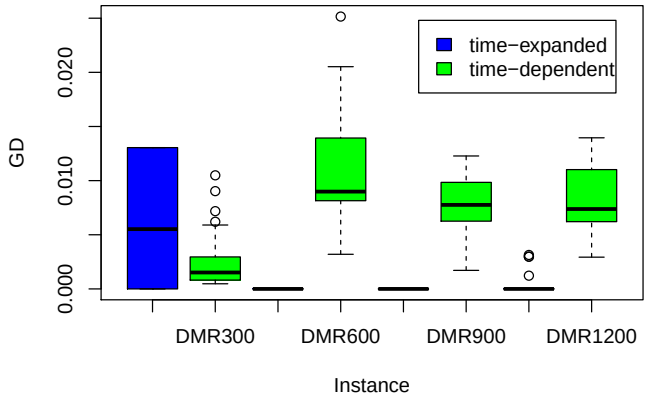
To explain the results, we utilize the major difference between the two representations that is the time expansion. The time-expanded model expands in time and the offers are modeled in ascending order regarding their timestamps. Given that we utilize the RW for solutions initialization and solutions mutation, the algorithm gets stuck at local optima and does not explore the lower parts of the graph especially if we have large number of offers that increases the probability to walk horizontally in the graph.

In other words, at station x, if the RW has the probability 50% to wait (explore later offers at x) and 50% to depart for another station, then with the increase of the number of offers at station x, the tendency of the RW is to explore solutions with smaller time (higher parts of the graph) and not to cover the whole search space. Therefore, setting the value of the RW parameter on the time-expanded model shall depend on the number of offers at each station which might introduce more complexities for the genetic approach which has many parameters to deal with.

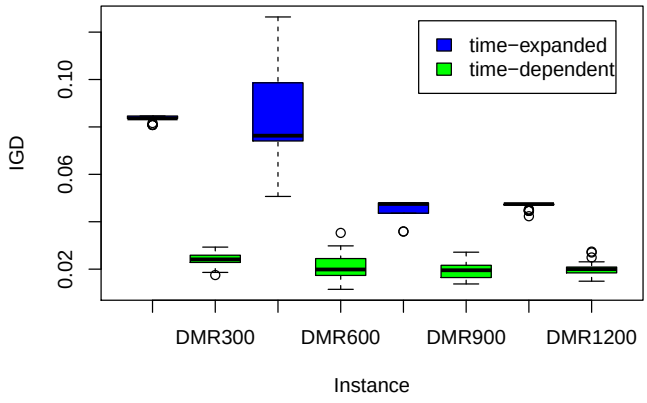
Based on the previous observation, we assume that the algorithm focuses the search on part of the time-expanded graph resulting in very close solutions to optimal ones yielding small GD. However, getting stuck at part of the search space results in missing some solutions to end up with high IGD. Using the time-dependent representation, the algorithm achieves higher HV because it produces solutions close to optimal ones and better covers the search space.



(a) Hypervolume HV

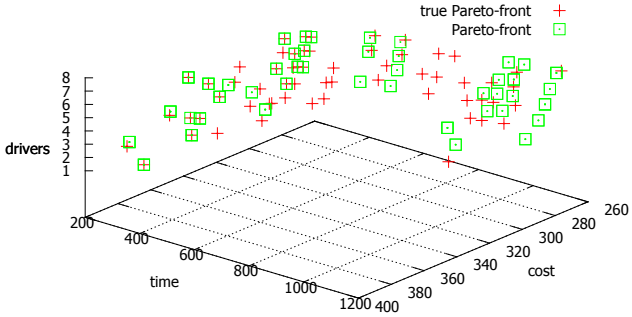


(b) Generational Distance GD

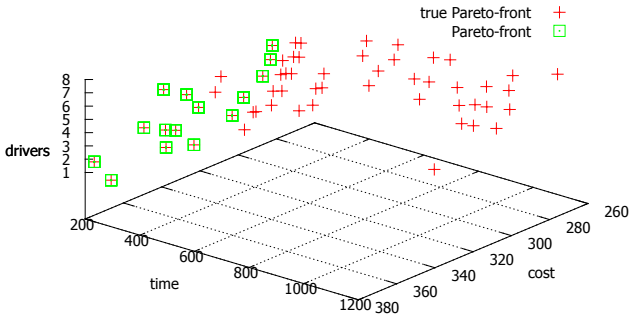


(c) Inverted Generational Distance IGD

Figure 4.18: Boxplots for the values of the quality indicators using the time-expanded and time-dependent solution representation on the different instances



(a) time-dependent representation

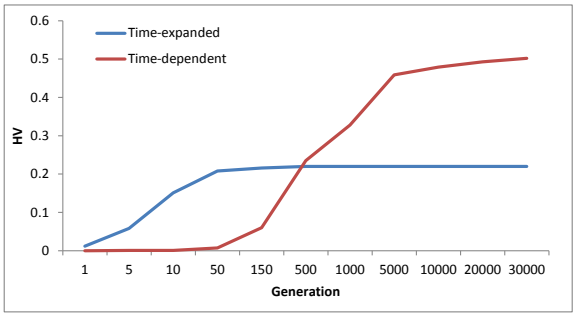


(b) time-expanded representation

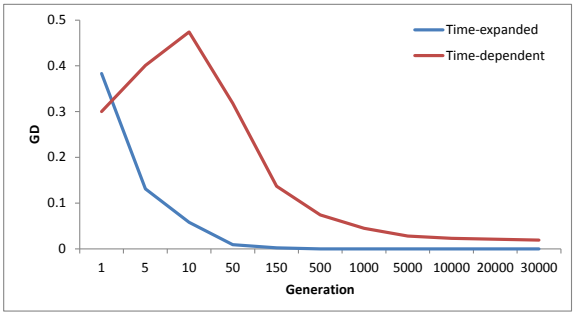
Figure 4.19: Approximate Pareto-front vs. true Pareto-front

To monitor the behavior of the algorithm when using each solution representation, we took the average values of the different quality indicators at different generations for each solution representation on the instance DMR600 (the same pattern applies for the other instances) as shown in Figure 4.20. We notice that the algorithm with the time-expanded representation converges early. After near about 500 generations, it achieved a zero GD which means that all discovered solutions are in the true Pareto-optimal set. We notice also that neither better IGD nor HV are achieved after 500 generations. This indicates that the algorithm gets stuck at local optima and parts of the search space are left without exploration as discussed earlier.

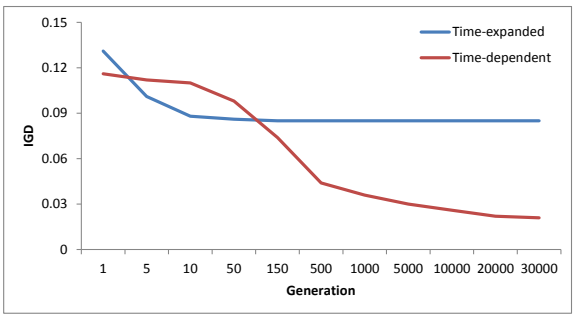
The algorithm converges slower when using the time-dependent representation. It requires more generations to achieve the same HV that is achieved when considering the time-expanded representation but it achieves higher HV after that. Also we notice that the algorithm when using the time-dependent representation converges with time (more generations) and produces smaller GD. The algorithm with time-dependent solution representation does not stuck at the local optima at which it stucks when using the time-expanded representation and continues to explores new solutions resulting in smaller IGD. Therefore, we decide to use the time-dependent solution representation throughout this work and we try to further improve the genetic algorithm by hybridizing it with a local search algorithm.



(a) Hypervolume HV



(b) Generational Distance GD



(c) Inverted Generational Distance IGD

Figure 4.20: The average values of the different quality indicators at different generations on the instance DMR600

4.8 Using Local Search to Improve the Genetic Algorithm

Recently many efforts have been directed to hybridize elements from different metaheuristics which resulted in new hybrid metaheuristics with better performance. Banos et al. [5] proposed a metaheuristic based on the hybridization of simulated annealing and Tabu search. Ishibuchi and Murata [49] and Jaszkiewicz [51] proposed two different genetic local search (GLS) algorithms as hybridization of genetic algorithms with local search. Burke and Cowling [9] have proposed a hybrid evolutionary simulated annealing algorithm for solving multiobjective optimization of space allocation problems. The hybridization of genetic algorithms and local search methods proved to be successful in many domains [51]. We try to improve the proposed genetic algorithm by providing hybridization with a local search algorithm.

Local search is an old, possibly the oldest, and simple metaheuristic method [87]. It starts with an initial solution and searches its neighborhood for a better solution that provides a larger/smaller value of the objective function. To design a local search algorithm, a neighborhood relation has to be defined. The algorithm stops when it finds the best solution in the neighborhood (best improvement). The resulting solution is usually local optima. Alternatively, the algorithm could stop when it finds the first solution with a better value of the objective function (first improvement). The problem with local search algorithms is that they usually get trapped in local optima.

Hybridizing the genetic algorithms with local search algorithms is usually fruitful. The genetic algorithm provides the local search algorithm with a promising starting solution. The local search algorithm goes through the neighborhood of the starting solution and returns a better solution to be used by the genetic algorithm to produce further better solutions using the genetic operators. This process usually speeds up the convergence of the genetic algorithm and avoids the local optima problem of the local search algorithm.

4.8.1 Genetic Local Search Algorithm

The idea of the proposed GLS algorithm is simple. We use the offspring generation created by the genetic algorithm as an input for the local search algorithm. The local search algorithm takes each solution in the offspring as a starting solution and tries to find a better solution in its neighborhood. This process creates a new, hopefully better, population from the offspring population which is merged with the parents population for environmental selection. Figure 4.21 shows the idea of the proposed GLS algorithm.

The proposed local search algorithm is simple iterated first-improvement local search as shown in Algorithm 5. Given a starting solution as a list of offers (using time-dependent solution representation), the adopted local search works as follows. It defines the neighbors of the solution to be all solutions reachable by replacing one offer with an alternative offer from the offers that annotate the same arc and satisfy the spatial and temporal connectivity. The local search stops when it finds a solution that dominates the starting solution and defines the dominating solution to be the new starting solution. The local search is repeated for each element/offer in the solution in sequence keeping the new starting solution each time (Alg. 5 line 4). Each time the genetic algorithm

provides the local search algorithm with a good starting point where the local search goes through the neighborhood searching for better solutions.

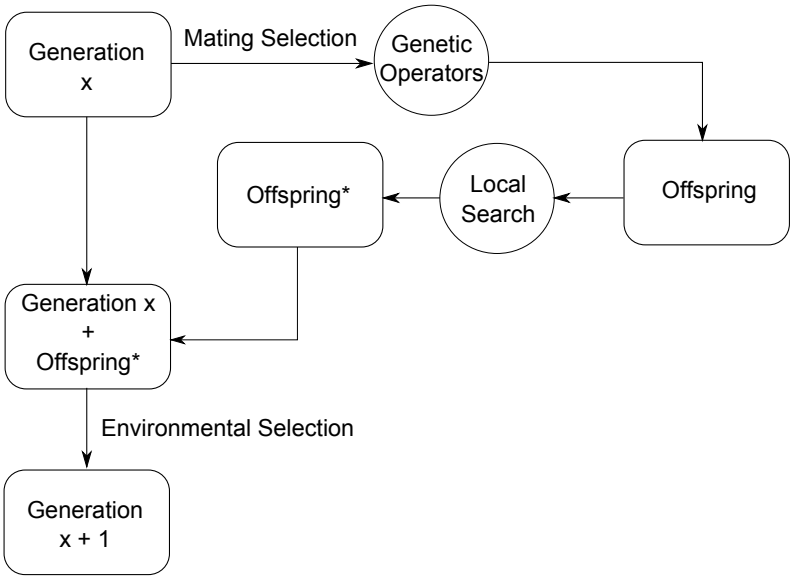


Figure 4.21: Multiobjective genetic local search algorithm.

Algorithm 5: Local Search Algorithm

input : Solution p
output: Solution p'

```
1  begin
2      bestSolution ⟵ p
3      currentSolution: x
4      for i ← 1 to length(p) step 1 do
5          x ← bestSolution
6          foreach offer o in AlternativeOffers(p_i) do
7              x_i ← o
8              Evaluate(x)
9              if x dominates bestSolution then
10                 bestSolution ← x
11                 break
12     return bestSolution
```

4.8.2 Experiments

To study the effect of the hybridization, we compare the results of the genetic algorithm (without local search) with the results of the proposed GLS algorithm. We utilize two problem instances DMR1400 and DMR700. The two problem instances are designed to be harder to solve than the previous problem instances to better understand the effect of the hybridization. The instance DMR1400 is generated on the 41 stations (nodes) network described in Section 3.5. We have generated 1400 random trips with length $x, 2 \leq x \leq 40$, on the 41 stations network which results in an instance with larger number of drivers' trips than the previous instances which makes the search space larger and the problem harder to solve. The instance DMR700 is created by the generation of 700 random trips with length $x, 1 \leq x \leq 9$, on a 10 stations (nodes) network. The 10 stations network is linear with stations 0 and 9 being the end network stations with one path connecting them. A random trip is generated by randomly selecting a trip start station, trip length and trip starting time. The trip time and cost between each two stations are set randomly and each trip is related for a unique driver. For the instance DMR700, the trips are generated on a smaller stations network (10 stations) and designed (from timing point of view) to have a high possibility of matching a rider with multiple drivers. This makes the instance harder to solve as there are many alternative solutions (larger search space). To get the true Pareto-fronts of DMR1400 and DMR700, we utilize the GLC algorithm. In order not to get out of memory, GLC compute only the values of objective functions of the route plans and does not maintain the route plans (GLC returns the true Pareto-front of the true Pareto-optimal set of route plans but not the true Pareto-optimal set itself).

We consider the linear network besides the 41 stations network in order to have large solutions' neighborhood. We generate the trips on a single path instead of generating them on a number of paths and engineer the trips at each edge to be close in time such that the number of alternative offers at each arc is large.

The genetic algorithm and the GLS algorithm share the same parameters values. Crossover probability and population size are 1.0 and 100 respectively. The mutation probability is 0.4 per chromosome (solution) for DMR1400 and 0.4 per gene (offer) for DMR700. The mutation operator has special behavior on DMR700 (linear network) with $i = j = locus\ of\ current\ gene$ in Algorithm 4. The performance of the two algorithms is compared under different number of generations. For each number of generations, each algorithm is executed 40 times to find an approximation of the Pareto-optimal solutions set of route plans from station zero to station forty on DMR1400 and from station 0 to station 9 on DMR700. The experimentation platform is the same platform described in Section 3.5.

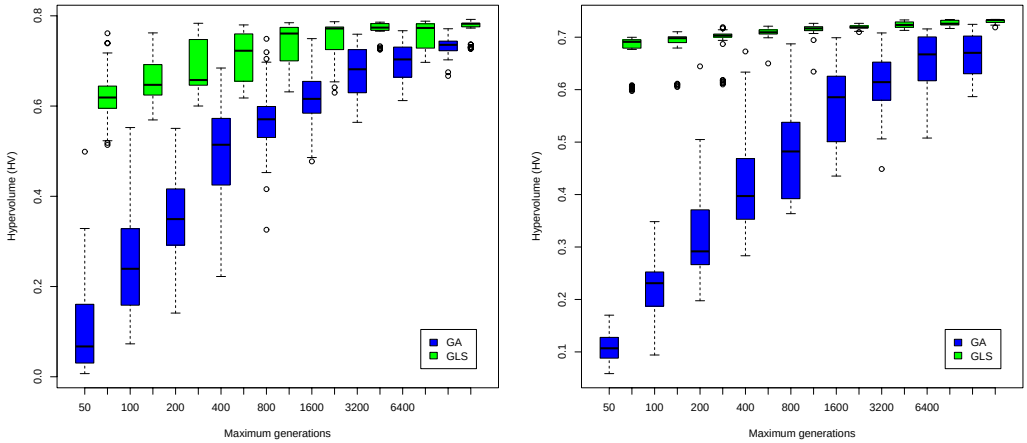
Tables 4.17 and 4.18 show the values of the quality indicators achieved by the two algorithms on the two problem instances under different numbers of generations. To improve the readability of the data, the same data is visualized as boxplots in Figure 4.22. We notice that the GLS outperforms the genetic algorithm on both instances under all generations regarding all quality indicators. However it might not be fair to compare the results of the GLS with the results of the genetic algorithm under the same number

Table 4.17: The values of the quality indicators of the results of the genetic and GLS algorithms on the instance DMR1400. Mean and standard deviation μ_σ(Better values highlighted dark gray)

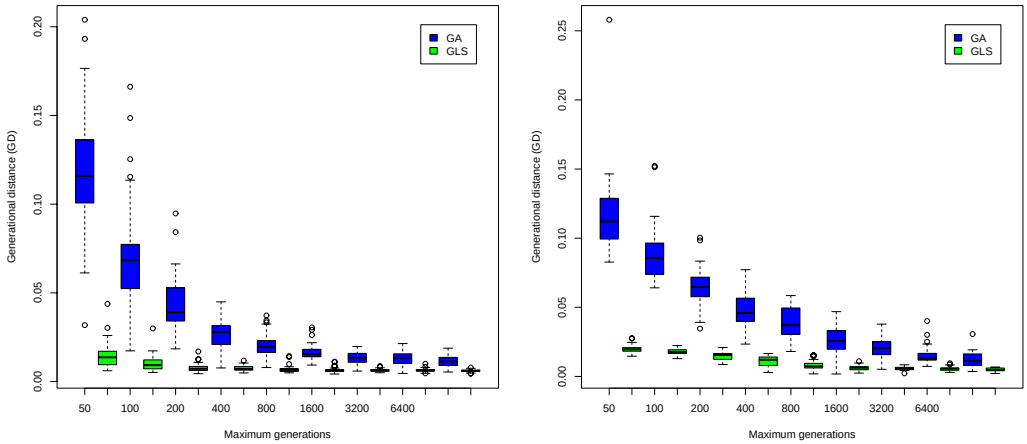
Generations	HV		GD		IGD	
	Genetic	GLS	Genetic	GLS	Genetic	GLS
50	$1.12e-01_{1.1e-01}$	$6.24e-01_{5.4e-02}$	$1.19e-01_{3.5e-02}$	$1.49e-02_{7.3e-03}$	$5.91e-02_{1.4e-02}$	$2.58e-02_{3.1e-03}$
100	$2.47e-01_{1.1e-01}$	$6.60e-01_{5.1e-02}$	$7.14e-02_{2.9e-02}$	$1.02e-02_{4.4e-03}$	$4.15e-02_{9.7e-03}$	$2.49e-02_{2.5e-03}$
200	$3.57e-01_{1.0e-01}$	$6.88e-01_{5.4e-02}$	$4.36e-02_{1.5e-02}$	$7.78e-03_{2.3e-03}$	$3.19e-02_{5.8e-03}$	$2.40e-02_{2.7e-03}$
400	$5.03e-01_{1.0e-01}$	$7.10e-01_{5.0e-02}$	$2.72e-02_{8.0e-03}$	$7.46e-03_{1.6e-03}$	$2.63e-02_{3.6e-03}$	$2.23e-02_{3.2e-03}$
800	$5.70e-01_{8.0e-02}$	$7.36e-01_{4.9e-02}$	$2.07e-02_{6.7e-03}$	$6.88e-03_{2.0e-03}$	$2.45e-02_{3.5e-03}$	$2.09e-02_{2.5e-03}$
1600	$6.18e-01_{5.6e-02}$	$7.47e-01_{4.0e-02}$	$1.64e-02_{4.4e-03}$	$6.53e-03_{1.4e-03}$	$2.28e-02_{2.7e-03}$	$1.95e-02_{2.0e-03}$
3200	$6.74e-01_{5.8e-02}$	$7.67e-01_{2.1e-02}$	$1.32e-02_{3.3e-03}$	$6.38e-03_{9.3e-04}$	$2.29e-02_{3.2e-03}$	$1.86e-02_{2.0e-03}$
6400	$6.97e-01_{4.1e-02}$	$7.56e-01_{2.7e-02}$	$1.30e-02_{3.5e-03}$	$6.45e-03_{9.9e-04}$	$2.30e-02_{3.0e-03}$	$1.72e-02_{1.9e-03}$
12800	$7.31e-01_{2.2e-02}$	$7.73e-01_{2.1e-02}$	$1.14e-02_{2.8e-03}$	$6.11e-03_{7.1e-04}$	$2.16e-02_{3.0e-03}$	$1.65e-02_{1.4e-03}$

Table 4.18: The values of the quality indicators of the results of the genetic and GLS algorithms on the instance DMR700. Mean and standard deviation μ_σ (Better values highlighted dark gray)

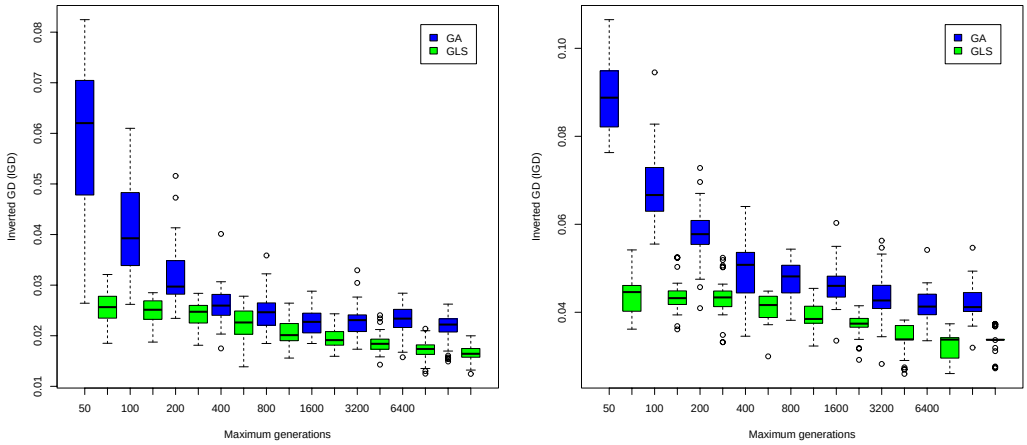
Generations	HV		GD		IGD	
	Genetic	GLS	Genetic	GLS	Genetic	GLS
50	$1.09e-01_{2.8e-02}$	$6.73e-01_{3.6e-02}$	$1.15e-01_{2.9e-02}$	$1.98e-02_{2.6e-03}$	$8.90e-02_{7.5e-03}$	$4.38e-02_{4.3e-03}$
100	$2.27e-01_{5.3e-02}$	$6.88e-01_{2.8e-02}$	$8.85e-02_{2.0e-02}$	$1.78e-02_{2.1e-03}$	$6.82e-02_{8.0e-03}$	$4.37e-02_{3.6e-03}$
200	$3.20e-01_{8.7e-02}$	$6.93e-01_{3.0e-02}$	$6.45e-02_{1.4e-02}$	$1.46e-02_{3.0e-03}$	$5.76e-02_{6.1e-03}$	$4.32e-02_{4.2e-03}$
400	$4.19e-01_{8.9e-02}$	$7.08e-01_{1.1e-02}$	$4.82e-02_{1.3e-02}$	$1.11e-02_{3.7e-03}$	$4.93e-02_{6.9e-03}$	$4.11e-02_{2.9e-03}$
800	$4.81e-01_{8.7e-02}$	$7.14e-01_{1.4e-02}$	$3.85e-02_{1.1e-02}$	$8.11e-03_{3.4e-03}$	$4.71e-02_{4.5e-03}$	$3.91e-02_{3.0e-03}$
1600	$5.66e-01_{7.5e-02}$	$7.19e-01_{3.6e-03}$	$2.63e-02_{9.4e-03}$	$6.10e-03_{1.6e-03}$	$4.62e-02_{4.5e-03}$	$3.73e-02_{2.9e-03}$
3200	$6.11e-01_{6.2e-02}$	$7.24e-01_{5.5e-03}$	$2.12e-02_{7.7e-03}$	$5.82e-03_{1.2e-03}$	$4.34e-02_{5.1e-03}$	$3.41e-02_{3.3e-03}$
6400	$6.55e-01_{5.0e-02}$	$7.26e-01_{5.0e-03}$	$1.52e-02_{6.2e-03}$	$5.70e-03_{1.2e-03}$	$4.16e-02_{3.8e-03}$	$3.26e-02_{3.4e-03}$
12800	$6.68e-01_{3.9e-02}$	$7.31e-01_{3.4e-03}$	$1.19e-02_{5.3e-03}$	$4.76e-03_{1.2e-03}$	$4.19e-02_{3.8e-03}$	$3.39e-02_{2.4e-03}$



(a) DMR1400 (b) DMR700



(c) DMR1400 (d) DMR700



(e) DMR1400 (f) DMR700

Figure 4.22: Boxplots of values of the quality indicators of the results of the genetic and GLS algorithms under different number of generations on the two problem instances.

of generations as the GLS definitely performs more operations and requires more time under the same number of generations. Therefore, in tables 4.19 and 4.20, we report the required runtime and the number of function evaluations performed by each algorithm for the two problem instances under each number of generations. The two algorithms share the same core and the same implementation of the genetic operators. The GLS has only the additional local search procedure.

Table 4.19: Mean runtime and number of function evaluations: Instance DMR1400.

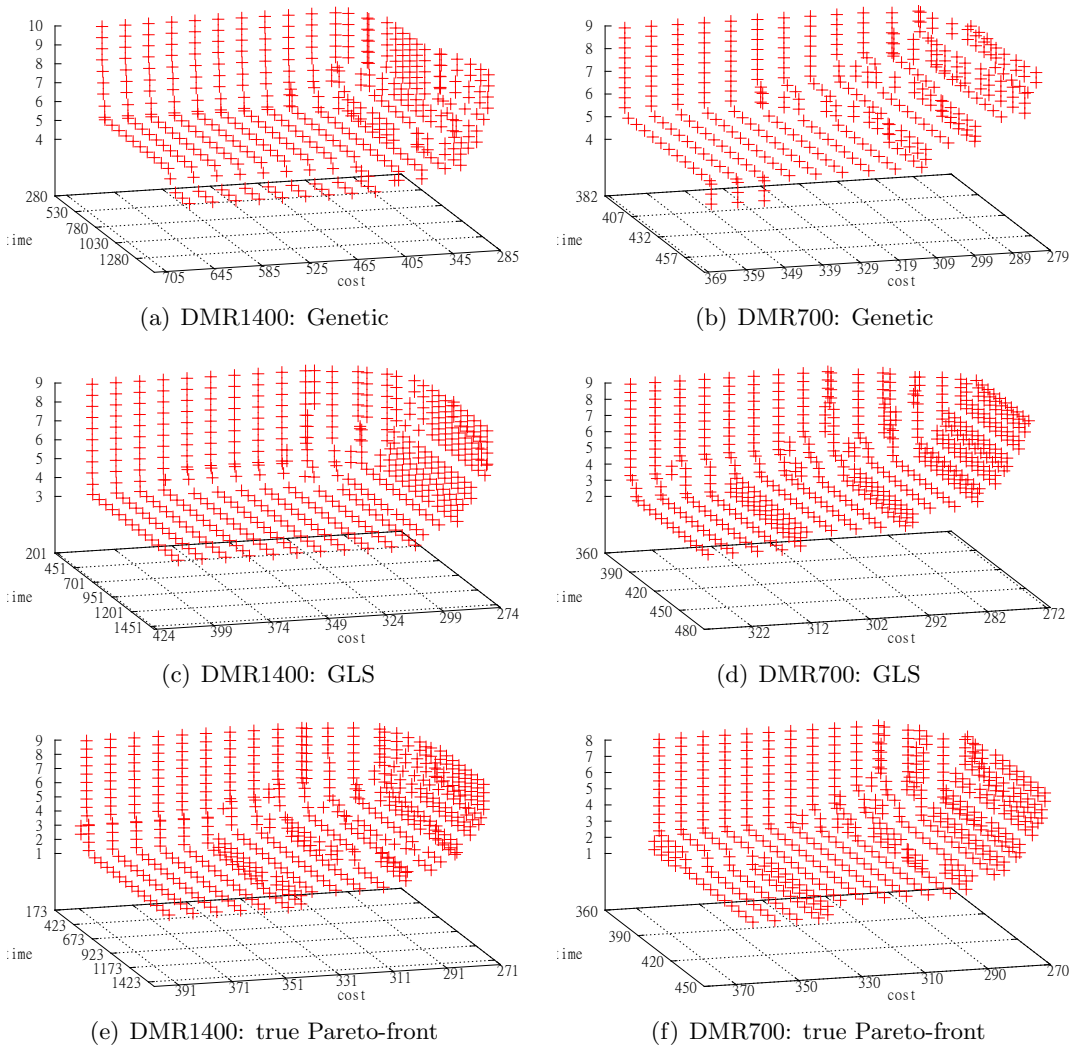
Generation	runtime (s)		thousands of function evaluations	
	Genetic	GLS	Genetic	GLS
50	1.55	1.75	5	966
100	1.98	2.44	10	1950
200	3.01	3.75	20	3940
400	4.93	6.44	40	7810
800	8.59	11.7	80	15700
1600	16.1	22.2	160	31600
3200	30.8	43.3	320	62900
6400	60.2	86.0	640	127000
12800	119.0	170.0	1280	252000

Table 4.20: Mean runtime and number of function evaluations: Instance DMR700

Generation	runtime (s)		thousands of function evaluations	
	Genetic	GLS	Genetic	GLS
50	0.203	0.842	5	1850
100	0.330	1.60	10	3940
200	0.660	3.11	20	8280
400	1.36	6.24	40	17000
800	2.92	12.5	80	34800
1600	6.18	24.8	160	71100
3200	12.4	49.9	320	145000
6400	24.7	101.0	640	295000
12800	50.0	200.0	1280	593000

From tables 4.19 and 4.20 we notice that the GLS requires relatively more time under the same number of generations. Also we notice that the time difference between the genetic and GLS algorithms is larger on the second problem instance DMR700. This is due to the larger solutions' neighborhood in DMR700 that requires the local search algorithm more time to find better solutions in the neighborhood of a given solution. However we notice that under the same number of generations, the GLS performs massively much more functions evaluations in small time difference enabling the GLS to better explore the search space. The GLS performs near about 195 times more function evaluations than the genetic algorithm under the same number of generations on DMR1400 as we can see in Table 4.19. It performs near about 400 times more function evaluations on DMR700 as we see in Table 4.20.

From Figure 4.22(a) and Table 4.17 we notice that the genetic algorithm requires 160 generations to achieve a competitive value of the hypervolume achieved by the GLS under 50 generations. In Figure 4.22(b) and Table 4.18 we see that the genetic algorithm under 12800 generations failed to achieve the value of the hypervolume achieved by the GLS under 50 generations.



(a) DMR1400: Genetic

(b) DMR700: Genetic

(c) DMR1400: GLS

(d) DMR700: GLS

(e) DMR1400: true Pareto-front

(f) DMR700: true Pareto-front

Figure 4.23: The 50% attainment surfaces of the 40 runs of the algorithms under 100 generations and the surfaces of the true Pareto-fronts

To better understand the results of the two algorithms in the objective space, we provide the 50% attainment surface of the 40 runs for each algorithm on each problem instance besides the surface of the true Pareto-fronts in Figure 4.23. The k%-attainment surface is the set of nondominated objective vectors among the objective vectors that are dominated by or equal to the output vectors of k% of the runs. As we can see that the

attainment surfaces of the results of the GLS algorithm in Figures 4.23(c) and 4.23(d) are closer to the surfaces of the true Pareto-fronts in Figures 4.23(e) and 4.23(f) as compared with the attainment surfaces of the results of the genetic algorithm in Figures 4.23(a) and 4.23(b).

4.9 Summary

In this chapter we have addressed the multiobjective route planning problem introduced in the previous chapter. We have proposed an evolutionary genetic algorithm to solve the addressed problem and compared different evolutionary algorithms in solving the same problem. The proposed algorithm was able to provide good quality route plans in reasonable time. In addition, we have considered different solution representations and studied their effect on the performance of the proposed algorithm. Finally we have improved the performance of the proposed algorithm by hybridizing it with a local search algorithm.

All the considered metaheuristics in this chapter are evolutionary algorithms. However, there are other classes of metaheuristics that could be used in solving the multiobjective route planning problem. Among the widely used metaheuristics is the swarm intelligence. In the next chapter, we try to solve the multiobjective route planning problem using ant colony optimization which is a swarm intelligence based metaheuristic. The goal is not to cover all types of metaheuristics as this is far to reach but to have a look on another class of metaheuristics that seems promising for solving this problem.

Chapter 5

Ant-Colony Multiobjective Route Planning

There are different classes of metaheuristics. In the previous chapter, we have studied the behavior of an evolutionary algorithm metaheuristic, a genetic algorithm, in solving the multiobjective route planning problem. Swarm intelligence represents another class of metaheuristics. Artificial ant colony optimization (ACO) and particle swarm optimization (PSO) are among the most successful swarm intelligence based metaheuristics. Originally ACO and PSO have been applied to combinatorial optimization problems and continuous optimization problems respectively. Later, ACO has been extended to solve continuous optimization problems and PSO has been extended to solve combinatorial optimization [87]. We use ACO to solve the multiobjective route planning problem, because it is more natural to solve combinatorial optimization problems using ACO than PSO. ACO also showed success in solving different problems such as scheduling, assignment, set covering and network routing problems [23].

We start by giving a quick overview of the ACO metaheuristic followed by discussing the extension of the ACO metaheuristic to solve multiobjective optimization problems. After that, we describe our ACO construction graph for solving the multiobjective route planning problem. Given the construction graph defined, we then discuss the use of ACO metaheuristic to solve the route planning problem on the construction graph. Finally we perform some experiments and compare the performance of the ACO metaheuristic with the performance of the genetic algorithm proposed in the previous chapter for solving the multiobjective route planning problem [1].

5.1 Ant Colony Optimization

Inspired by the behavior of nature ant colonies, M. Dorigo has proposed the ACO metaheuristic during his PhD studies [22]. Natural ant colonies consist of simple individuals

[1] The findings in this chapter are published in [39]

, ants, that present a highly structured social organization with capabilities far surpass the capabilities of single individuals. The main driving power behind ant colonies is the self-organizing property which results in highly coordinated behavior. Ants use stigmergic communication, by laying out pheromone forming pheromone trails, to coordinate their activities.

Ants are required to transport food from its source to their nest. This includes finding the shortest path between two points; the food source and the nest. In a collaborative fascinating way, ants succeed in finding the shortest path. The process of finding the shortest path by the ants works as follows. While moving between the food source and the nest, ants lay a volatile chemical substance called pheromone on the ground forming pheromone trails. An ant chooses a path probabilistically based on the amount of pheromone on it. The more the amount of pheromone on a specific path, is the more the probability that ants will follow that path. Because shorter paths require less travel time, they tend to have more amount of pheromone which results in larger amount of ants traversing them. This indirect communication between ants using pheromone trails is called stigmergic communication. A famous experiment that shows how ants converge to the shortest path is done by Goss et al. [35]. They have tried to put obstacles in the way of the ants towards the food source. The obstacles resulted in different paths between the nest and the food source. After some trials by the colony, it was able to converge to the shortest path. Figure 5.1 is an illustration of the experiment.

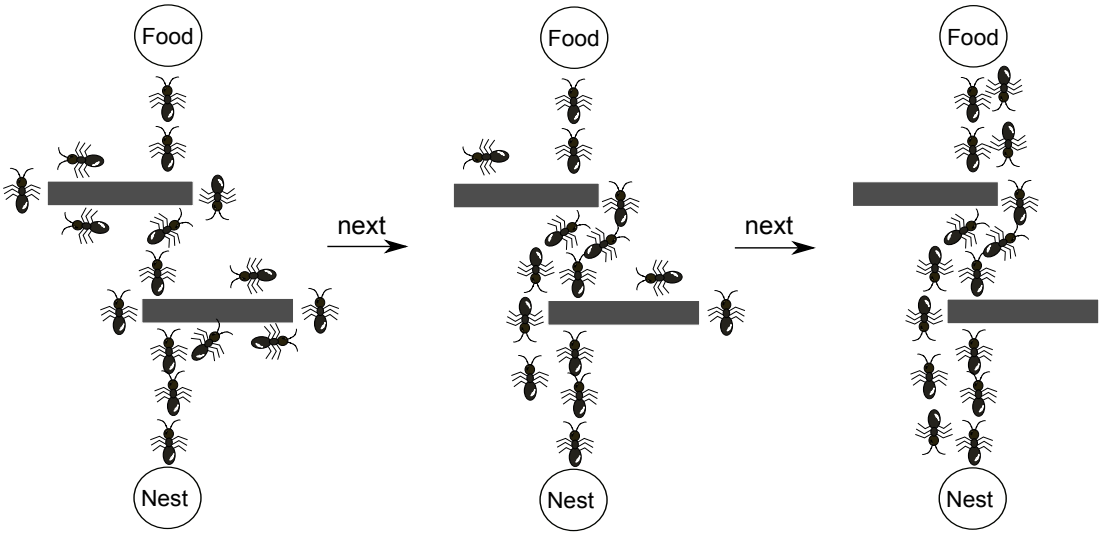


Figure 5.1: Ant colony convergence to the shortest path

The idea behind ACO metaheuristic is to mimic the behavior of the natural ant colonies using a population of artificial ants that collaborate using artificial stigmetry to solve computational problems. Besides the capabilities of natural ants, artificial ants exhibit additional capabilities like being associated with special data structures containing the memory of their previous actions and the ability to perform some daemon actions

like local search to improve their computed solutions [24].

In artificial ant colonies, a number of ants (# ants), constituting an ant colony, construct solutions on a construction graph G by gradually selecting the components of the solutions (vertices or arcs) from the graph. Components of the construction graph have pheromone structures that are used to store pheromone values to simulate pheromone trails. A solution's component v is selected by ants of colony i, $i = 1$ for single objective optimization, among a set of candidate components $Cand$ to be added to a solution with probability $P^i(v)$. $P^i(v)$ depends on the amount of artificial pheromone laid over the component and possibly on some problem specific heuristics as in Equation 5.1.

$$P^i(v) = \frac{[\tau^i(v)]^\alpha.[\eta^i(v)]^\beta}{\sum_{u \in Cand}[\tau^i(u)]^\alpha.[\eta^i(u)]^\beta} \tag{5.1}$$

where $\tau^i(v)$ is the pheromone factor (value of pheromone structure) and $\eta^i(v)$ is the heuristic factor of the component v considered by colony i and α and β are two parameters indicating their relative importance.

After solutions construction, pheromone trails are updated as in Equation 5.2. The update includes reducing the amount of the pheromone on the trails by a constant factor ρ to simulate evaporation. It also includes laying an amount of pheromone $\Delta\tau^i(v)$, by the ants with better solutions, over each component v of the better solutions to encourage the selection of the better components. The process of solutions construction and pheromone update is repeated for a specified number of cycles (Stop criteria) as shown Algorithm 6 (a basic ACO algorithm).

$$\tau^c(v) = (1 - \rho).\tau^c(v) + \Delta\tau^c \tag{5.2}$$

Algorithm 6: Basic Ant Colony Algorithm

```
begin
    Initialize the pheromone trails
    while Stop criteria not met do
        foreach ant do
            construct a solution
            perform optional daemon actions
        Evaporate pheromone trails
        Lay out pheromone by ants with better solutions
    return best solution
```

5.2 Multiobjective Ant Colony Optimization

In single objective optimization, one ant colony and one pheromone structure per construction graph component are required. The pheromone structure is mapped to the

objective function. Ants with better solutions (better values of the objective function) in the colony lay out pheromone on the solutions' components. However, in multiobjective optimization problems the mapping between the pheromone structure and the many objectives is not straightforward. How the ants optimize the different objectives using the pheromone structure is a question to be answered.

Alaya et al. [3] have defined a framework for using ACO in solving multiobjective optimization problems. They have described four different variants of the multiobjective ACO (m-ACO). The different variants are extensions of the single objective ACO. The main differences between the variants are whether to use one pheromone structure for all objectives or one for each objective and whether to use one or more ant colonies. All of the variants follow the MAX-MIN Ant System scheme [86] where pheromone values are bounded by τ_{min} and τ_{max} and initialized with τ_{max}. Following we briefly describe the four different variants of m-ACO to optimize a problem with m objectives.

In the first variant m-ACO(m+1,m), m+1 ant colonies and m pheromone structures (one for each objective) are utilized. Each ant colony i, $i \in \{1, 2, ..., m\}$, optimizes a single objective function i by considering the pheromone structure $\tau^i(v)$ to be its pheromone structure. Colony $m+1$ optimizes all objectives. The pheromone structure of colony $m+1$ is considered to be the structure $\tau^i(v)$ of the i^{th} single objective colony where i is randomly selected at each construction step. This means that the colony $m+1$ considers a randomly chosen objective to optimize at each construction step. The set of all non-dominated solutions created by all colonies is archived.

For pheromone update, pheromone is laid over the components of the best solution S^i of the i^{th} colony that minimizes the i^{th} objective function (f_i). The amount of pheromone to be laid out is defined in Equation 5.3.

$$\Delta\tau^i(v) = \begin{cases} \frac{1}{1+f_i(S^i)-f_i(S^i_{best})} & \text{if v is a} \\ & \text{component of } S^i \\ 0 & \text{Otherwise} \end{cases} \tag{5.3}$$

where $f_i(S^i_{best})$ is the best value of the objective function f_i since the begin of the run.

The second variant is $m{-}ACO^2(m{+}1, m)$. This variant is the same as m-ACO(m+1,m) with the only difference how it defines the pheromone structure of colony $m+1$. The pheromone structure of colony $m+1$ is defined as the sum of the values of the pheromone structures of all single objective ant colonies ($\sum_{i\in\{1,...,m\}} \tau^i(v)$).

In the third variant, m-ACO(1,1), one colony and one pheromone structure $\tau^1(v)$ are considered. For pheromone update, an amount of pheromone $\Delta\tau^1(v)$ is laid over the components of all non-dominated solutions constructed by the colony as in Equation 5.4. The set of all non-dominated solutions is archived.

$$\Delta\tau^1(v) = \begin{cases} 1 & \text{if v is a component of a} \\ & \text{non-dominated solution} \\ 0 & \text{otherwise} \end{cases} \tag{5.4}$$

The last variant is m-ACO(1,m). In this variant there is one colony and m pheromone structures (one for each objective). Ants randomly choose an objective $i \in \{1, ..., m\}$ to optimize at each solution construction step and the pheromone structure of the colony is defined as the pheromone structure associated with objective i.

For pheromone update, an amount of pheromone is laid over the components of the best m solutions regarding the m objective functions. Let S^i be the solution minimizing the i^{th} objective function f_i and S^i_{best} be the best value of f_i so far. The amount of pheromone $\Delta\tau^i(c)$ to be laid over the components of S^i is given in Equation 5.3.

5.3 Multiobjective Route Planning Construction Graph

We define our construction graph to solve the multiobjective route planning problem as a directed multigraph $G = (V, A)$ as shown in Figure 5.2. V is the set of ridesharing stations on the road network. A is a multiset of arcs that represent the set of offers. An arc is inserted from station s_i to station s_j for each offer with departure station s_i and arrival station s_j. Pheromone structures are associated with the arcs of the graph. Each arc is associated with 3 pheromone structures (one for each objective) for the variants m-ACO(m+1,m), $m-ACO^2(m+1, m)$ and m-ACO(1,m). For the variant m-ACO(1,1), 1 pheromone structure is associated with each edge. The multiobjective route planning problem becomes a multiobjective shortest path on the construction graph.

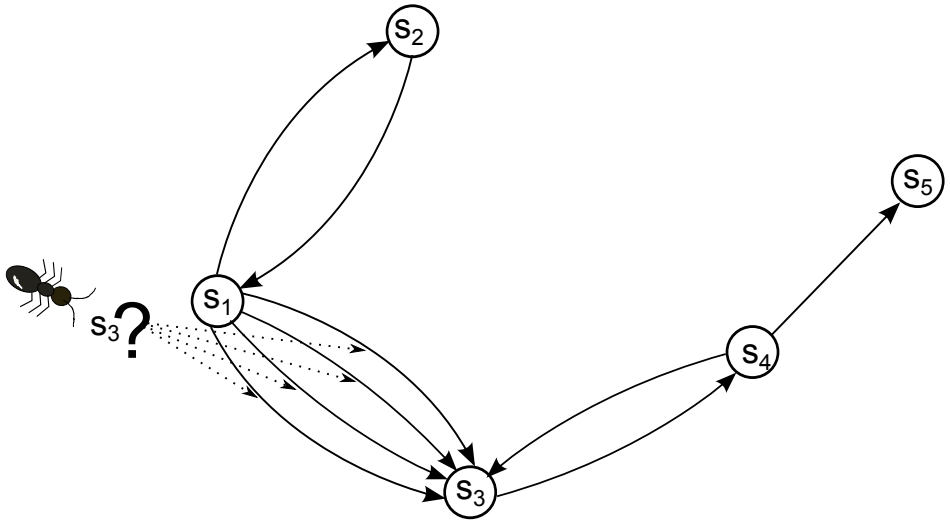


Figure 5.2: A construction graph for multiobjective route planning

5.4 Ant Colony Optimization for Multiobjective Route Planning

Given that the multiobjective route planning problem is reduced to a multiobjective shortest path problem on the construction graph, we have implemented the four variants of m-ACO to solve the multiobjective shortest path problem on the construction graph. Following we describe how m-ACO works on our construction graph. We will be a bit abstract regarding the number of pheromone structures and the number of ant colonies as they are already described in section 5.2.

Initially, the pheromone structures associated with arcs of the construction graph are initialized to a maximum value. Then each ant of each colony (if more than colony exist) constructs a solution on the construction graph. Procedure solutionConstruction shows the process of solution construction by ants. Ants start at the source station of the rider. At each construction step, ants randomly choose an adjacent vertex v (a vertex reached by one arc) to reach from vertex u. To reach the adjacent vertex, ants probabilistically select feasible arcs (u, v) based on the pheromone value on them. A feasible arc is an arc that represents an offer with departure time no earlier than the arrival time of the offer represented by the arc leading to u (maintaining temporal connectivity). This process repeats until the ants reach the destination station or reaching a dead end station. A valid ant solution is a list of arcs (offers) that represents a path from the source station to the destination station on the construction graph. To ensure that ants are not looping forever, we maintain a counter to make sure that the number of visited stations by each ant is not more than the total number of stations.

Procedure solutionConstruction

```
begin
    u ← source station
    while u ≠ destination station and u is not a dead end do
        v ← random adjacent vertex for u
        a ← probabilistically select a feasible arc (u, v)
        solution ← solution ‖ a
        u ← v
```

After solution construction, ants perform local search (daemon action) to improve their solutions. For the local search, we define the neighborhood of a solution to be the set of solutions reachable by replacing one of the arcs with another arc between the same two stations. The used local search algorithm is a simple iterated best-improvement local search the same as in Algorithm 5 but without the break at line 11.

After constructing the solutions and improving them with local search, the amount of pheromone on all arcs is reduced by a constant factor. Then the arcs of the better solutions are rewarded by increasing the amount of pheromone on them as described in Section 5.2 for the different variants of m-ACO.

5.5 Experiments

We compare the performance of the different variants of the ant colony optimizers with the performance of the genetic algorithm with time-dependent solution representation proposed in the previous chapter. We study the behavior of the ant colony optimizers with and without equipping them with local search facility.

To compare the performance of the different algorithms in solving the multiobjective route planning problem, we utilize four problem instances DMR500, DMR600, DMR700 and DMR800 in this experiment. The instances DMR500, DMR600, DMR700 and DMR800 are created by the generation of 500, 600, 700 and 800 random trips with length $x, 1 \leq x \leq 9$, on a 10 stations network. The 10 stations network is linear with stations 0 and 9 being the end network stations with one path connecting them. A random trip is generated by randomly selecting a trip start station, trip length and trip starting time. Trip length defines how many arcs to traverse where each arc traversal accounts for an offer with random cost, time and a specific driver. We did not utilize the problem instances on the 41-stations network, described in the previous chapter, because the ACO algorithm becomes very slow on these instances as explained in the experimentation section in this chapter. Hence, we have utilized easier instances (from runtime point of view) to test the ACO algorithm to have an initial idea on how it performs in comparison with the previously proposed genetic algorithm. Each algorithm is executed 20 times to find an approximation of the Pareto-optimal solutions set of route plans from station 0 to station 9 on the different problem instances. To get the true Pareto-fronts of the instances, we utilize the GLC algorithm. In order not to get out of memory, GLC compute only the values of objective functions of the route plans and does not maintain the route plans (GLC returns the true Pareto-front of the true Pareto-optimal set of route plans but not the true Pareto-optimal set itself).

The settings of the genetic algorithm are as follows. Crossover and mutation probabilities are 1.0 and 0.4 per gene (offer) respectively. Population size is 100 and the maximum generations are 5000 with a total of 500000 function evaluations. Table 5.1 summarizes the parameters values of the different variants of m-ACO. For space saving we abbreviate the different variants of m-ACO in Table 5.1 where an m-ACO variant abbreviation is combined with (*) when it is equipped with local search. We do not consider any heuristic beside pheromone values and therefore we set $\beta = 0$ for all m-ACO variants.

Table 5.1: m-ACO parameters settings

variant	abbreviation	α	ρ	# ants	# cycles	τ_{max}	τ_{min}
m-ACO(1,1)	A	2	0.0099	100	5000	300	30
m-ACO(1,m)	B	2	0.0004	100	5000	300	30
m-ACO(m+1,m)	C	2	0.0004	100	1250	300	30
m-ACO(m+1,m)^2	D	2	0.0004	100	1250	300	30

Tables 5.2 - 5.4 show the mean and standard deviation of the quality indicators regarding the different algorithms and problem instances. Tables 5.5 - 5.7 provide pairwise Wilcoxon rank sum test of the results of the different algorithms regarding each quality indicator and problem instance. The symbols $\triangledown$, $\blacktriangle$ and – mean row item is statistically significant worse than column item, row item is statistically significant better than column item and no significant difference regarding the problem instance respectively.

We start by discussing the behavior of the different variants of m-ACO. From tables 5.5 - 5.7 we notice that the variant m-ACO(1,1) (abbreviated A) performs the best among all variants of m-ACO (B, C and D). It achieved better values of the quality indicators for all problem instances. We relate this to the pheromone update strategies (that determine which solutions to lay pheromone over their components) adopted by the other variants. In the other three variants (B, C and D), the selected solutions to lay pheromone on their components are those having the smallest value of at least one objective function. While the rewarded solutions (pheromone is laid over their components) are guaranteed to be non-dominated, many other non-dominated solutions are left without laying pheromone over their components. This results in neglecting promising components. For example, consider the following cost vectors of four different solutions: (2,5,5)(5,2,5)(5,5,2) and (3,3,3). Pheromone will be laid over the components of the first three solutions but not on the components of the fourth solution despite the fact that it is Pareto-optimal solution. However, m-ACO(1,1) considers it as a solution to lay pheromone over its components as its update strategy is related to the concept of Pareto-dominance.

The local search highly improves the performance of the ant colony approach as we can see in tables 5.5 - 5.7. All m-ACO variants with local search (A^*, B^*, C^* and D^*) outperform the variants without local search (A, B, C and D). This indicates that the utilization of the local search, that finds the local optima in the neighborhood of a good starting solution provided by ant colony, was fruitful. Still m-ACO(1,1) when equipped with local search (A^*) is the best among all variants of m-ACO regarding the HV and GD quality indicators.

The genetic algorithm resulted in better values of the HV quality indicator on all instances than all variants of the m-ACO even when they are equipped with local search. Regarding the GD and IGD quality indicators, the genetic algorithm outperformed most of the m-ACO variants on most of the problem instances. It is worth mentioning that the genetic algorithm is faster than the m-ACO in all its variants especially if we consider the instances on the 41-stations network considered in the previous chapter. For the instances on the 41-stations network, the algorithm has to consider alternative physical paths between the source and destination stations which are not given as input. Because m-ACO is a constructive approach, it build solutions gradually by adding components to them by individuals (ants in our case) at each algorithm iteration (cycle). During solutions construction, many ants get lost which need to reconstruct the solutions. Ants get lost in one of two cases. The first case is to end at a station where there is no physical path to the destination station or (second case) to end at a station where the physical path exists to the destination station but no offers can be utilized to reach the destination.

Table 5.2: HV quality indicator of the results of the different algorithms on the different problem instances. Mean and standard deviation μ_σ.

	A	B	C	D	A*	B*	C*	D*	Genetic
DMR500	$2.71e-01$ $_{4.1e-02}$	$4.44e-02$ $_{7.0e-03}$	$4.32e-02$ $_{7.0e-03}$	$4.22e-02$ $_{5.3e-03}$	$4.50e-01$ $_{1.3e-02}$	$4.41e-01$ $_{1.1e-02}$	$4.44e-01$ $_{9.7e-03}$	$4.44e-01$ $_{8.9e-03}$	$5.56e-01$ $_{4.3e-02}$
DMR600	$1.93e-01$ $_{2.7e-02}$	$4.03e-02$ $_{7.0e-03}$	$2.90e-02$ $_{3.9e-03}$	$2.73e-02$ $_{4.9e-03}$	$3.82e-01$ $_{1.5e-02}$	$3.51e-01$ $_{2.5e-02}$	$3.55e-01$ $_{2.0e-02}$	$3.57e-01$ $_{2.4e-02}$	$4.93e-01$ $_{3.8e-02}$
DMR700	$2.54e-01$ $_{4.5e-02}$	$3.55e-02$ $_{4.0e-03}$	$3.36e-02$ $_{6.4e-03}$	$3.66e-02$ $_{7.5e-03}$	$5.10e-01$ $_{2.3e-02}$	$4.49e-01$ $_{2.8e-02}$	$4.57e-01$ $_{2.8e-02}$	$4.57e-01$ $_{2.5e-02}$	$6.96e-01$ $_{1.8e-02}$
DMR800	$1.09e-01$ $_{1.8e-02}$	$3.31e-03$ $_{2.4e-03}$	$2.88e-03$ $_{2.1e-03}$	$3.91e-03$ $_{2.8e-03}$	$3.56e-01$ $_{3.6e-02}$	$3.99e-01$ $_{2.0e-02}$	$4.24e-01$ $_{3.3e-02}$	$4.24e-01$ $_{2.7e-02}$	$5.07e-01$ $_{3.3e-02}$

Table 5.3: GD quality indicator of the results of the different algorithms on the different problem instances. Mean and standard deviation μ_σ.

	A	B	C	D	A*	B*	C*	D*	Genetic
DMR500	$4.38e-02$ $_{1.2e-02}$	$1.20e-01$ $_{1.1e-02}$	$1.20e-01$ $_{6.8e-03}$	$1.24e-01$ $_{1.1e-02}$	$1.22e-02$ $_{2.7e-03}$	$1.81e-02$ $_{3.0e-03}$	$1.98e-02$ $_{1.7e-03}$	$1.99e-02$ $_{2.2e-03}$	$1.34e-02$ $_{7.2e-03}$
DMR600	$3.78e-02$ $_{3.7e-03}$	$9.37e-02$ $_{5.0e-03}$	$1.02e-01$ $_{9.0e-03}$	$1.02e-01$ $_{7.3e-03}$	$1.07e-02$ $_{8.3e-03}$	$1.78e-02$ $_{3.5e-03}$	$2.10e-02$ $_{4.5e-03}$	$1.99e-02$ $_{3.5e-03}$	$2.25e-02$ $_{6.6e-03}$
DMR700	$5.31e-02$ $_{8.7e-03}$	$1.01e-01$ $_{7.6e-03}$	$1.04e-01$ $_{7.4e-03}$	$1.07e-01$ $_{8.5e-03}$	$1.27e-02$ $_{1.0e-02}$	$3.20e-02$ $_{9.1e-03}$	$3.12e-02$ $_{6.6e-03}$	$3.11e-02$ $3.11e-02_{6.4e-03}$	$1.41e-02$ $_{3.7e-03}$
DMR800	$6.57e-02$ $_{1.1e-02}$	$1.46e-01$ $_{1.1e-02}$	$1.43e-01$ $_{9.6e-03}$	$1.41e-01$ $_{7.3e-03}$	$3.21e-02$ $_{2.6e-03}$	$3.07e-02$ $_{2.2e-03}$	$2.92e-02$ $_{4.5e-03}$	$2.94e-02$ $_{3.4e-03}$	$1.88e-02$ $_{3.3e-03}$

Table 5.4: IGD quality indicator of the results of the different algorithms on the different problem instances. Mean and standard deviation μ_σ.

	A	B	C	D	A*	B*	C*	D*	Genetic
DMR500	$4.90e-02$ $_{6.6e-03}$	$8.68e-02$ $_{4.5e-03}$	$8.61e-02$ $_{3.1e-03}$	$8.76e-02$ $_{3.0e-03}$	$3.56e-02$ $_{6.0e-03}$	$2.72e-02$ $_{3.8e-03}$	$2.55e-02$ $_{2.9e-03}$	$2.55e-02$ $_{2.9e-03}$	$2.84e-02$ $_{2.6e-03}$
DMR600	$5.79e-02$ $_{4.7e-03}$	$9.14e-02$ $_{3.4e-03}$	$9.43e-02$ $_{2.3e-03}$	$9.58e-02$ $_{3.1e-03}$	$4.50e-02$ $_{2.9e-03}$	$4.22e-02$ $_{3.2e-03}$	$4.15e-02$ $_{3.2e-03}$	$4.01e-02$ $_{3.2e-03}$	$3.93e-02$ $_{1.3e-02}$
DMR700	$6.92e-02$ $_{8.2e-03}$	$1.18e-01$ $_{3.5e-03}$	$1.18e-01$ $_{4.0e-03}$	$1.17e-01$ $_{4.5e-03}$	$4.77e-02$ $_{6.5e-03}$	$4.82e-02$ $_{7.1e-03}$	$4.36e-02$ $_{5.9e-03}$	$4.50e-02$ $_{6.2e-03}$	$4.08e-02$ $_{3.2e-03}$
DMR800	$7.68e-02$ $_{3.3e-03}$	$1.13e-01$ $_{3.3e-03}$	$1.13e-01$ $_{2.7e-03}$	$1.11e-01$ $_{4.9e-03}$	$3.68e-02$ $_{4.4e-03}$	$3.15e-02$ $_{5.1e-03}$	$2.74e-02$ $_{5.3e-03}$	$2.80e-02$ $_{5.8e-03}$	$3.08e-02$ $_{4.7e-03}$

Table 5.5: HV pairwise Wilcoxon rank sum test of the results of the algorithms on the different problem instances in the order DMR500, DMR600, DMR700, DMR800 (names of the instances are hidden in each column for space saving).

	B				C				D				A*				B*				C*				D*				Genetic			
A	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽
B					–	▲	–	–	–	▲	–	–	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽
C									–	–	–	–	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽
D													▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽
*A**																	–	▲	▲	▽	–	▲	▲	▽	–	▲	▲	▽	▽	▽	▽	▽
*B**																					–	–	–	▽	–	–	–	▽	▽	▽	▽	▽
*C**																									–	–	–	–	▽	▽	▽	▽
*D**																													▽	▽	▽	▽

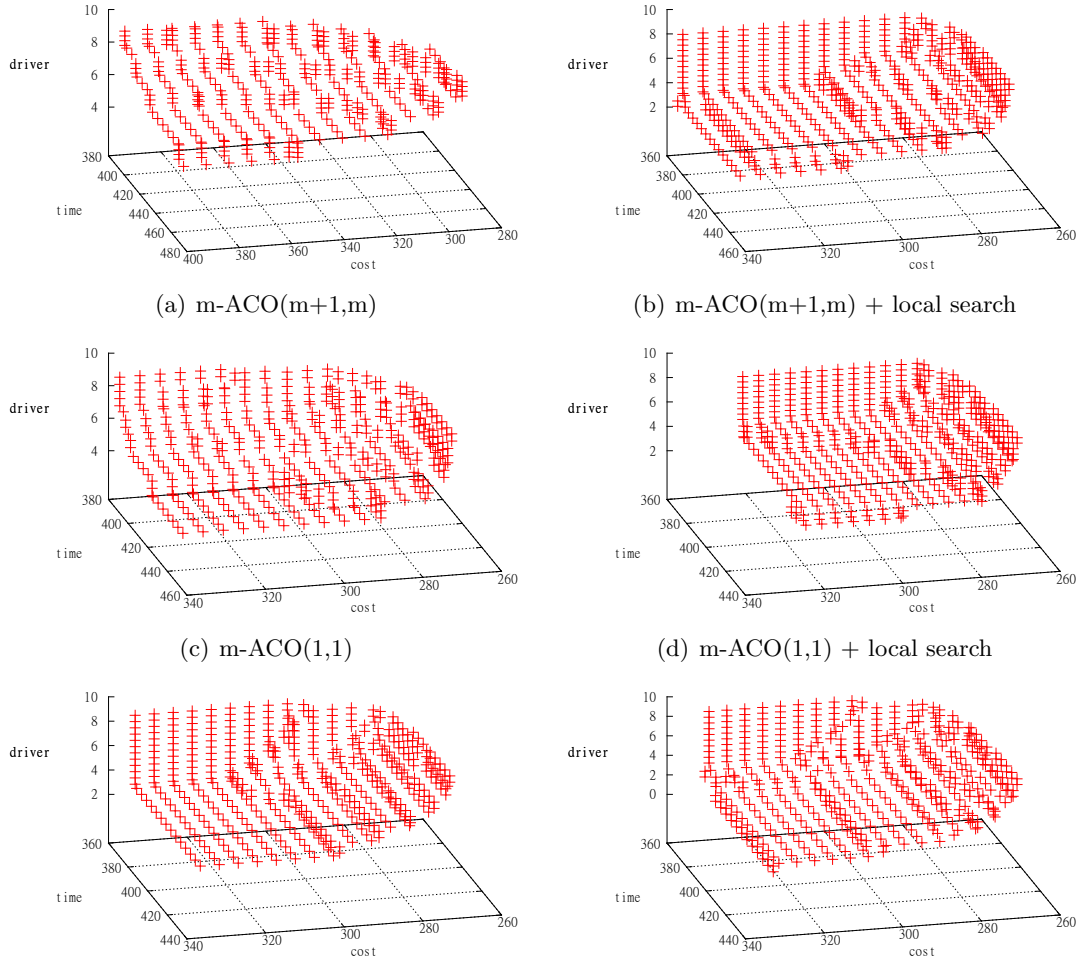
Table 5.6: GD pairwise Wilcoxon rank sum test of the results of the algorithms on the different problem instances in the order DMR500, DMR600, DMR700, DMR800 (names of the instances are hidden in each column for space saving).

	B				C				D				A*				B*				C*				D*				Genetic			
A	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽
B					–	▲	–	–	–	▲	▲	–	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽
C									–	–	–	–	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽
D													▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽
*A**																	▲	▲	▲	–	▲	▲	▲	▽	▲	▲	▲	▽	–	▲	–	▽
*B**																					▲	▲	–	▽	▲	–	–	–	▽	▲	▽	▽
*C**																									–	–	–	–	▽	–	▽	▽
*D**																													▽	–	▽	▽

Table 5.7: IGD pairwise Wilcoxon rank sum test of the results of the algorithms on the different problem instances in the order DMR500, DMR600, DMR700, DMR800 (names of the instances are hidden in each column for space saving).

	B				C				D				A*				B*				C*				D*				Genetic			
A	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽
B					–	▲	–	–	–	▲	–	–	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽
C									–	▲	–	–	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽
D													▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽
*A**																	▽	▽	–	▽	▽	▽	–	▽	▽	▽	–	▽	▽	–	▽	▽
*B**																					–	–	–	–	–	▽	–	–	–	–	▽	–
*C**																									–	–	–	–	▲	–	–	–
*D**																													▲	–	–	–

To better understand the objective space and the results of the different variants of m-ACO and the genetic algorithm, we plot the 50% attainment surface of the 20 runs of m-ACO(m+1,m)^2, m-ACO(m+1,m)^2 with local search, m-ACO(1,1), m-ACO(1,1) with local search and the genetic algorithm besides the surface of the true Pareto-front in Figure 5.3 for the instance DMR500. The behavior of the other variants of m-ACO is almost represented by m-ACO(m+1,m)^2 and its local search version so we do not show their plots for space saving. The k%-attainment surface is the set of nondominated objective vectors among the objective vectors that are dominated by or equal to the output vectors of k% of the runs.



(a) m-ACO(m+1,m)

(b) m-ACO(m+1,m) + local search

(c) m-ACO(1,1)

(d) m-ACO(1,1) + local search

(e) genetic

(f) True Pareto-front

Figure 5.3: The 50% attainment surfaces of the 20 runs and the surface of the true Pareto-front for the instance DMR500.

Figures 5.3(a) and 5.3(c) present the worst two attainment surfaces for m-ACO with higher values for all objective functions. However, still the attainment surface of the

m-ACO(1,1) is better than m-ACO(m+1,m)^2. After equipping the m-ACO algorithms with local search we notice the improvements in the attainment surfaces in Figures 5.3(b) and 5.3(d).

From Figures 5.3(b), 5.3(d) and 5.3(e) we notice that the attainment surfaces of the m-ACO variants are comparable with the attainment surface of the genetic algorithm after equipping them with local search. Nevertheless m-ACO(m+1,m) with local search suffer from higher values of the objective functions and m-ACO(1,1) missed part of the search space between 320 and 340 on the cost dimension. Still the bordering attainment surface of the true Pareto-front, Figure 5.3(f), is the one achieved by the genetic algorithm.

5.6 Summary

In this chapter we have proposed an ant colony optimization algorithm for solving the multiobjective route planning problem introduced in Chapter 3. The results of the ant colony algorithm are compared with the results of the genetic algorithm proposed in the previous chapter. Experimentation results indicate that the genetic algorithm is more suitable for solving the multiobjective route planning in terms of solution quality and runtime.

The ridematching variant considered in this chapter and in the previous two chapters is the multiple drivers and single rider ridematching where a rider can be matched with multiple drivers at different times and a driver can be matched with only one rider. We did not consider time windows for drivers' offers and riders' requests. In the next chapter, we consider another variant of the ridematching problem. The considered variant in the next chapter is the single driver and multiple riders ridematching with time windows where a rider can be matched with only one driver and a driver can be matched with multiple riders.

Chapter 6

Single Driver and Multiple Riders Ridematching

In this chapter, we address the single driver and multiple riders (SDMR) ridematching problem . In this ridematching variant, a driver can be matched with multiple riders and a rider can be matched with only one driver. The problem is considered with time windows. An example problem is shown in Figure 6.1. The problem consists of a set of drivers' offers (representing a set of vehicles) and riders' requests. Each participant (driver or rider) specifies a source and a destination location for her/his trip. In addition, each participant specifies an earliest departure time and a latest arrival time which we use to define a time window at each location as we show later. Drivers define maximum travel time and distance for their trips and they are willing to detour to take some riders. Riders define maximum travel time for their trips. The main task in this problem is to assign riders to drivers (matching) and to determine the timing and order of the pickup and delivery of riders (routing of drivers). We will use the words vehicle and driver and the phrases matched request and served request interchangeably.

The single driver and multiple riders ridematching with time windows is an optimization problem and it is subject to multiobjective optimization. In this work we consider the following objectives to optimize:

c_1. The total distance of vehicles' trips (to be minimized).

c_2. The total time of vehicles' trips (to be minimized).

c_3. The total time of the trips of the matched (served) riders' requests (to be minimized).

c_4. The number of matched riders' requests (to be maximized).

We allow drivers/vehicles to detour to serve riders and also drivers can wait for riders in this variant of ridematching. Therefore it is important to consider the vehicles' total travel distance and time as objectives to minimize. The trip's time (including the waiting time) is usually more important for the rider than the trip's distance, and therefore we

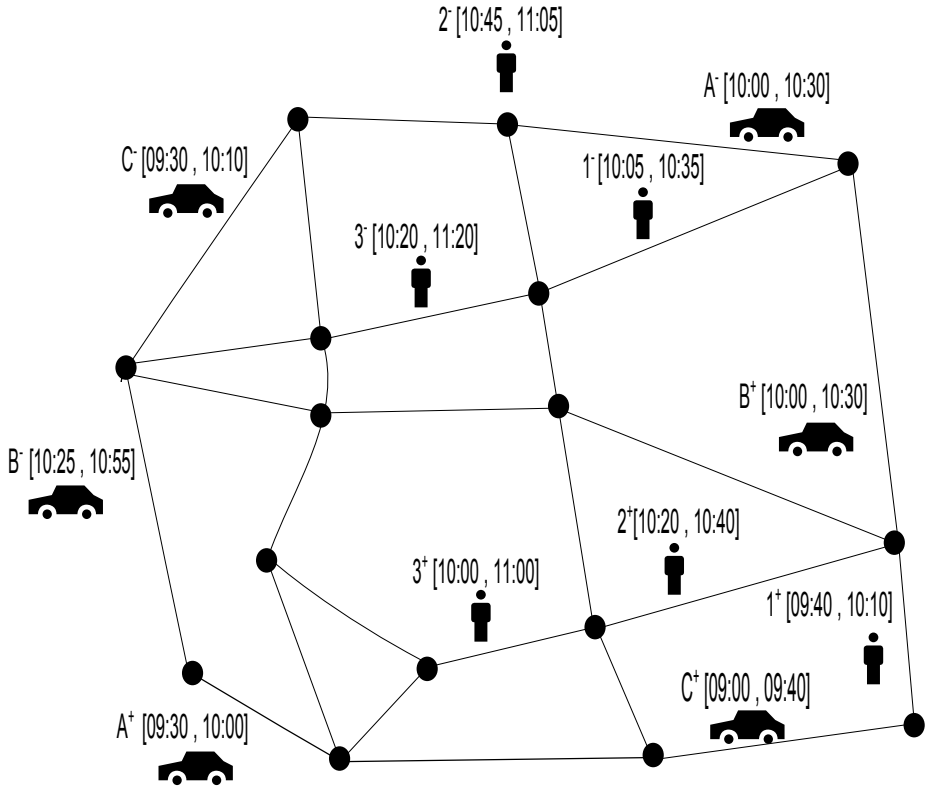


Figure 6.1: The single driver and multiple riders ridematching problem with time windows. + and - denote sources and destinations respectively of the ridesharing participants on an artificial road network.

consider the total time of the trips of the served riders' requests as an objective to be minimized. We consider the number of served riders' requests as an objective to be maximized as serving the largest number of riders is a priority in any ridesharing system. Note that the trip's time might include some waiting time (A vehicle with possibly some riders is waiting to pick up a rider when his time window allows). Criterion c_4 is conflicting with the other three criteria. Also criteria c_1 and c_2 could be conflicting with criterion c_3 as vehicles might tend to take riders for longer time to reduce their own time and distance. We do not minimize each objective independently for this variant of ridematching, but we construct an objective function as the weighted sum of the different objectives as we show later in this chapter. This is because, for this variant of ridematching, we do not perform the matching per a rider's request as for the previous variant, but we perform it on a batch of riders' requests and drivers' offers. Hence, the decision maker will be the ridematching server and not the rider who can use the returned Pareto-front to choose a solution as before.

We start by providing a formal definition of the single driver and multiple riders ridematching with time windows, and then we describe and test the proposed genetic

algorithm to solve the ridematching problem. After that we describe and test an algorithm that alternate between the proposed genetic algorithm and an insertion heuristic to solve the ridematching problem in a more realistic scenario[1].

6.1 Problem Definition

The single driver and multiple riders ridematching with time windows is similar to the pickup and delivery problem with time windows (PDPTW) especially the dial-a-ride problem (DAR) which is a hard optimization problem. Therefore the formulated mathematical model is very similar to the DAR problem model in [53] which is in turn an extension to the pickup and delivery problem model in [90] and [6]. The main difference between our problem and the DAR problem is that, in our problem the vehicles are not free to move everywhere\when and limited by the vehicles' sources and destinations (with possible detour flexibility) and time windows. We adapt the model of DAR problem for our problem and modify the definition of the objective function and add additional constraints on it (Constraints 6.16, 6.17 and 6.18 in the model).

The problem consists of a set $R = \{1, 2, ..., n\}$ of n riders' requests and a set $V = \{2n + 1, 2n + 2, ..., 2n + v\}$ of v vehicles representing drivers' offers, $R \cap V = \varnothing$. For each rider's request $i \in R$, we have a pickup point i, a delivery point $i + n$, demand dm_i defining the number of persons to be delivered from point i to point $i + n$, earliest departure time ED_i from point i, latest arrival time LA_i to point $i + n$ and constants AT_i and BT_i to define the rider's maximum travel time MTT_i. Let $t_{i,j}$ be the direct travel time between points i and j. We compute MTT_i as done in [52]:

$$MTT_i = AT_i + BT_i \cdot t_{i,i+n}.$$

For each pickup point i, we compute a time window $[a_i,b_i]$ (Figure 6.2), denoting the earliest and latest pickup time, as follows:

$$a_i = ED_i$$
$$b_i = LA_i - t_{i,i+n}$$

A time window $[a_{i+n},b_{i+n}]$ (Figure 6.2) denoting the earliest and latest delivery time for each delivery point $i + n$ is computed as follows:

$$a_{i+n} = ED_i + t_{i,i+n}$$
$$b_{i+n} = LA_i$$

For each vehicle $k \in V$ we have a source point k, destination point $k + v$, a maximum capacity C^k, earliest departure time ED_k from point k, latest arrival time LA_k to point $k + v$, and constants AT_k, BT_k, AD_k and BD_k to define the vehicle's maximum travel time MTT_k and distance MTD_k. The calculation of the time windows for the points k and $k + v$ ($[a_k, b_k]$ and $[a_{k+v}, b_{k+v}]$), that denote the earliest and latest departure

[1] The findings in this chapter are published in [45, 46]

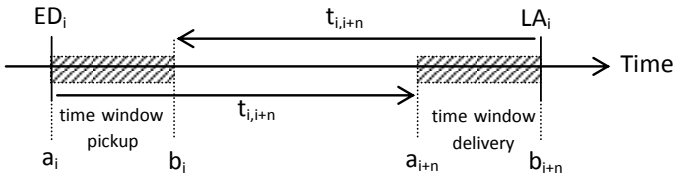


Figure 6.2: Time window calculation at pickup point i and delivery point $i+n$ for request i (Adapted from [52]).

time and the earliest and latest arrival time of vehicle k respectively, is the same as for riders' points. The calculation of the maximum vehicle's travel time MTT_k is the same as for riders. Let $d_{i,j}$ denote the direct travel distance between points i and j, then the maximum vehicle's travel distance MTD_k is:

$$MTD_k = AD_k + BD_k \cdot d_{k,k+v}.$$

Let $P = \{1, ..., n\}$ denote the set of pickup points and $D = \{n+1, ..., 2n\}$ denote the set of delivery points. The set of all pickup and delivery points is $N = P \cup D$ and the set of all points including the vehicles' sources and destinations is $A = N \cup \{k, k+v\}$ $\forall k \in V$. Let $l_i = dm_i$ and $l_{i+n} = -dm_i$ represent the load change at points i and $i+n$ respectively. The load after servicing point i is L_i^k. Let T_i^k denote the service starting time at point i by vehicle k. It denotes the pickup and delivery times for riders and the departure and arrival times for vehicles (drivers). The service time at point $i \in N$ (required time for pickup or delivery) is s_i and $\forall i \in A \setminus N$, $s_i = 0$. A binary decision variable $x_{i,j}^k$ is set to 1 if vehicle k services point i and travels directly to point j to service it. It is set to 0 otherwise. Table 6.1 summarizes the problem variables and parameters.

Following we explain the calculation of each component of the objective function, and then we show how we formulate the multiobjective function of the ridematching problem. The total distance of vehicles' trips is calculated by summing the travel distances for all vehicles. The travel distance for each vehicle is calculated by summing the travel distances between each two consecutive points visited by the vehicle starting from its source and ending by its destination as in Equation 6.1.

$$\sum_{k \in V} \sum_{i,j \in A} x_{i,j}^k d_{i,j} \tag{6.1}$$

The calculation of the total travel time of vehicles' trips is done by summing the travel time for all vehicles. The travel time for a vehicle is calculated as the time difference between its arrival to its destination and its departure from its source as in Equation 6.2.

$$\sum_{k \in V} (T_{k+v}^k - T_k^k) \tag{6.2}$$

Table 6.1: Summary of problem variables and parameters

R	The set of riders' requests
V	The set of drivers' offers (vehicles)
P	The set of riders' pickup points
D	The set of riders' delivery/drop-off points
N	$P \cup D$
A	$N \cup$ (drivers' source and destination points)
dm_i	Demand of request i, $i \in R$ (how many persons)
l_i	Load change at point i, $i \in N$
L_i^k	Load change of vehicle k after servicing point i, $i \in N$
C^k	Capacity of vehicle k, $k \in V$
s_i	Service time at point i
$x_{i,j}^k$	A binary decision variable. 1 means vehicle k services point i and then directly point j , $i, j \in A$
$t_{i,j}$	Direct travel time between i and j
$d_{i,j}$	Direct travel distance between points i and j
a_i , b_i	Earliest and latest pickup/departure or delivery/arrival at a specific point (location) i, $i \in A$. $[a_i, b_i]$ defines a time window
T_i^k	The service starting time at point i by vehicle k
ED_i	Earliest pickup/departure time $i \in R \cup V$
LA_i	Latest delivery/arrival time $i \in R \cup V$
AT_i, BT_i	Parameters to determine the maximum allowed travel time MTT_i, $i \in R \cup V$
AD_k, BD_k	Parameters to determine the maximum allowed travel distance for vehicles MTD_k, $k \in V$

The total riders travel time is calculated by summing the travel time for each rider as in Equation 6.3. The travel time for each rider is calculated as the time difference between his arrival to his destination and his departure from his source. The time of departure is calculated as the vehicle service start time at the pickup point of the rider plus the service time required to pick up the rider as in Equation 6.3.

$$\sum_{k \in V} \sum_{i \in R} max\{0, (T_{i+n}^k - s_i - T_i^k)\} \tag{6.3}$$

The number of served riders' requests is calculated by summing the number of all visited points by all vehicles then dividing the sum by two to get the total number of pairs of points visited by the vehicles. There are v pairs related to vehicles' sources and destinations which are subtracted from the sum to get the total number of served riders' requests as in Equation 6.4.

$$\frac{1}{2}(\sum_{k \in V} \sum_{i,j \in A} x_{i,j}^k - v) \tag{6.4}$$

The multiobjective function that minimizes the total distance and time of the vehicles' trips and the total time of the trips of the matched riders' requests and maximizes the number of matched riders' requests r is provided in Equation 6.5. The weights α, β, γ

and δ define the relative importance of the different components. Note that to maximize the number of served riders' requests, we minimize the number of non-served requests (n is the number of riders' requests).

$$\begin{aligned} minimize\ \alpha \sum_{k \in V} \sum_{i,j \in A} x_{i,j}^k d_{i,j} + \beta \sum_{k \in V} (T_{k+v}^k - T_k^k) + \\ \gamma \sum_{k \in V} \sum_{i \in R} max\{0, (T_{i+n}^k - s_i - T_i^k)\} + \\ \delta (n - \frac{1}{2} (\sum_{k \in V} \sum_{i,j \in A} x_{i,j}^k - v)) \end{aligned} \tag{6.5}$$

Now we discuss the necessary constraints for a proper formulation of the SDMR ridematching problem with time windows. The number of vehicles leaving their sources should be equal to the number of vehicles arriving to their destinations. This is ensured by the two constraints in Equations 6.6 and 6.7.

$$\sum_{k \in V} \sum_{j \in P \cup \{k+v\}} x_{k,j}^k = v \tag{6.6}$$

$$\sum_{k \in V} \sum_{i \in D \cup \{k\}} x_{i,k+v}^k = v \tag{6.7}$$

In this ridematching problem variant, a riders' request can be served at most by one vehicle. This is ensured by the constraint in Equation 6.8. The constraint ensures that for a given pickup/delivery point i, there should be at most one vehicle that visits the point i and then visits some point j.

$$\sum_{k \in V} \sum_{j \in A} x_{i,j}^k \leq 1, \forall i \in N \tag{6.8}$$

As a riders' request could be served at most by one vehicle, both the pickup and delivery of the rider should be performed by the same vehicle. This is ensured by the constraint in Equation 6.9 by ensuring that if a vehicle visits a pickup point i, then it should visit its corresponding delivery point $i + n$ and vice versa.

$$\sum_{j \in N} x_{i,j}^k - \sum_{j \in N} x_{j,i+n}^k = 0, \forall k \in V, i \in P \tag{6.9}$$

The service starting time of a vehicle at some point j cannot be earlier than the departure time of the vehicle from the previous point i plus the direct travel time between the two points as ensured by the constraint in Equation 6.10.

$$x_{i,j}^k (T_i^k + s_i + t_{i,j} - T_j^k) \leq 0, \forall k \in V, i, j \in A \tag{6.10}$$

The service starting time by some vehicle at point i cannot be earlier than the earliest possible service time or later than the latest possible service time (time window constraint) of point i. This is ensured by the constraint in Equation 6.11

$$a_i \leq T_i^k \leq b_i, \forall k \in V, i \in A \tag{6.11}$$

The precedence constraint among the pickup and delivery points should be maintained. In order for some vehicle to serve a riders' request i, it should visit its pickup point i before its delivery point $i + n$. This is ensured by the constraint in Equation 6.12.

$$T_i^k + s_i + t_{i,i+n} \leq T_{i+n}^k, \forall k \in V, i \in P \tag{6.12}$$

The number of loaded riders at a pickup point should be equal to the number of unloaded riders at its corresponding delivery point. More precisely, if vehicle k has a load L_i^k after servicing point i, it must have a load of $l_j + L_i^k$ after servicing point j as in Equation (6.13).

$$x_{i,j}^k (L_i^k + l_j - L_j^k) = 0, \forall k \in V, i, j \in A \tag{6.13}$$

After servicing some point i by vehicle k, the load of the vehicle should be greater than or equal to the change of load at point i. In addition, the load of the vehicle cannot be greater than its maximum capacity. This is ensured by the constraint in Equation 6.14

$$l_i \leq L_i^k \leq C^k, \forall k \in V, i \in P \tag{6.14}$$

We suppose that each vehicle departs empty (no riders onboard) from its source point and arrives empty to its destination point. This is ensured by the constraint in Equation 6.15. Note that riders and vehicles might share the same geographic sources and destinations, however we consider their sources and destinations to be distinct points.

$$L_k^k = L_{k+v}^k = 0, \forall k \in V \tag{6.15}$$

As already discussed, each driver (vehicle) defines a maximum travel distance and time. The constraint in Equation 6.16 ensures that the maximum vehicle's travel distance is not exceeded and the constraint in Equation 6.17 ensures that the maximum vehicle's travel time (including waiting time) is not exceeded.

$$\sum_{i,j \in A} x_{i,j}^k d_{i,j} \leq MTD_k, \forall k \in V \tag{6.16}$$

$$(T_{k+v}^k - T_k^k) \leq MTT_k, \forall k \in V \tag{6.17}$$

We need also to ensure that the maximum travel time (including waiting time) defined by each rider is not exceeded. This is ensured by the constraint in Equation 6.18.

$$(T_{i+n}^k - s_i - T_i^k) \leq MTT_i, \forall i \in P \tag{6.18}$$

The decision variable $x_{i,j}^k$ is binary and limited to values 0 and 1.

$$x_{i,j}^k \in \{0,1\}, \forall i,j \in A \tag{6.19}$$

Given the multiobjective function and the set of constraints defined, the optimization problem is defined as:

$$\begin{aligned} minimize\ \alpha \sum_{k\in V}\sum_{i,j\in A} x_{i,j}^k d_{i,j} + \beta \sum_{k\in V}(T_{k+v}^k - T_k^k) + \\ \gamma \sum_{k\in V}\sum_{i\in R} max\{0,(T_{i+n}^k - s_i - T_i^k)\} + \\ \delta(n - \frac{1}{2}(\sum_{k\in V}\sum_{i,j\in A} x_{i,j}^k - v)) \end{aligned} \tag{6.5}$$

Subject to:

$$\sum_{k\in V}\sum_{j\in P\cup\{k+v\}} x_{k,j}^k = v \tag{6.6}$$

$$\sum_{k\in V}\sum_{i\in D\cup\{k\}} x_{i,k+v}^k = v \tag{6.7}$$

$$\sum_{k\in V}\sum_{j\in A} x_{i,j}^k \leq 1, \forall i \in N \tag{6.8}$$

$$\sum_{j\in N} x_{i,j}^k - \sum_{j\in N} x_{j,i+n}^k = 0, \forall k \in V, i \in P \tag{6.9}$$

$$x_{i,j}^k (T_i^k + s_i + t_{i,j} - T_j^k) \leq 0, \forall k \in V, i,j \in A \tag{6.10}$$

$$a_i \leq T_i^k \leq b_i, \forall k \in V, i \in A \tag{6.11}$$

$$T_i^k + s_i + t_{i,i+n} \leq T_{i+n}^k, \forall k \in V, i \in P \tag{6.12}$$

$$x_{i,j}^k (L_i^k + l_j - L_j^k) = 0, \forall k \in V, i,j \in A \tag{6.13}$$

$$l_i \leq L_i^k \leq C^k, \forall k \in V, i \in P \tag{6.14}$$

$$L_k^k = L_{k+v}^k = 0, \forall k \in V \tag{6.15}$$

$$\sum_{i,j\in A} x_{i,j}^k d_{i,j} \leq MTD_k, \forall k \in V \tag{6.16}$$

$$(T^k_{k+v} - T^k_k) \leq MTT_k, \forall k \in V \tag{6.17}$$

$$(T^k_{i+n} - s_i - T^k_i) \leq MTT_i, \forall i \in P \tag{6.18}$$

$$x^k_{i,j} \in \{0, 1\}, \forall i, j \in A \tag{6.19}$$

6.2 A Genetic Algorithm for SDMR Ridematching

Given the SDMR ridematching problem with time windows defined, now we discuss the proposed genetic algorithm used to solve it. First we give an overview of the proposed genetic algorithm, and then we describe its different components. The proposed algorithm is a generational genetic algorithm (gGA) as shown in the pseudo code in Algorithm 7. In order not to lose promising solutions during generations in searching for the best solution, elitism [84] is implemented. We implement elitism by reserving two places in the offspring generation for the fittest two solutions in the current generation. In the following we discuss the important parts of the algorithm; the solution representation and initialization and the crossover and mutation operators. The used selection operator is the binary tournament selection.

Algorithm 7: Generational Genetic Algorithm with Elitism

```
begin
1   P ← Initial population
2   Offspring ← φ
3   Evaluate (P)
4   while Stop criteria not met do
5       Offspring ← Offspring ∪ fittest_two_solutions(P)
6       for i ← 1 to |P|/2 − 1 do
7           Parent1 ← Selection_Operator(P)
8           Parent2 ← Selection_Operator(P)
9           Offspring1, Offspring2 ← Crossover_Operator(P)
10          Offspring1 ← Mutation_Operator(Offspring1)
11          Offspring2 ← Mutation_Operator(Offspring2)
12          Evaluate (Offspring1, Offspring2)
13          Offspring ← Offspring ∪ Offspring1 ∪ Offspring2
14      P ← Offspring
15      Offspring ← φ
16  return fittest_solutions(P)
```

6.2.1 Solution Representation

A solution is represented as a schedule of v routes, one route for each vehicle. Each route starts with the source point of the vehicle followed by a list of riders' pickup and delivery points and ends with the destination point of the vehicle. The pickup and delivery points are inserted satisfying the constraints defined in the model. A solution may not match all riders' requests and therefore we keep a list of all non-matched requests. Figure 6.3 shows an example solution representation matching 5 vehicles (A - E) and 12 riders' requests (1 - 12). To speed up the genetic algorithm, we keep for each vehicle a list of matchable rider's requests taking time and distance feasibility into consideration.

A^+	5^+	5^-	4^+	7^+	4^-	7^-	A^-		
B^+	1^+	3^+	3^-	1^-	B^-				
C^+	8^+	2^+	8^-	2^-	C^-				
D^+	6^+	6^-	D^-						
E^+	10^+	10^-	9^+	11^+	9^-	11^-	12^+	12^-	E^-

Figure 6.3: Solution representation. Five Vehicles (A-E) and twelve matched riders' requests (1-12). + - denote the pickup and delivery points of riders' requests and the sources and destinations of the vehicles

6.2.2 Solution Initialization

An initial solution is created as follows. We randomly select a route to initialize among the v routes. The first two points to be inserted in the route are the source and destination points of the vehicle. Initially the two points are scheduled at their earliest time a_i (to take advantage of the maximum possible push forward as below). Then we continuously select a random pair of pickup and delivery points (from a set of feasible points for the current vehicle (route) taking the maximum vehicle's travel time and distance into consideration) and try to insert them in the route. The process repeats for all routes.

During the initialization process, we adopt the Solomon's insertion method in [82]. Given a partially created route $p = (p_1, p_2, ..., p_u)$ for vehicle k, the service starting time $T^k_{p_i}$ at point p_i is:

$$T^k_{p_i} = max\{a_{p_i}, T^k_{p_{i-1}} + s_{p_{i-1}} + t_{p_{i-1},p_i}\}.$$

If the vehicle arrives too early at point p_i, then it should wait before servicing p_i and the waiting time w_{p_i} at p_i is:

$$w_{p_i} = max\{0, a_{p_i} - T^k_{p_{i-1}} - s_{p_{i-1}} - t_{p_{i-1},p_i}\}.$$

Now we discuss the necessary conditions for time feasibility when inserting a new point between the points p_{i-1} and p_i, $1 < i \leq u$. Let $newT^k_{p_i}$ denote the new service time by vehicle k for point p_i after inserting the new point before p_i. Assuming that

the triangular inequality holds both on time and distance between the points, then the insertion of the new point before p_i will result in a time push forward PF_{p_i} in the route at p_i such that:

$$PF_{p_i} = newT^k_{p_i} - T^k_{p_i} \geq 0.$$

The insertion of the new point before p_i will also result in a time push forward $PF_{p_{j+1}}$ at the points after p_i such that:

$$PF_{p_{j+1}} = max\{0, PF_{p_j} - w_{p_{j+1}}\},\ i \leq j < u.$$

After the time push forward, the time window and/or the maximum travel time constraints of some point p_j, $i \leq j \leq u$, might be violated. We sequentially check these two time feasibility constraints until we reach a point p_j with $PF_j = 0$ or its time window or maximum travel time constraint is violated or in the worst case until $j = u$. In addition to time feasibility, we check the driver's maximum travel distance constraint after inserting the new point p. The distance should be less than MTD_k for vehicle k. For the precedence constraint, we always insert the pickup point before its corresponding delivery point and we search for a feasible insertion position for the delivery point in the route to the right of its pickup point. Any insertion that violates the constraints is rejected.

6.2.3 Crossover Operator

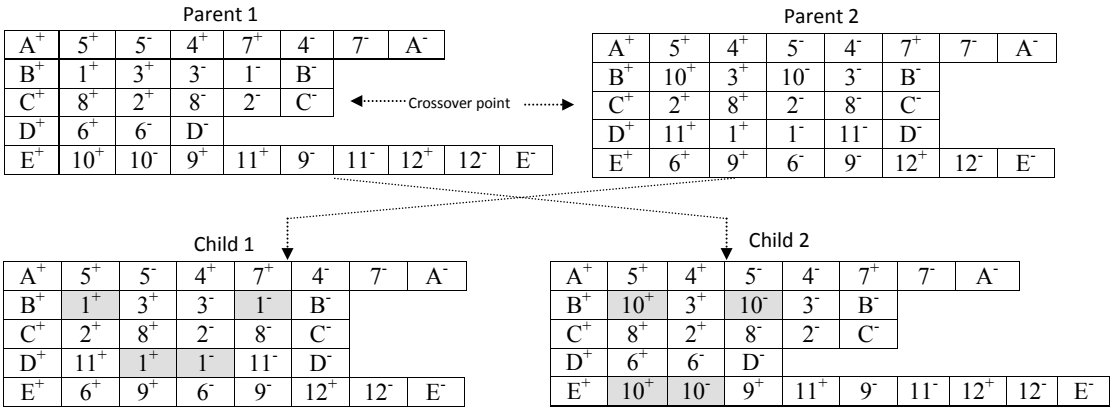


Figure 6.4: Single point crossover operator. Shaded points violate constraints (6.8) and (6.9)

We have defined a single point crossover operator. Given two parent solutions, we randomly select a crossover position (route index) and we make the crossover by exchanging the routes among the two parents starting from the crossover position upwards as in Figure 6.4. Note that the resulting children of the crossover do not violate any of the defined constraints except constraints (6.8) and (6.9) (a riders' request could be

matched twice with different vehicles). In addition to the constraints violation, some pairs of pickup and delivery points that exist in the parents might disappear in the children because of the crossover (e.g. points 10^+ and 10^- in child 1 in Figure 6.4).

After the crossover operation, we perform a repair operation for the children. The repair operation removes all the pickup and delivery points in the routes after the crossover position whenever they exist in any route before the crossover position. Besides the repair operation, we also try to insert the pickup and delivery points of the non-matched requests in the resulting children after each crossover in a process similar to the one during solutions initialization.

6.2.4 Mutation Operators

We use five different mutation operators. A mutation operation that violates any of the constraints is rejected. The first mutation operator is the push backward. In this operator, we try to schedule (service) a randomly selected point p_i, $1 \leq i \leq u = length(p)$, in route p at an earlier time than its current scheduled time $T^k_{p_i}$ with vehicle k including the departure and arrival points of the vehicle. This is done by defining a push backward value PB_{p_i} for p_i as a random percentage of the time difference between the current scheduled time of the selected point and its earliest possible service time:

$$PB_{p_i} = random(T^k_{p_i} - a_{p_i}).$$

After making a push backward at point p_i, we push backward the points after p_i in the route until we reach a point p_j with $PB_{p_j} = 0$ or when $j = u - 1$ such that:

$$PB_{p_{j+1}} = min\{PB_{p_j}, T^k_{p_{j+1}} - a_{p_{j+1}}\},\ i \leq j < u.$$

The second mutation operator is the push forward (very similar to initialization push forward). In this operator, we try to schedule (service) a randomly selected point p_i, $1 \leq i \leq u = length(p)$, in route p at a later time than its current scheduled time $T^k_{p_i}$ with vehicle k including the departure and arrival points of the vehicle. This is done by defining a time push forward value PF_{p_i} for p_i as a random percentage of the time difference between the latest possible service time of the selected point and its current scheduled time:

$$PF_{p_i} = random(b_{p_i} - T^k_{p_i}).$$

We also push forward all the points after p_i in the route using the same process during solution initialization: $PF_{p_{j+1}} = max\{0, PF_{p_j} - w_{p_{j+1}}\},\ i \leq j < u$. We stop the push forward operation when we reach a point p_j with $PF_{p_j} = 0$ or when $j = u - 1$.

The push backward and push forward mutation operators could advance or delay the departure of the vehicles which could open new chances for new matches. For example, a request with an earliest departure time later than the current departure time of the vehicle could not be served because it requires the vehicle to wait long time. If the departure of the vehicle is delayed, then it could be feasible to serve the request. In

addition, the push backward and push forward mutation operators could help in avoiding the waiting at some pickup points which could reduce the total drivers and riders travel time. They could also help in aggregating the waiting times at different points in one waiting block which can be utilized to serve new requests.

The third mutation operator is the remove-insert operator. We randomly select a pair of pickup and delivery points to delete from route p (and mark their request as non-matched). After the delete operation, we perform a push backward operation in the positions of the deleted points. By the end of the delete operations, we try to insert as much pairs of pickup and delivery points as possible from the points of the non-matched requests. This mutation operator helps in exploring different matching alternatives. A matched request could require a long travel distance or time and prevents a large number of requests matching.

The fourth mutation operator is the transfer mutation. In this operator, we randomly select a pair of pickup and delivery points from route p and try to insert them in another route. This mutation operator helps in finding the best driver to serve a request. Serving a request with some vehicle could require long travel time, long waiting time or long travel distance where the request could be better served by another vehicle.

The last mutation operator is the swap mutation operator. In this operator, we swap a randomly selected point p_i and its neighbor point p_{i+1} in route p as long as they do not belong to the same request. Figure 6.5 explains the idea of swap mutation and gives an example improvement of such operator. This operator could reduce the travel distance or eliminate some waiting time. The proposed swap mutation is a type of the city swap mutation (exchange mutation) [87]. However, the swap is limited to only neighbor points to avoid breaking the precedence constraint (scheduling a delivery point before its pickup point).

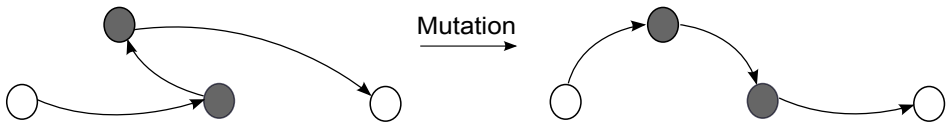


Figure 6.5: Swap mutation operator. Change the visit order of the gray points

We noticed that the push forward and push backward mutation operators are the most important among the mutation operators in the sense that they play with the scheduled time of the points in the different routes, and their work is not compensated by the crossover operator as compared with the other mutation operators. However, still the other mutation operators participate in the quality of the solution.

6.2.5 Experimentation

Following we describe the utilized real world problem instances that are considered in this work and the settings of the genetic algorithm and problem instances. Then we discuss the experimentation results.

Problem Instances

As drivers are not required to provide an exact travel path for this variant of ridematching and because it is sufficient that they provide their source and destination points, we can utilize real world problem instances in testing the proposed genetic algorithm. Hence, to test the behavior of the proposed algorithm for solving the SDMR ridematching problem with time windows, we have utilized a travel and activity survey for northeastern Illinois conducted by Chicago Metropolitan Agency for Planning CMAP [2]. Data collection took place between January 2007 and February 2008 and covered a total of 10,552 households participant. A trip in the study is defined as a pair of source and destination positions (using latitude and longitude) with departure and arrival timestamps.

Our problem instances are mainly based on two travel patterns. The first consists of a total of 698 trips (RM698) among them 469 trips with participants travel with their own vehicles and the rest rely on other forms of transportation. The second consists of a total of 744 trips (RM744) among them 533 trips with participants travel with their own vehicles and the rest rely on other forms of transportation. We select from the trips with participants traveling with their own vehicles what constitutes 36% of the total trips in each pattern to be as drivers and the rest as riders. The drivers are either selected randomly or those with the maximum travel distance as indicated by DMD in Table 6.2. A time window is engineered for each trip. We consider a vehicle speed of 60km/h. The distance between two points is measured using the Haversin formula [80] and the travel time between them is the ceil of the computed travel time (always integer). Table 6.2 shows the characteristics of the eight problem instances used in this study.

Settings of the Problem Instances and the Genetic Algorithm

The parameters' settings (defined in Sec. 6.1) of the problem instances are $AT_i = 0$ and $BT_i = 1.3$ for all riders' requests and vehicles (drivers' offers), $C^k = 5$, $AD_i = 0$ and $BD_i = 1.3$ for all vehicles and $s_i = 0$ and $dm_i = 1$ for all riders' requests. The objectives weights are: $\alpha = 0.7$ and $\beta = \gamma = \delta = 0.1$.

The parameters settings of the genetic algorithm are: population size = 100, number of generations = 100, crossover probability = 1.0 and mutation probability =0.4. The selection of the probabilities of the crossover and mutation is based on a study result summarized in Table 6.3. The algorithm is executed 30 independent times for each problem instance.

Results Discussion

First we provide the results of a single run of the genetic algorithm on the problem instances and we visualize the value of the objective function at different generations. We select one instance, RM698_L60, and visualize the values of the components of the objective function at different generations. Then we provide the statistical results of the 30 independent runs of the algorithm on the problem instances. To avoid scaling

[2] http://www.cmap.illinois.gov/travel-tracker-survey.

Table 6.2: Problem instances. DMD = divers with maximum distance, TW= time window, DTD and DTT are the total direct travel distance and time respectively of the vehicles (drivers) without ridesharing.

Instance	No. drivers	No. riders	DMD	Earliest departure	Latest departure	TW ± min	DTD (km)	DTT (min)
RM744_R15	268	476	no	10:00	16:40	15	2621.8	2737
RM744_L15	268	476	yes	10:00	16:40	15	4468.2	4582
RM744_R60	268	476	No	10:00	16:40	60	2621.8	2737
RM744_L60	268	476	yes	10:00	16:40	60	4468.2	4582
RM698_R15	250	448	No	00:30	23:30	15	2451.9	2565
RM698_L15	250	448	yes	00:30	23:30	15	4602.6	4717
RM698_R60	250	448	No	00:30	23:30	60	2451.9	2565
RM698_L60	250	448	yes	00:30	23:30	60	4602.6	4717

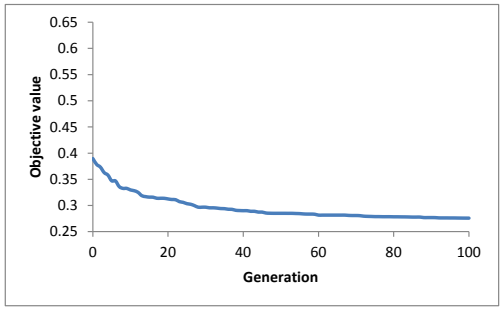
Table 6.3: Pairwise Wilcoxon rank sum test comparing different configurations G_i with 30 independent runs for each configuration. ▲ = row element is better than column element regarding the value of the objective function and the opposite for ▽.

crossover	mutation		G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9	G_{10}	G_{11}
1.0	1.0	G_0	–	–	▲	▲	▲	▲	–	▲	▲	▲	▲
1.0	0.7	G_1		–	▲	▲	▲	▲	▲	▲	▲	▲	▲
1.0	0.4	G_2			▲	▲	▲	▲	▲	▲	▲	▲	▲
1.0	0.1	G_3				▽	▽	–	▽	▽	▲	▽	–
0.7	1.0	G_4					▲	▲	–	–	▲	–	▲
0.4	1.0	G_5						▲	▽	▽	▲	▽	▲
0.1	1.0	G_6							▽	▽	▲	▽	–
0.7	0.7	G_7								–	▲	–	▲
0.7	0.3	G_8									▲	–	▲
0.7	0.1	G_9										▽	▽
0.4	0.7	G_{10}											▲
0.1	0.7	G_{11}											

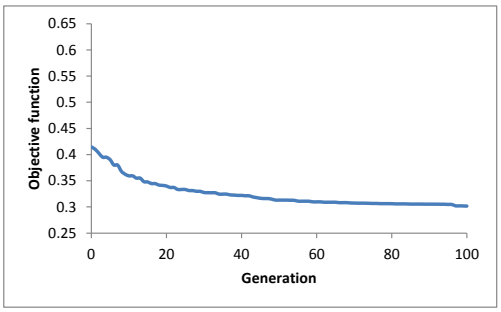
problems, the values of the different criteria are normalized before computing the value of the objective function. Note that the direct travel time and distance and the maximum allowed travel time and distance can provide us with upper and lower bounds for the total travel time and distance. The number of the riders' requests gives us an upper bound for the number of matched riders' requests.

Figure 6.6 shows the optimization of the objective function by the genetic algorithm for the eight problem instances. It shows the value of the objective function at different generations (fitness of the best individual) for each instance. We see that the proposed algorithm is able to minimize the multiobjective function. The most optimization is for instances with $DMD = yes$ and with time windows ± 60 as in Figures 6.6(a) and 6.6(b). Figure 6.6(h) shows the least optimization for the instance RM698_R15 with time window ± 15 and $DMD = no$ where the algorithm converges after near about 20 generations. The vehicles in the instances with $DMD = yes$ travel for longer distances and have the possibility to serve larger number of riders' requests which opens a larger room for optimization. The drivers and riders in the instances with time windows ± 60 have wide range of possible departure and arrival times which also opens a larger room for optimization. Following, we look at each component of the multiobjective function through generations for the instance RM698_L60 in Figure 6.7.

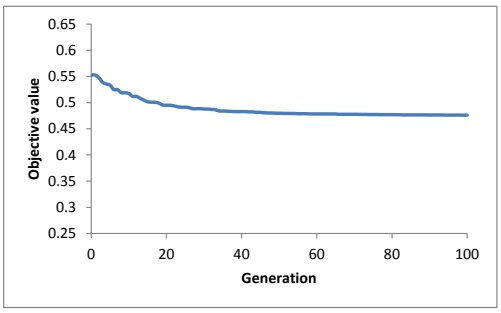
Figure 6.7(a) shows the number of matched (served) riders' requests at different generations. We notice that the general trend is that the number of matched riders' requests increases with generations as this objective has the highest weight. Figure 6.7(b) shows the total distance of the vehicles' trips at different generations. We notice a sawteeth behavior in the curve of this objective which is explained as follows. By the increase of the generations, new riders' requests are matched resulting in increasing the total distance of vehicles' trips. The algorithm then minimizes the distance of the vehicles' trips that is required to serve the matched riders' requests which opens a room for matching new riders' requests. Matching new riders' requests increases the distance of the vehicles' trips again.



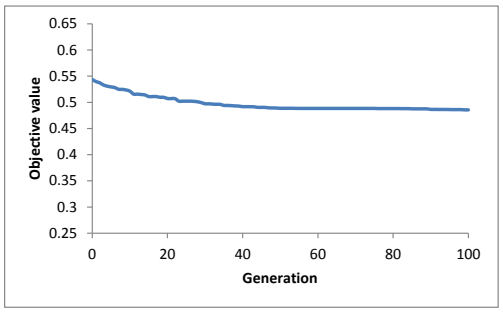
(a) RM744_L60



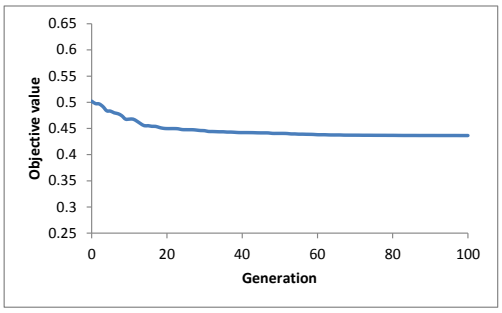
(b) RM698_L60



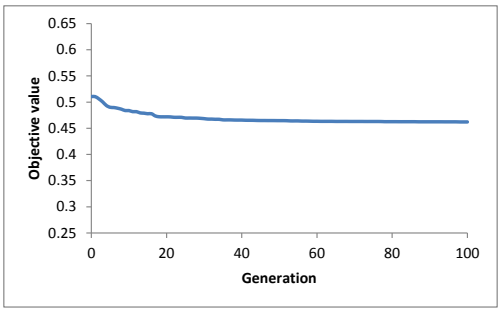
(c) RM744_R60



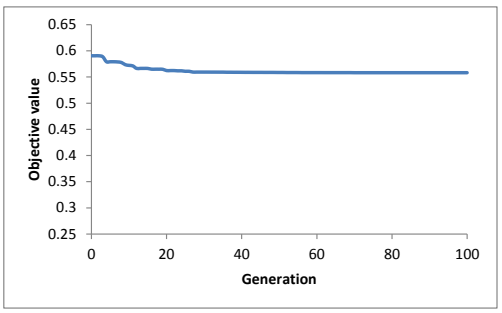
(d) RM698_R60



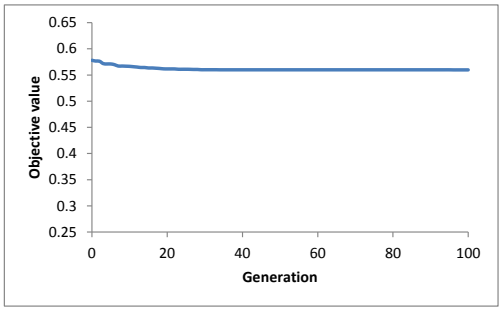
(e) RM744_L15



(f) RM698_L15

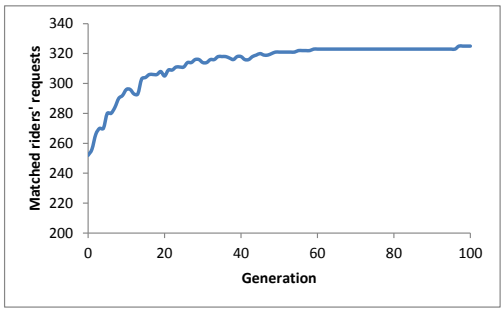


(g) RM744_R15

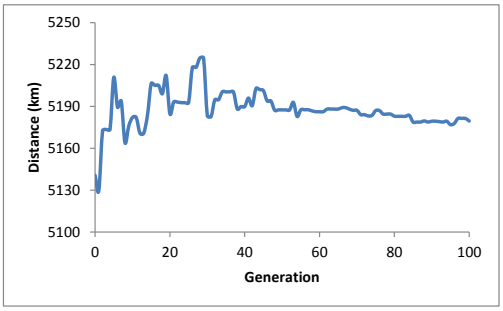


(h) RM698_R15

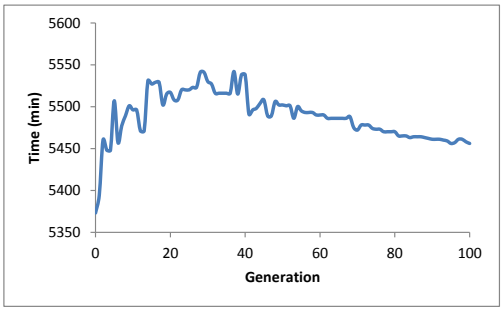
Figure 6.6: The best fitness at different generations of the genetic algorithm



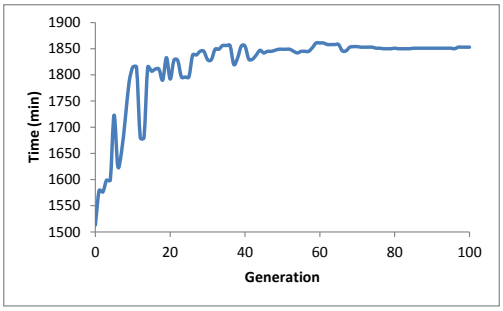
(a) The number of matched riders' requests



(b) The total distance of the vehicles' trips



(c) The total time of the vehicles' trips



(d) The total time of the trips of the matched riders' requests

Figure 6.7: The best values of the components of the objective function at different generations of the genetic algorithm

Figure 6.7(c) shows the total time of the vehicles' trips. This curve is very similar to the curve of the total distance of the vehicles' trips and the sawteeth behavior is explained in the same way. Optimizing the time of the vehicles' trips opens a new room for matching new riders' requests which increases the time of the vehicles' trips again. The total time of a vehicle's trip depends on the distance of the trip, service time of the riders' requests in the trip (at pickup and delivery points) and potential waiting time at some of the pickup points. While the service time is constant, the waiting time changes for different configurations. This means that it is not necessary that the time of drivers' trips always increases/decreases with the increase/decrease of their distance. We might increase the distance of some route and reduce the waiting time at some of its points resulting in smaller total travel time. We notice also from Figures 6.7(a) - 6.7(d) that after near about 40 generations, the algorithm mostly optimizes the objectives other than the number of matched riders' requests as it becomes harder to find new matches.

Figure 6.7(d) shows the total time of the trips of the matched riders' requests. The value of this objective depends on the number of matched riders' requests, travel distance to serve these requests and the waiting time at particular pickup points of the matched requests. Note that for this objective, the vehicle's waiting time at particular point is multiplied by the number of riders in the vehicle.

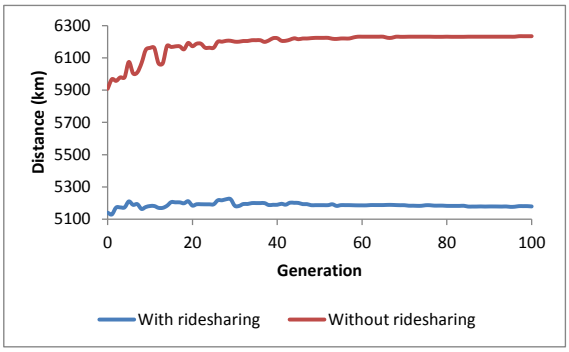


Figure 6.8: The total distance of the trips of the vehicles and the matched riders' requests with and without ridesharing for the instance RM698_L60

Figure 6.8 shows the total distance of the trips with and without ridesharing. The curve with ridesharing is the same as the curve of the total distance of the vehicles' trips in Figure 6.7(b). The curve without ridesharing shows the total distance of the trips of the vehicles and the matched riders' requests as if each one travels alone. The figure shows that the ridesharing could save a significant amount of travel distance. We notice also that the distance saving generally increases with generations (till generation ≈ 40 in the Figure) as new riders' requests are matched and trips' distances are minimized.

Table 6.4 shows the mean and standard deviation of the results of the 30 independent runs on the problem instances. The results include for each instance the value of the objective function and each of its components in addition to the runtime of the genetic algorithm. Figure 6.9 visualizes the average number of matched riders' requests for the different instances. The figure shows that for each trips' pattern (RM698 and RM744), the instances with $DMD = yes$ have higher number of matched riders' requests than instances with $DMD = no$. In the instances with $DMD = yes$ (RM698_L15, RM698_L60, RM744_L15, RM744_L60), vehicles travel for longer distances and are supposed to serve larger number of riders' requests. In addition, for a given instance ($RM698\_L$,$RM698\_R$, $RM744\_L$, $RM744\_R$), the number of matched riders' requests is higher when $TW = 60$ as compared with $TW = 15$. Wider time windows (TW) increases the chance of matching riders' requests.

From tables 6.4 and 6.2 we can get the percentage of the total vehicles' travel distance (with serving riders) relative to the total vehicles' distance that would have been traveled without serving riders. This is done by dividing the entry of the vehicles' distance in table 6.4 on the DTD entry of the corresponding instance in table 6.2. The same applies for the travel time. For example, for the instance RM698_L60, the algorithm was able to match 326 riders' requests in average (73% of the riders' requests) with 1.12% of the total vehicles' direct travel distance and 1.15% of the total vehicles' direct travel time. The algorithm was able to match 187 riders' requests in average (42% of the riders' requests) for the instance RM698_R60 with 1.09% of the total vehicles' direct travel distance and 1.11% of the total vehicles' direct travel time.

Table 6.4: Summary of the results of the genetic algorithm on the different instances. Mean and standard deviation μ_σ. OF= objective function, No. MRR= Number of matched riders' requests, GRT= genetic alg. runtime.

Instance	**OF**	**No. MRR**	**Vehicles' distance**	**Vehicles' time**	**Riders' time**	**GRT (sec)**
RM744_R15	$0.558_{\sigma=0.0010}$	$138.7_{\sigma=0.69}$	$2784.907_{\sigma=2.259}$	$2953.8_{\sigma=2.701}$	$952.967_{\sigma=6.374}$	$21.509_{\sigma=0.431}$
RM744_L15	$0.433_{\sigma=0.0030}$	$234.9_{\sigma=2.856}$	$4872.177_{\sigma=15.818}$	$5104.2_{\sigma=16.138}$	$1298.133_{\sigma=17.043}$	$43.301_{\sigma=1.628}$
RM744_R60	$0.479_{\sigma=0.0040}$	$207.933_{\sigma=2.502}$	$2863.787_{\sigma=11.109}$	$3062.367_{\sigma=10.88}$	$1270.1_{\sigma=24.701}$	$96.662_{\sigma=4.273}$
RM744_L60	$0.271_{\sigma=0.0050}$	$363.933_{\sigma=3.316}$	$5010.591_{\sigma=13.59}$	$5316.367_{\sigma=14.538}$	$1802.667_{\sigma=33.336}$	$110.767_{\sigma=8.226}$
RM698_R15	$0.558_{\sigma=0.0010}$	$124.4_{\sigma=1.083}$	$2605.949_{\sigma=5.883}$	$2750.433_{\sigma=7.149}$	$685.467_{\sigma=8.913}$	$13.079_{\sigma=0.943}$
RM698_L15	$0.461_{\sigma=0.0030}$	$198.267_{\sigma=1.459}$	$4983.294_{\sigma=11.52}$	$5195.0_{\sigma=12.565}$	$1264.9_{\sigma=10.888}$	$21.718_{\sigma=1.657}$
RM698_R60	$0.482_{\sigma=0.0030}$	$187.867_{\sigma=2.68}$	$2679.89_{\sigma=13.82}$	$2853.967_{\sigma=15.096}$	$1014.733_{\sigma=41.151}$	$109.378_{\sigma=10.636}$
RM698_L60	$0.295_{\sigma=0.0070}$	$326.067_{\sigma=3.855}$	$5158.488_{\sigma=21.662}$	$5438.733_{\sigma=22.044}$	$1798.033_{\sigma=38.815}$	$68.063_{\sigma=3.992}$

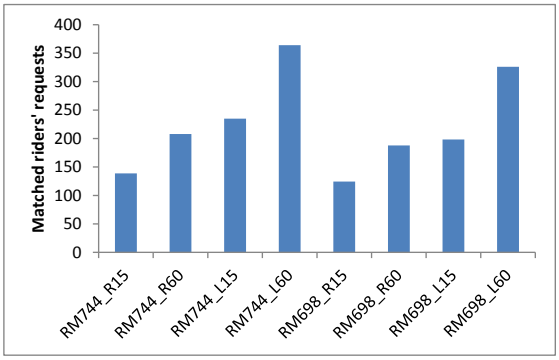


Figure 6.9: The average number of matched riders' requests for the different instances.

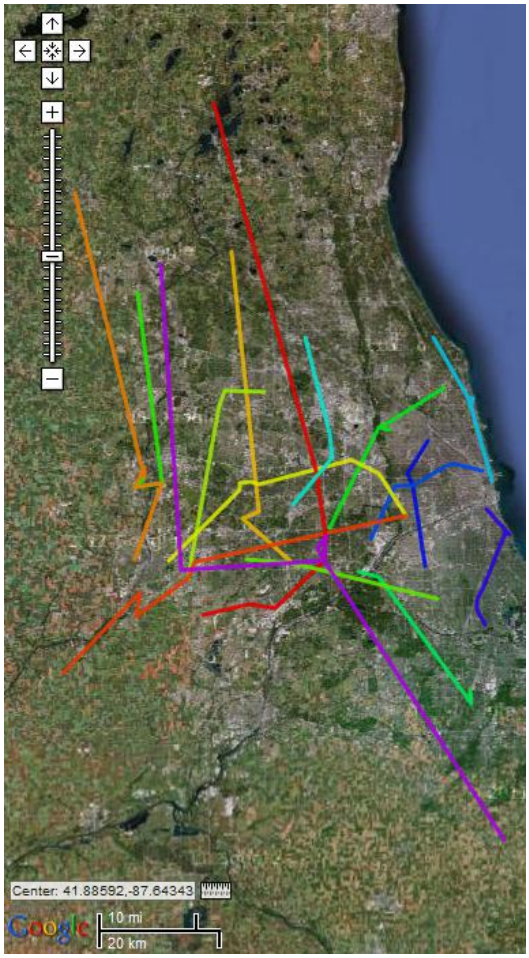


Figure 6.10: Visualization of selected vehicles' trips. The different lines represent the trips of the different drivers from their sources to their destinations including visiting the pickup and delivery points of the served riders. The used GPX visualizer is: http://www.gpsvisualizer.com.

To see how the trips of the vehicles look like, we have created a GPX[3] file for some of the vehicles' trips and used a GPX visualizer to visualize it on Google Maps. The visualization result is provided in Figure 6.10.

6.3 A Genetic Insertion Heuristic Algorithm for SDMR Ridematching

In a typical ridesharing scenario, not all riders' requests and drivers' offers are known in advance before the execution of the genetic algorithm. New requests and offers keep arriving and need to be matched. Matching the newly arriving requests/offers requires the modification of the result of the genetic algorithm.

We propose an algorithm that switches between the already proposed genetic algorithm and an insertion heuristic algorithm to match the newly arriving requests/offers. In the first stage, the algorithm works as a genetic algorithm (previously described) while in the second stage it works as an insertion heuristic that modifies the solution of the genetic algorithm to do ridematching in real-time. We divide the day into a set of time periods. Each time period starts with executing the genetic algorithm to solve a static version of the problem given all known requests/offers. After the execution of the genetic algorithm, we utilize an insertion heuristic very similar to the one proposed by Jaw et al. [52] until the end of the time period. The insertion heuristic gives an online answer for each newly received request/offer by updating the solution produced by the genetic algorithm. All non-matched requests/offers by the genetic algorithm and the insertion heuristic are stored for future matching and to be as an input for the genetic algorithm in the next time period.

The genetic algorithm as a metaheuristic is supposed to produce better quality matches and to better optimize the objective function as compared to the insertion heuristic that is subject to local optima. However, the insertion heuristic is more suitable for the ridematching problem in dynamic ridesharing as it can give answers in real-time. The shorter the time period, the more often the genetic algorithm is executed resulting in better results on the cost of longer execution time. On the other hand, if the time period is long, then we put more pressure on the insertion heuristic resulting in a more responsive algorithm with probably lower quality solutions.

6.3.1 The Insertion Heuristic

Given a solution from the genetic algorithm, the insertion heuristic answers each newly received offer or request by modifying the solution when possible. If a driver offer is received, then a route is added to the solution and the heuristic tries to match as much as possible from the non-matched riders' requests. If a rider's request is received, then the heuristic tries to insert it in one of the routes. Finding a match for a given request involves finding all feasible insertions for the pickup and delivery points of the request in all routes and selecting the best insertion. To find a feasible insertion of a rider's pickup

[3]GPS Exchange Format

and delivery points, the same insertion method that is used in solution initialization is used here. The best insertion is defined as the one that adds the minimum value to the heuristic objective function. The heuristic objective function is the weighted sum of the total distance and time of vehicles' trips and the total time of the trips of the matched riders' requests. We do not consider the number of matched riders in the heuristic's objective function because each insertion possibility adds one rider.

By the time of receiving a new request x, parts of the solution might have already been executed (points already visited). Therefore we apply an additional constraint when finding a feasible insertion position for the pickup point of the new request. The constraint states that the scheduled time of the point just before the candidate insertion position i of the new pickup point in the route p of vehicle k should be greater than the arrival time inT_x of the new request x:

$$T^k_{p_{i-1}} > inT_x.$$

6.3.2 Experimentation

The same instances in Table 6.2 are used in this study. For each one of the eight instances, we study the behavior of the algorithm in case of the percent of known riders' requests (KRR) by the time of executing the genetic algorithm is 20%,60% and 100%. The insertion heuristic is then utilized to match the $\overline{KRR}$ remaining requests. We consider that all drivers' offers are known in advance by the time of executing the genetic algorithm. The settings of the problem instances and the genetic algorithm are all the same as in Section 6.2.5. The algorithm is executed 30 independent times for each problem instance and percentage of KRR. The experimentation platform is the same platform described in Section 3.5.

Table 6.5 shows the results of the 30 independent runs of the algorithm for each instance and percentage of KRR (mean and standard deviation). The results include the value of the objective function and each of its components, the runtime of the genetic algorithm and the runtime per request of the insertion heuristic. The values of the different objectives are normalized before computing the value of the objective function.

We make use of figures to further discuss our findings. Figure 6.11 shows for each instance the ratio of the average value of the objective function under percentages 20% and 60% of the KRR to its average value under percentage 100%. In other words, it shows how worse the value of the objective function if we rely on the insertion heuristic than if we totally rely on the genetic algorithm. We see that by decreasing the percentage of the known riders' requests at the time of executing the genetic algorithm, we get worse final values of the objective function (larger ratio). Also we notice that the ratio is larger for instances with $DMD = yes$. This is because for instances with $DMD = yes$, the vehicles travel for longer distances and are supposed to serve larger number of riders requests. Therefore, there is larger room for optimization considering the ordering and timing of the riders' points where the insertion heuristic gets stuck at local optima and the genetic algorithm performs better.

Table 6.5: Summary of the results of the genetic insetion heuristic algorithm. Mean and standard deviation μ_σ. %KRR= percentage of the known riders' requests, OF= objective function, No. MRR= Number of matched riders' requests, GRT= genetic alg. runtime, HRT= heuristic runtime per request.

Instance	%KRR	OF	No. MRR	vehicles' distance	vehicles' time	riders' time	GRT (sec)	HRT (ms)
RM744_R15	20	$0.605_{\sigma=0.0050}$	$105.333_{\sigma=4.671}$	$2786.127_{\sigma=8.524}$	$2932.833_{\sigma=10.453}$	$834.233_{\sigma=38.518}$	$3.15_{\sigma=0.17}$	$0.116_{\sigma=0.024}$
	60	$0.585_{\sigma=0.0030}$	$119.367_{\sigma=2.168}$	$2786.254_{\sigma=4.507}$	$2947.667_{\sigma=4.643}$	$901.267_{\sigma=9.889}$	$9.443_{\sigma=0.251}$	$0.137_{\sigma=0.039}$
	100	$0.558_{\sigma=0.0010}$	$138.7_{\sigma=0.69}$	$2784.907_{\sigma=2.259}$	$2953.8_{\sigma=2.701}$	$952.967_{\sigma=6.374}$	$21.509_{\sigma=0.431}$	$0.0_{\sigma=0.0}$
RM744_L15	20	$0.585_{\sigma=0.0020}$	$115.8_{\sigma=2.072}$	$4736.434_{\sigma=6.795}$	$4902.6_{\sigma=8.048}$	$1026.8_{\sigma=6.6}$	$4.188_{\sigma=0.073}$	$0.127_{\sigma=0.012}$
	60	$0.483_{\sigma=0.0050}$	$199.2_{\sigma=4.261}$	$4849.71_{\sigma=15.703}$	$5072.8_{\sigma=15.01}$	$1194.167_{\sigma=12.662}$	$17.004_{\sigma=0.705}$	$0.201_{\sigma=0.041}$
	100	$0.433_{\sigma=0.0030}$	$234.9_{\sigma=2.856}$	$4872.177_{\sigma=15.818}$	$5104.2_{\sigma=16.138}$	$1298.133_{\sigma=17.043}$	$43.301_{\sigma=1.628}$	$0.0_{\sigma=0.0}$
RM744_R60	20	$0.576_{\sigma=0.0050}$	$134.567_{\sigma=4.295}$	$2820.904_{\sigma=9.317}$	$2985.433_{\sigma=11.635}$	$1019.567_{\sigma=24.415}$	$5.892_{\sigma=0.472}$	$0.352_{\sigma=0.023}$
	60	$0.515_{\sigma=0.0040}$	$182.667_{\sigma=2.675}$	$2861.524_{\sigma=8.328}$	$3050.867_{\sigma=8.755}$	$1203.467_{\sigma=23.581}$	$38.229_{\sigma=1.82}$	$0.472_{\sigma=0.048}$
	100	$0.479_{\sigma=0.0040}$	$207.933_{\sigma=2.502}$	$2863.787_{\sigma=11.109}$	$3062.367_{\sigma=10.88}$	$1270.1_{\sigma=24.701}$	$96.662_{\sigma=4.273}$	$0.0_{\sigma=0.0}$
RM744_L60	20	$0.484_{\sigma=0.0070}$	$197.2_{\sigma=6.172}$	$4794.537_{\sigma=19.037}$	$4997.067_{\sigma=21.695}$	$1384.2_{\sigma=27.165}$	$5.641_{\sigma=0.321}$	$0.45_{\sigma=0.029}$
	60	$0.321_{\sigma=0.0060}$	$326.5_{\sigma=4.395}$	$4990.478_{\sigma=13.664}$	$5278.2_{\sigma=13.531}$	$1660.9_{\sigma=27.795}$	$36.614_{\sigma=2.223}$	$0.726_{\sigma=0.043}$
	100	$0.271_{\sigma=0.0050}$	$363.933_{\sigma=3.316}$	$5010.591_{\sigma=13.59}$	$5316.367_{\sigma=14.538}$	$1802.667_{\sigma=33.336}$	$110.767_{\sigma=8.226}$	$0.0_{\sigma=0.0}$
RM698_R15	20	$0.575_{\sigma=0.0030}$	$96.8_{\sigma=2.821}$	$2543.046_{\sigma=7.241}$	$2670.867_{\sigma=8.838}$	$557.8_{\sigma=10.028}$	$2.84_{\sigma=0.023}$	$0.081_{\sigma=0.01}$
	60	$0.573_{\sigma=0.0030}$	$110.533_{\sigma=3.096}$	$2593.739_{\sigma=8.929}$	$2730.133_{\sigma=9.976}$	$671.8_{\sigma=13.088}$	$6.072_{\sigma=0.156}$	$0.095_{\sigma=0.034}$
	100	$0.558_{\sigma=0.0010}$	$124.4_{\sigma=1.083}$	$2605.949_{\sigma=5.883}$	$2750.433_{\sigma=7.149}$	$685.467_{\sigma=8.913}$	$13.079_{\sigma=0.943}$	$0.0_{\sigma=0.0}$
RM698_L15	20	$0.584_{\sigma=0.0080}$	$106.6_{\sigma=6.859}$	$4796.58_{\sigma=21.38}$	$4947.433_{\sigma=26.087}$	$1050.233_{\sigma=17.701}$	$3.323_{\sigma=0.057}$	$0.092_{\sigma=0.018}$
	60	$0.495_{\sigma=0.0030}$	$174.3_{\sigma=1.754}$	$4973.511_{\sigma=12.49}$	$5170.5_{\sigma=13.455}$	$1198.4_{\sigma=13.827}$	$8.795_{\sigma=0.361}$	$0.147_{\sigma=0.018}$
	100	$0.461_{\sigma=0.0030}$	$198.267_{\sigma=1.459}$	$4983.294_{\sigma=11.52}$	$5195.0_{\sigma=12.565}$	$1264.9_{\sigma=10.888}$	$21.718_{\sigma=1.657}$	$0.0_{\sigma=0.0}$
RM698_R60	20	$0.551_{\sigma=0.0060}$	$135.633_{\sigma=4.963}$	$2651.593_{\sigma=14.225}$	$2799.333_{\sigma=17.034}$	$786.4_{\sigma=46.253}$	$5.657_{\sigma=0.23}$	$0.593_{\sigma=0.021}$
	60	$0.505_{\sigma=0.0030}$	$170.6_{\sigma=2.715}$	$2677.996_{\sigma=8.112}$	$2841.667_{\sigma=8.42}$	$1054.267_{\sigma=37.865}$	$32.277_{\sigma=2.622}$	$0.673_{\sigma=0.048}$
	100	$0.482_{\sigma=0.0030}$	$187.867_{\sigma=2.68}$	$2679.89_{\sigma=13.82}$	$2853.967_{\sigma=15.096}$	$1014.733_{\sigma=41.151}$	$109.378_{\sigma=10.636}$	$0.0_{\sigma=0.0}$
RM698_L60	20	$0.472_{\sigma=0.0090}$	$194.067_{\sigma=6.011}$	$4937.461_{\sigma=15.07}$	$5130.933_{\sigma=17.16}$	$1448.0_{\sigma=36.303}$	$4.824_{\sigma=0.13}$	$0.358_{\sigma=0.017}$
	60	$0.347_{\sigma=0.0060}$	$291.667_{\sigma=3.037}$	$5140.778_{\sigma=14.842}$	$5398.133_{\sigma=17.627}$	$1681.267_{\sigma=27.566}$	$18.235_{\sigma=0.983}$	$0.385_{\sigma=0.04}$
	100	$0.295_{\sigma=0.0070}$	$326.067_{\sigma=3.855}$	$5158.488_{\sigma=21.662}$	$5438.733_{\sigma=22.044}$	$1798.033_{\sigma=38.815}$	$68.063_{\sigma=3.992}$	$0.0_{\sigma=0.0}$

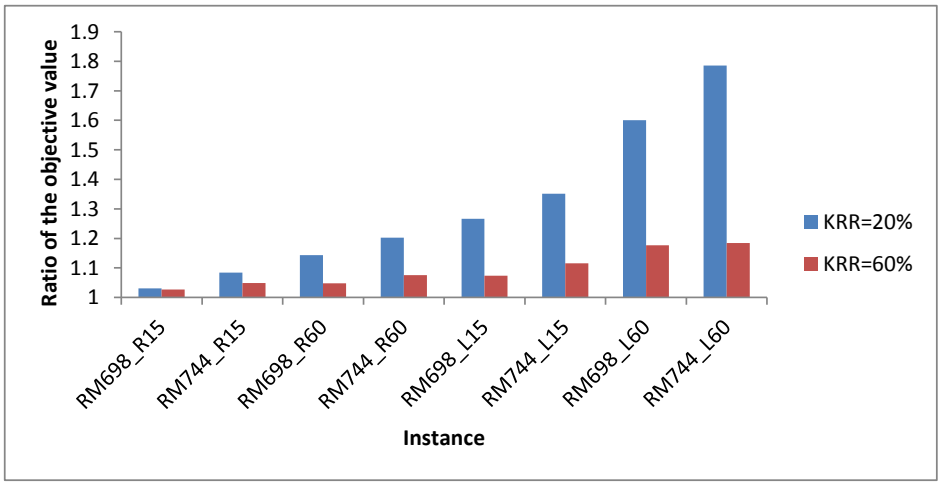
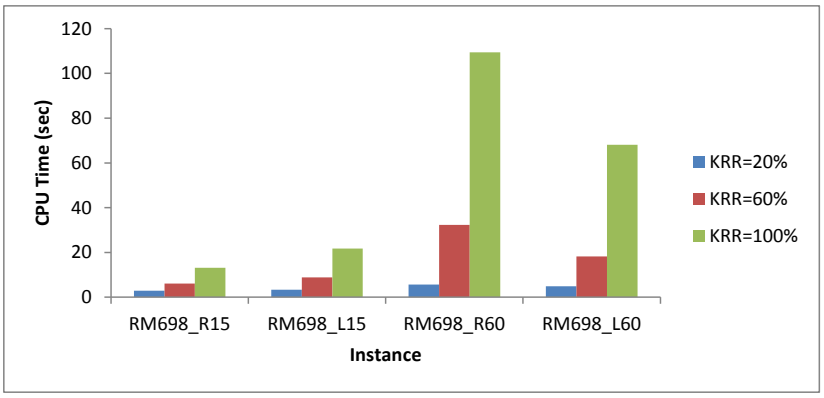


Figure 6.11: The ratio of the average value of the objective function (OF) under the percentages 20% and 60% of the KRR (Known Riders' Requests) to its average value under percentage 100%.

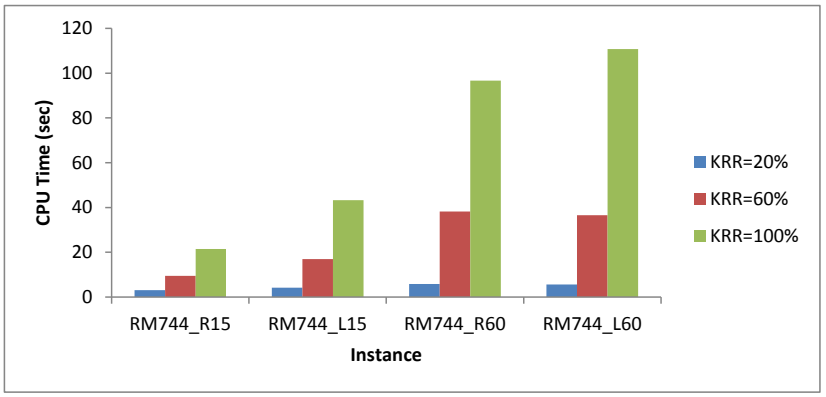
Within each group of instances with $DMD = yes$ or $DMD = no$ (e.g. RM744_L15, RM744_L60, RM698_L15,RM698_L60), the instances with time windows of $\pm$ 60 minutes showed larger ratio than the instances with time windows of $\pm$ 15 minutes. Increasing the time windows increases the search space by increasing the matchable riders' requests list of each vehicle. Also we notice that for each two instances with the same DMD and the same time window (e.g. RM744_L60,RM698_L60), we notice that the ratio is larger for the instances RM744. This is because for the instances RM744 we have more trips and the trips' departures are closer to each other as compared to instances RM698. This means that there is larger search space and riders' requests can be served with larger number of vehicles.

Figure 6.12(a) shows the average runtime of the genetic algorithm for the instances RM698 under different percentages of KRR. Obviously the runtime of the genetic algorithm increases with the increase of the percentage of KRR at the time of its execution. Also we notice that the length of the time window plays a major role in the runtime of the genetic algorithm. Wider time windows increase the size of the list of the matchable riders' requests of each vehicle which is used often after each crossover and remove-insert mutation to try to match additional non-matched requests. This leads to increase the runtime of the genetic algorithm. Figure 6.12(b) shows a similar runtime pattern of the genetic algorithm on the instances RM744.

Figure 6.13 shows the average runtime of the insertion heuristic per rider's request on the instances RM698 and RM744 under different percentages of KRR. The runtime increases for instances with higher percentages of KRR as the solution of the genetic algorithm contains routes with more served requests. This results in larger number of possible insertions which needs to be considered by the heuristic.



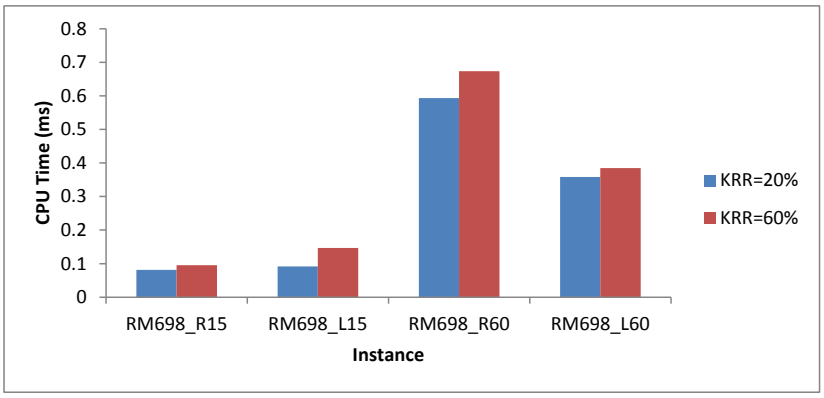
(a) Instances RM698



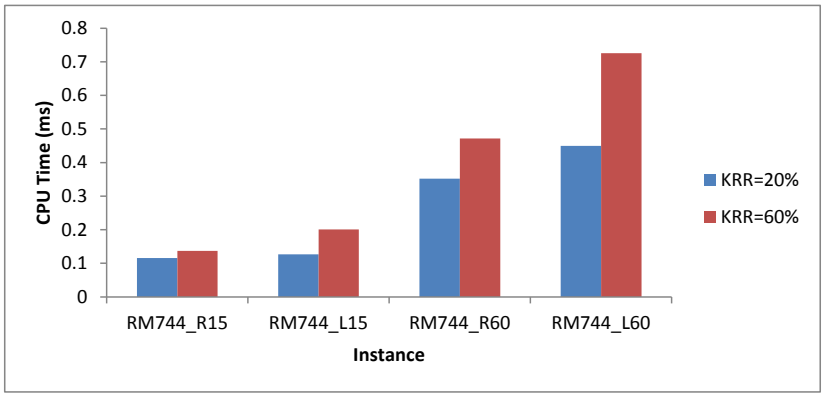
(b) Instances RM744

Figure 6.12: The average runtime of the genetic algorithm for the different instances under different percentages of KRR.

It takes the insertion heuristic fractions of millisecond to answer a request on all problem instances and percentages of KRR which is a real-time answer and in the worst case it takes the genetic algorithm 110.767 seconds. The runtime of the genetic algorithm is justifiable as we run it only once per time period and we rely on the insertion heuristic for the rest of the period. The results in Table 6.5 and Figure 6.11 indicate that the genetic algorithm helps to better optimize the objective function.



(a) Instances RM698



(b) Instances RM744

Figure 6.13: The average runtime of the insertion heuristic per request for the different instances under different percentages of KRR.

6.4 Summary

In this chapter we have considered the single driver and multiple riders ridematching problem with time windows. We have proposed a genetic algorithm to solve the considered problem and applied it on ridematching problem instances built on real world trip traces. The proposed algorithm was able to match a reasonable number of riders' requests. In addition, we have proposed an algorithm that alternates between the proposed genetic algorithm and an insertion heuristic to solve the SDMR ridematching problem in more realistic scenarios and provides a balance between the solution's quality and the algorithm responsiveness.

In this chapter, a rider is limited to be matched with at most one driver. While this might make the problem easier to solve, it might reduce the number of matched riders. In the next chapter we relax this constraint and consider the variant multiple drivers and multiple riders ridematching.

Chapter 7

Multiple Drivers and Multiple Riders Ridematching

In this chapter we consider the multiple drivers and multiple riders (MDMR) ridematching. In this ridematching variant, a driver can be matched with multiple riders and a rider can be matched with multiple drivers. The problem is considered with time windows. We consider the same problem description and the same objective function in the previous chapter with the sole difference that a rider can be matched with more than one driver. The main tasks in this ridematching variant are to assign riders to drivers (matching) and to determine the timing and order of the pickup and delivery of riders (routing of drivers) and to route the riders among the multiple drivers. We impose a limitation on the MDMR ridematching that is a rider can be matched with at most two drivers to make the problem easier to attack.

We start by motivating the MDMR ridematching. Then we discuss how we could achieve this variant of ridematching through the use of transfer points. After that we discuss the necessary modifications to the genetic algorithm proposed in the previous chapter to handle the presence of transfers. Then we compare the results of the algorithm in view of MDMR ridematching and SDMR ridematching (considered in the previous chapter)[1].

7.1 Motivation

We motivate the use of multiple drivers and multiple riders ridematching using the scenario in Figure 7.1. For some given settings, one driver could not be able to serve the riders' request alone because of the distance/time constraints discussed in the previous chapter. However, if two drivers cooperate to serve the rider' request, it could be possible to serve it and this would increase the number of served riders' requests in the whole system. One driver picks up the rider from his source point and drops him off at a

[1]The findings in this chapter are published in [44]

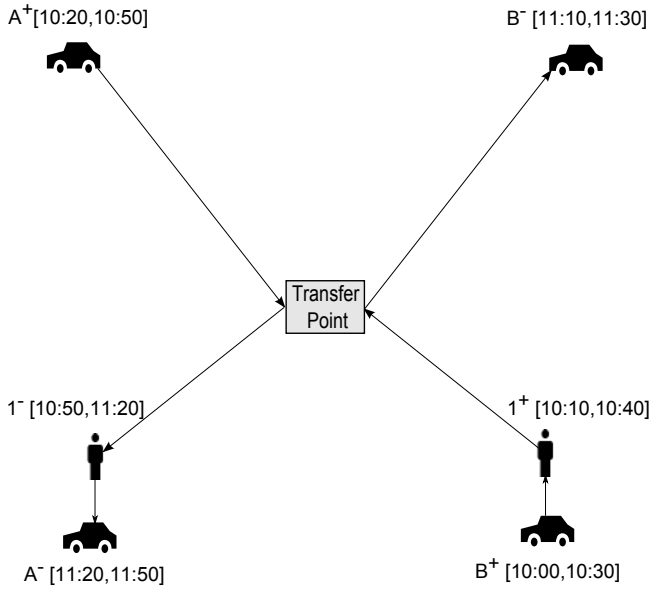


Figure 7.1: Drivers cooperation to serve a rider

transfer point, and then the second driver picks up the rider from the transfer point and drops him off at his destination point. This requires the addition of two new constraints which are the spatial and temporal connectivity constraints. Spatial connectivity means that if two drivers cooperate to serve a rider' request using a transfer point, then the second driver shall pick up the rider from the transfer point at which the first driver dropped him off. Temporal connectivity means that the second pickup of the rider from the transfer point shall be no earlier than his drop-off at the transfer point.

7.2 Transfer Points for MDMR Ridematching

The idea of MDMR ridematching is achieved through the proper use of transfer points. A riders' request is split into two different requests through some transfer point where two drivers could cooperate to serve them. We define a set of transfer points, with fixed geographical locations, to be the possible places where a rider can change a vehicle. For each rider's request, we create a pair of pickup and delivery points exactly as we do in the SDMR ridematching in the previous chapter. In addition, for each rider's request and each transfer point, we check if the rider's request can be served by splitting it through the transfer point. Given a pickup point i and delivery point j for the rider's request r and a transfer point x, we check if the time required to travel from the pickup point i to x and from x to the delivery point j is less than the maximum travel time allowed by the rider: $t_{i,x} + t_{x,j} \leq MTT_r$. If it is feasible to serve the request through transfer x, then we create two pairs of pickup and delivery points (connecting pairs for each other) as shown in Figure 7.2.

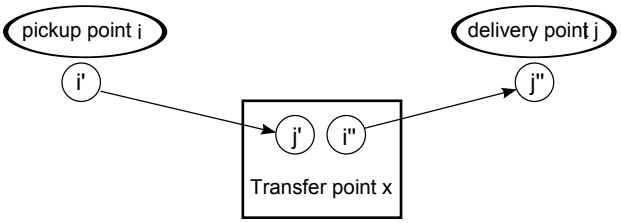


Figure 7.2: Splitting a request through a transfer point.

The first pair is a pickup point $i^{'}$ from i and a delivery point $j^{'}$ to x and the second pair is a pickup point $i^{''}$ from x and a delivery point $j^{''}$ to j. The setting of the time windows of the new pickup and delivery points is visualized in Figure 7.3. The earliest pickup time, $a_{i'}$, for point $i^{'}$ is set to the earliest departure time ED_r for request r and the latest pickup time, $b_{i'}$, is set to the latest arrival time LA_r for request r minus the direct travel time from $i^{'}$ to $j^{'}$ and from $i^{''}$ to $j^{''}$:

$a_{i'} = ED_r$

$b_{i'} = LA_r - t_{i',j'} - t_{i'',j''}$.

The earliest delivery time, $a_{j'}$, for point $j^{'}$ is set to the earliest departure time of request r plus the direct travel time between $i^{'}$ and $j^{'}$. The latest delivery time, $b_{j'}$, is set to the latest arrival time of request r minus the direct travel time between $i^{''}$ and $j^{''}$:

$a_{j'} = ED_r + t_{i',j'}$

$b_{j'} = LA_r - t_{i'',j''}$.

The earliest and latest pickup times, $a_{i''}$ and $b_{i''}$, for point $i^{''}$ from the transfer point are set to be the same as the earliest and latest delivery time to the transfer point:

$a_{i''} = a_{j'}$

$b_{i''} = b_{j'}$.

The earliest delivery time, $a_{j''}$, for point $j^{''}$ is set to the earliest departure time of request r plus the direct travel time from $i^{'}$ to $j^{'}$ and from $i^{''}$ to $j^{''}$. The latest delivery time $b_{j''}$ of point $j^{''}$ is set as the latest arrival time of request r:

$a_{j''} = ED_r + t_{i',j'} + t_{i'',j''}$.

$b_{j''} = LA_r$

7.3 A Genetic Algorithm for MDMR Ridematching

We modify the genetic algorithm proposed to solve the SDMR ridematching in the previous chapter to be used to solve the MDMR ridematching.

7.3.1 Solution Representation

Very similar to the solution representation for SDMR ridematching in the previous chapter, a solution is represented as a schedule of v routes with one route for each vehicle (driver). Each route starts with the source point of the vehicle followed by a list of

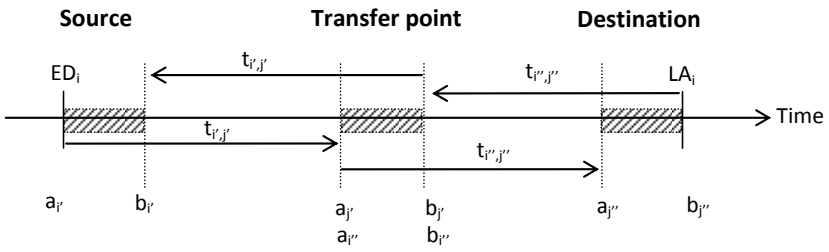


Figure 7.3: Time window calculation at source, transfer point and destination of riders' request i.

riders' pickup and delivery points and ends with the destination point of the vehicle. The difference in the solution representation in this chapter is because of the fact that a rider's request could be served through a transfer point (multiple drivers). If a rider's request is being served through a transfer point, then the two pairs of pickup and delivery points of that request will appear in two different routes. In the previous chapter, a rider's request is represented by only one pair of pickup and delivery points and can appear in one route only. Figure 7.4 shows an example solution representation matching 5 vehicles (A - E) and 11 riders' requests (1 - 11).

A^+	5^+	5^-	4^+	7^+	4^-	7^-	A^-		
B^+	$1'^+$	3^+	3^-	$1'^-$	B^-				
C^+	8^+	2^+	8^-	2^-	C^-				
D^+	6^+	6^-	D^-						
E^+	$1''^+$	$1''^-$	9^+	11^+	9^-	11^-	10^+	10^-	E^-

Figure 7.4: Solution representation. Five Vehicles (A-E) and eleven matched riders requests (1-11). Request 1 is served with transfer. + - denote the pickup and delivery points of riders requests and the sources and destinations of the vehicles

7.3.2 Solution Initialization

A solution is initialized in the same way as done for the SDMR ridematching in the previous chapter with two major differences. First, there are different pairs of pickup and delivery points representing a single rider's request compared to only one pair for each request in the SDMR ridematching in the previous chapter. Among the different pairs, we randomly select a pair to be inserted in the schedule. If the inserted pair contains a transfer, we try to insert its connecting pair. If it is not feasible to insert a pair or its connecting pair (in case of a request being served with multiple drivers), we randomly select a different pair for the same rider's request. This process is repeated until the rider's request is marked served or non-servable otherwise.

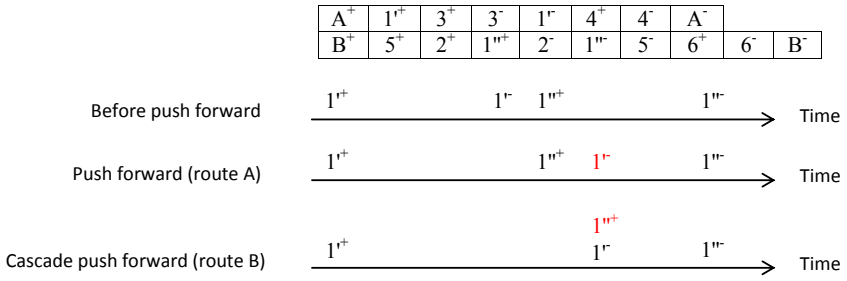


Figure 7.5: Cascade push forward. Delivery point $1^{'-}$ of riders' request 1 in route A is pushed forward followed by a push forward for the pickup point $1^{''+}$ of the same riders' reuest in route B

Second, trying to insert a pickup or delivery point in some route could result in time push forward as discussed in chapter 6. In the case of SDMR ridematching, this could result in further push-forwards in the same route. However in the case of MDMR ridematching, with the presence of transfers, this could result in what we call cascade push-forwards in different routes to maintain the temporal connectivity.

Let the point p_i be the point before which we try to insert a new point in route $p = (p_1, p_2, ..., p_u)$. The insertion of new point between p_{i-1} and p_i, $1 < i \leq u$, could result in time push forward PF_i at point p_i: $PF_{p_i} = newT^k_{p_i} - T^k_{p_i} \geq 0$ where $newT^k_{p_i}$ is the new service starting time at point p_i after inserting a new point before it. It could also result in time push forward at the points after p_i in the route: $PF_{p_{j+1}} = max\{0, PF_{p_j} - w_{p_{j+1}}\}$, $i \leq j < u$ as discussed in the previous chapter.

To maintain the temporal connectivity, we check if each pushed forward point p_i in route p requires cascade push forward. If p_i is a delivery point at some transfer station, we do cascade push forward. This includes pushing forward the pickup point p^*_j from the same transfer point in the connecting pair in route p^* such that:

$$PF_{p^*_j} = max\{0, T_{p_i} - T_{p^*_j}\}.$$

Figure 7.5 is a pictorial example of cascade push forward. The constraints time window and maximum travel time and distance are checked after the insertion of each new point in the schedule in the same way done in chapter 6. Any insertion that violates one of these constraints is rejected.

7.3.3 Crossover Operator

The crossover operator proposed for SDMR ridematching in Chapter 6 requires few modifications to be suitable for the MDMR ridematching in this chapter. The child solutions of the crossover operation in the SDMR ridematching could violate only the constraints (6.8) and (6.9) (a riders' request could be matched twice with different vehicles). In the ridematching problem considered in this chapter, the result of the crossover operator might violate the constraints (6.8) and (6.9) and the temporal and spatial connectivity

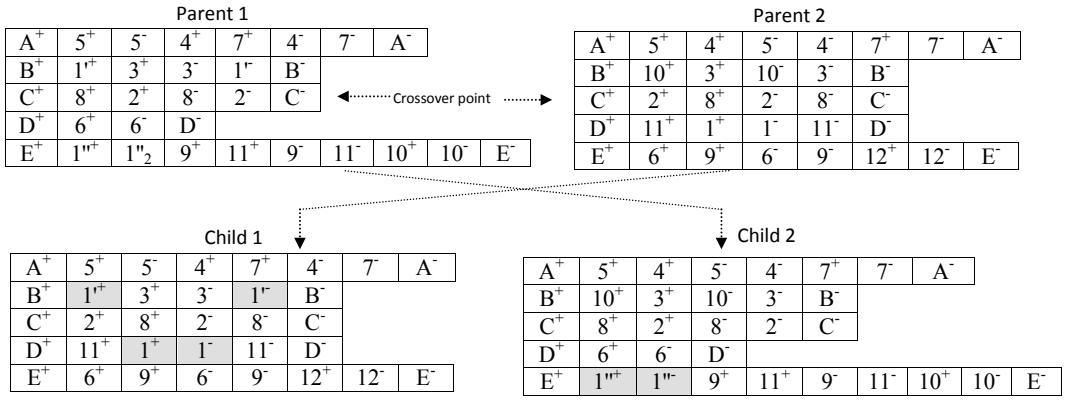


Figure 7.6: Single point crossover operator. Shaded points are constraint violating points

constraints. A request could be scheduled twice, or one of the two pairs of pickup and delivery points for serving a request with transfer could be scheduled while the connecting pair is missing in the schedule (violating spatial connectivity) as in Figure 7.6. In addition, the connecting pair could exist in the schedule but violates the temporal connectivity constraint. After each crossover operation, we apply a repair operation that removes all the constraint violating points followed by a try to insert as much as possible from points of the non-matched requests.

7.3.4 Mutation Operators

The proposed mutation operators in the previous chapter for SDMR ridematching require some modifications to be used in MDMR ridematching. In the following we discuss the required modifications for each mutation operator.

We start with the adaptation of the push backward mutation operator. As proposed in Chapter 6 for SDMR ridematching, a point p_i is randomly selected from some route p of driver k. The point p_i is pushed backward with a value $PB_{p_i} = random(T^k_{p_i} - a_{p_i})$ (a random percentage of the time difference between the current service starting time of the selected point and its earliest service time). The points after p_i in route p are pushed backward too: $PB_{p_{j+1}} = min\{PB_{p_j}, T^k_{p_{j+1}} - a_{p_{j+1}}\}$, $i \leq j < u$.

For the SDMR ridematching, only local push-backwards in route p for the points after p_i are required. However, in the MDMR ridematching, special care has to be taken to maintain the temporal connectivity of the requests served with transfers. Let p_i be a pickup point at some transfer point in route p. If p_i is pushed backward, then the temporal connectivity has to be maintained. To maintain the temporal connectivity, the delivery point p^*_j (at the same transfer point) of the same request in route p^* has to be pushed backward with value $PB_{p^*_j}$:

$$PB_{p^*_j} = max\{0, T_{p^*_j} - T_{p_i}\}$$

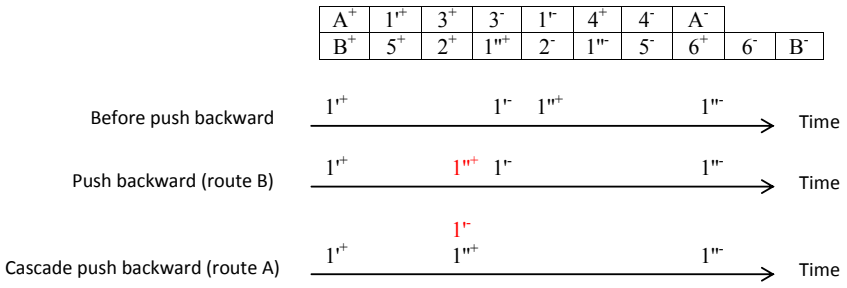


Figure 7.7: Push backward mutation. Pickup point $1^{''+}$ of riders' request 1 in route B is pushed backward followed by a push backward for the delivery point $1^{'-}$ of the same riders' reuest in route A

Note that the push backward of p_j^* could result in a set of push-backwards in cascading manner. Therefore, we push backward p_j^* only if $T_{p_j^*} - T_{p_i} > 0$ to reduce the number of push backward operations. The other option would be to push backward p_j^* always with a value of PB_{p_i}. Figure 7.7 shows an example push backward mutation for the MDMR ridematching.

The idea of the adaptation of the second mutation operator (push forward) is almost covered in the discussion of the adaptation of the solution initialization as it includes push forward operation. As proposed for the SDMR ridematching in the previous chapter, a point p_i is randomly selected in some route p. Then p_i is pushed forward with a value $PF_{p_i} = random(b_{p_i} - T^k_{p_i})$ (a random percentage of the time difference between the latest service time of the selected point and its current service time). The points after p_i in route p are pushed forward too: $PF_{p_{j+1}} = max\{0, PF_{p_j} - w_{p_{j+1}}\}$, $i \leq j < u$. To maintain the temporal connectivity, cascade push-forwards might be necessary and are done in the same way as discussed in solution initialization.

The third mutation operator (remove-insert operator) includes removing a pair of pickup and delivery points from a specific route p. The only required modification is that when we remove a pair of pickup and delivery points from p, we check if the removed pair contains transfer. If so, we delete the connecting pair from the other route p^*. The insert operation is the same as the insertion done during the initialization phase and follows the same adaptation.

The swap mutation operator includes swapping two randomly selected adjacent points in some route as long as they do not belong to the same request. No additional adaptation is required. Swapping the two points involves removing them and reinserting them each in the position of the other point. All necessary adaptation is covered in the insertion step as done during the initialization phase. Regarding the transfer mutation, we realized that it is destructive for both the SDMR ridematching and the MDMR ridematching and therefore we excluded it.

7.3.5 Experimentation

We compare the results of the SDMR ridematching and the behavior of the MDMR ridematching on the same problem instance. The goal of the comparison is to prove that there could be a scenario where the MDMR ridematching outperforms the SDMR ridematching.

Problem Instance

The instances used in chapter 6 are not suitable for MDMR ridematching as the trips are far in time from each other and this makes it hard to find two drivers cooperating to serve a rider' request. We propose a new instance with 200 trips (50 drivers and 150 riders) in an area of $105.5km \times 133.5km$. The trips are extracted from a travel and activity survey for northeastern Illinois conducted by Chicago Metropolitan Agency for Planning CMAP [2]. The earliest departure among the 200 trips is 11:35 and the latest departure is 12:00. Note that these trips are close in time to each other which makes it feasible for drivers' cooperation to serve riders' requests. An important parameter for studying the MDMR ridematching is the participants maximum travel time (MTT) as we see soon. To be able to study different values of the MTT parameter, we provide wide time windows for the trips ($\pm$ Random(280,320)).

We define 29 transfer points in the considered area. In this study, we are not interested in the optimal distribution of the transfer points. The selection of the transfer points is as follows. First, we have defined a grid of transfer points over the area with granularity of $2km$ which results in 11616 transfer points. Then we performed a single run of the MDMR ridematching for the 200 trips considering the 11616 transfer points. Among the 11616 transfer points, 29 transfer points were utilized during the single run of the MDMR ridematching which we consider the transfer points in this study.

Settings of the Problem Instance and the Genetic Algorithm

The maximum drivers' travel distance MTD is set to 1.2 of their direct travel distance. The maximum travel time MTT of the riders and drivers takes different values: $1.2x$, $1.6x$, $2.0x$, $2.4x$, $2.8x$, $3.2x$ and $10.0x$ ($MTT = 1.2x$ means that each participant allows a maximum travel time of 1.2 of his direct travel time from his source to his destination). The objectives weights are: $\alpha = 0.7$ (for served riders' requests) and $\beta = \gamma = \delta = 0.1$ (for drivers' travel distance, drivers' travel time and riders' travel time respectively).

The parameters' settings of the genetic algorithm are: population size = 100, number of generations = 100, crossover probability = 1.0 and mutation probability =0.4. The algorithm is executed 30 independent times for each MTT considering SDMR and MDMR ridematching. The experimentation platform is the same platform described in Section 3.5.

[2] http://www.cmap.illinois.gov/travel-tracker-survey.

Results Discussion

Tables 7.1 and 7.2 summarize the results of the experiments considering SDMR and MDMR ridematching respectively under different values of MTT. The entries of the tables are visualized as boxplots in figures 7.8 - 7.12 for better presentation.

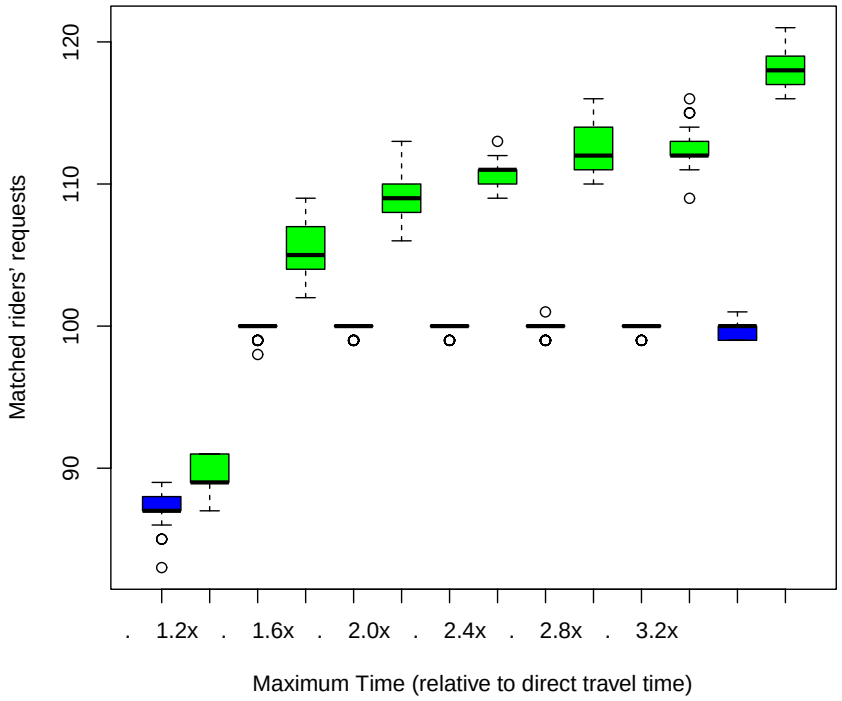


Figure 7.8: Boxplot for the number of matched riders' requests.

The main goal of the MDMR ridematching is to increase the number of matched (served) riders' requests. Figure 7.8 shows the number of served riders' requests under different values of MTT. Under $MTT = 1.2x$, we notice that the MDMR ridematching did not add significant improvement over the SDMR ridematching. However, the gain of MDMR ridematching increases with the increase of the MTT.

The MDMR ridematching made it possible for some riders' requests to be served which could not be served with SDMR ridematching. However, and following the theory of no free lunch, the use of MDMR ridematching adds additional waiting time for both riders and drivers. The optimal scenario of a rider trip with multiple drivers is when the first driver drops the rider at the transfer point and directly after that the second driver arrives to the transfer point to pick up the rider. Unfortunately, the optimal scenario is rare. Note that this optimal scenario does not require waiting neither for the rider nor for the second driver. If the rider arrives too early, then he has to wait for the second driver. If the second driver arrives before the rider, then he also needs to wait for the rider. Hence, if the constraint MTT has a small value, then this prevents matching a

Table 7.1: Summary of the results of the genetic algorithm considering SDMR ridematching and different values of MTT. Mean and standard deviation μ_σ.

	$1.2x$	$1.6x$	$2.0x$	$2.4x$	$2.8x$	$3.2x$	$10.0x$
Objective function	$0.412_{0.0030}$	$0.332_{0.0020}$	$0.317_{0.0020}$	$0.31_{0.0030}$	$0.307_{0.0020}$	$0.304_{0.0010}$	$0.296_{0.0020}$
No. Matched Riders' Requests	$87.1_{1.165}$	$99.767_{0.496}$	$99.767_{0.423}$	$99.867_{0.34}$	$99.833_{0.453}$	$99.867_{0.34}$	$99.767_{0.496}$
Total Drivers Travel Distance	$2356.443_{9.239}$	$2427.698_{7.177}$	$2424.565_{6.731}$	$2423.995_{5.836}$	$2424.419_{7.91}$	$2422.593_{6.282}$	$2423.648_{7.324}$
Total Drivers Travel Time	$2446.7_{10.172}$	$2531.433_{7.256}$	$2530.0_{7.062}$	$2530.033_{5.498}$	$2530.367_{7.969}$	$2529.8_{6.306}$	$2531.767_{6.761}$
Total Riders Travel Time	$1222.133_{21.38}$	$1466.367_{16.75}$	$1482.167_{15.91}$	$1484.133_{10.862}$	$1484.6_{18.224}$	$1487.867_{13.268}$	$1493.567_{15.937}$
Runtime	$24.974_{1.321}$	$26.994_{1.167}$	$27.283_{0.96}$	$26.946_{0.953}$	$26.87_{0.889}$	$27.228_{0.947}$	$27.959_{1.325}$

Table 7.2: Summary of the results of the genetic algorithm considering MDMR ridematching and different values of MTT. Mean and standard deviation μ_σ.

	$1.2x$	$1.6x$	$2.0x$	$2.4x$	$2.8x$	$3.2x$	$10.0x$
Objective function	$0.409_{0.0020}$	$0.326_{0.0050}$	$0.308_{0.0050}$	$0.3_{0.0050}$	$0.293_{0.0070}$	$0.291_{0.0050}$	$0.257_{0.0060}$
No. Matched Riders' Requests	$89.467_{1.231}$	$105.267_{1.59}$	$109.1_{1.399}$	$110.533_{1.118}$	$112.433_{1.43}$	$112.6_{1.306}$	$118.433_{1.453}$
Total Drivers Travel Distance	$2364.541_{9.51}$	$2448.65_{8.153}$	$2462.99_{10.251}$	$2464.9_{8.347}$	$2467.648_{10.972}$	$2467.063_{9.059}$	$2485.071_{10.335}$
Total Drivers Travel Time	$2459.433_{10.285}$	$2620.533_{25.191}$	$2838.433_{67.756}$	$3017.4_{98.02}$	$3280.333_{112.732}$	$3480.567_{124.296}$	$5396.8_{322.814}$
Total Riders Travel Time	$1263.967_{21.527}$	$1627.433_{48.589}$	$1806.833_{55.143}$	$1978.433_{86.33}$	$2151.433_{84.007}$	$2275.567_{102.196}$	$4746.633_{360.119}$
Runtime	$101.928_{8.32}$	$185.035_{15.003}$	$249.98_{18.006}$	$302.598_{26.219}$	$362.509_{28.198}$	$437.341_{25.487}$	$920.774_{65.667}$

rider with multiple drivers as it requires travel time (including waiting time) more than what the riders and drivers tolerate. Therefore, as Figure 7.8 shows, by increasing the MTT, the number of served riders' requests increases. Increasing the MTT means that the participants show more tolerance for waiting and this enables matching a rider with multiple drivers.

We notice that increasing the MTT from $1.2x$ to $1.6x$ helps to serve more riders' requests in SDMR ridematching. However, increasing the MTT more than $1.6x$ does not help in increasing the number of served riders' requests because of the fact that the travel distance becomes a limiting factor. This gives evidence that the use of MDMR ridematching could be helpful to reduce the effect of the distance constraint.

Figure 7.9 shows the total drivers travel distance. Increasing the MTT from $1.2x$ to $1.6x$ increases the total travel distance in the SDMR and MDMR ridematching. Increasing the MTT more than $1.6x$ does not increase the total drivers travel distance in the SDMR ridematching. However, in the MDMR ridematching, increasing the MTT increases the total drivers travel distance. This indicates that after $MTT = 1.6x$, some drivers are left with additional possible travel distances (distance fractions). The distance fractions are not sufficient alone to serve a single riders' request and therefore not utilized in the SDMR ridematching. However using the MDMR ridematching, two drivers cooperate to serve a single riders' request (by adding their distance fractions). This makes it possible to utilize the drivers' additional possible travel distances.

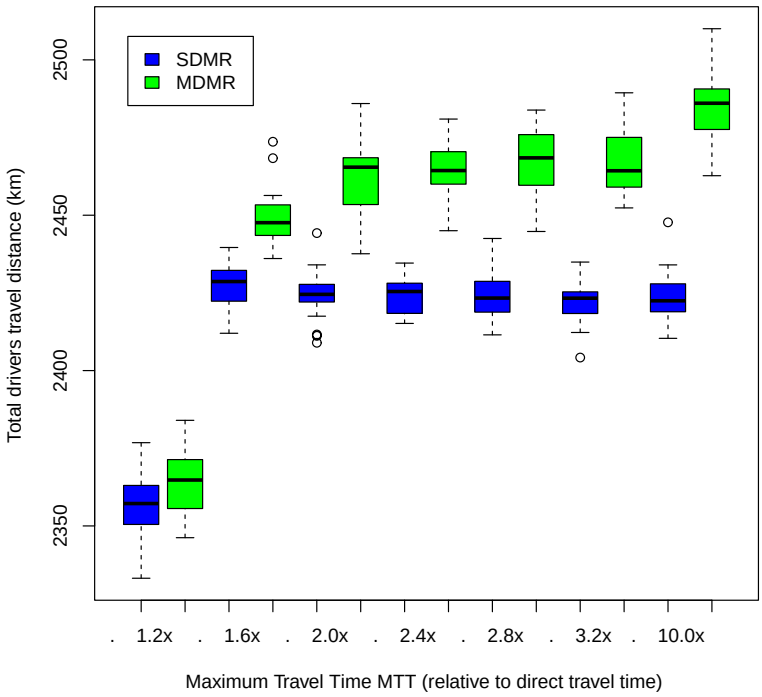


Figure 7.9: Boxplot for the total drivers' travel distance.

Figure 7.10 shows the total travel time of the drivers. Increasing the MTT more than $1.6x$ does not increase the total drivers' travel time in the SDMR ridematching because of the distance constraint. In the MDMR ridematching, increasing the MTT increases the total drivers travel time. The total drivers travel time consists of the actual travel time and some waiting time for riders. By comparing the values of the travel distances and travel times in table 7.2 and figures 7.9 and 7.10, we notice that a considerable amount of the drivers' travel time is waiting time in the MDMR ridematching.

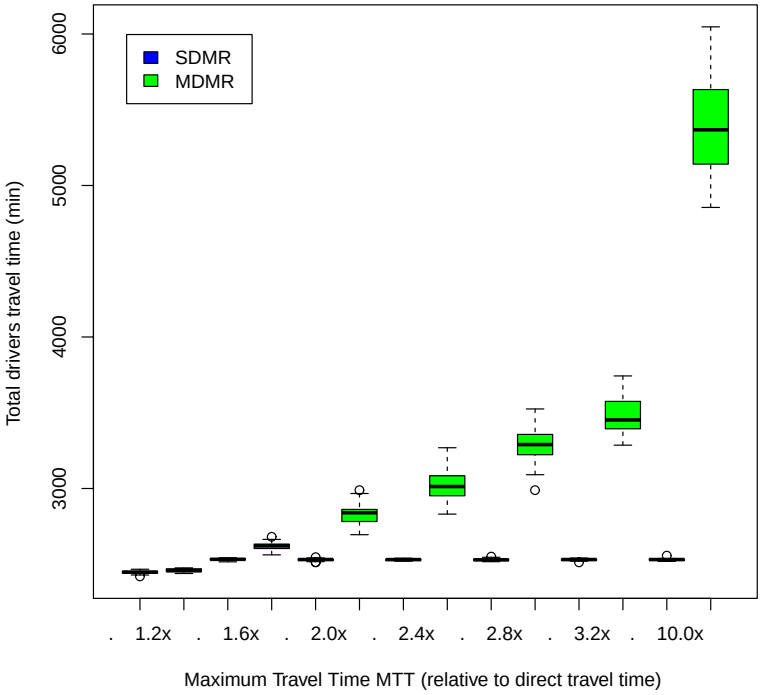


Figure 7.10: Boxplot for the total drivers' travel time.

The total riders' travel time is shown in Figure 7.11. It increases with the increase of the number of served riders' requests. The number of served riders' requests increases with the increase of the MTT as discussed previously. Note that if a vehicle (driver) is waiting for some rider at some pickup point, then the waiting time will be multiplied by the number of the riders already in the vehicle.

The overall image comparing the SDMR and MDMR ridematching is presented in Figure 7.12. The figure shows the values of the objective function. We notice that for each MTT, the MDMR ridematching improves the values of the objective function over the SDMR ridematching. The improvement increases with the increase of the MTT. One misleading result is that in the case of SDMR ridematching, the value of the objective function decreases with the increase of the MTT. However the results indicate that the values of the components of the objective function remains almost constant after $MTT = 1.6x$ for the SDMR ridematching.

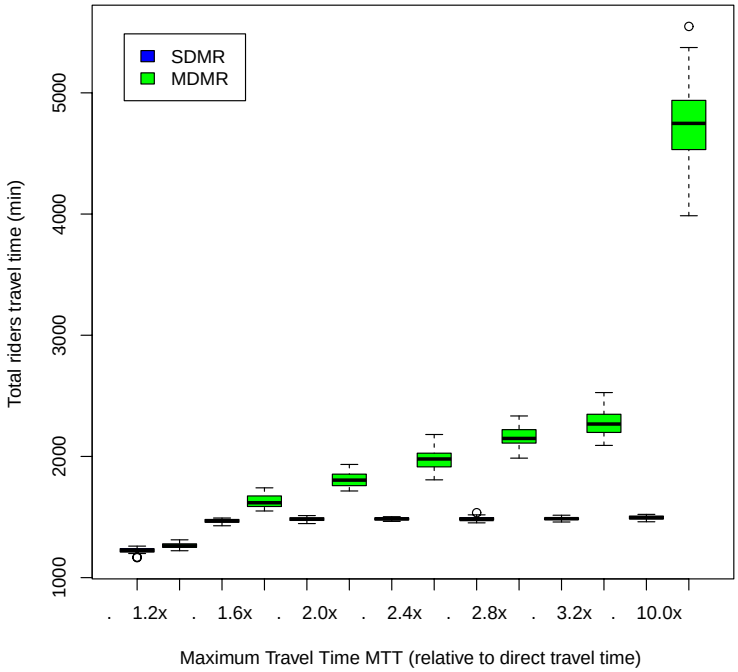


Figure 7.11: Boxplot for the total riders' travel time.

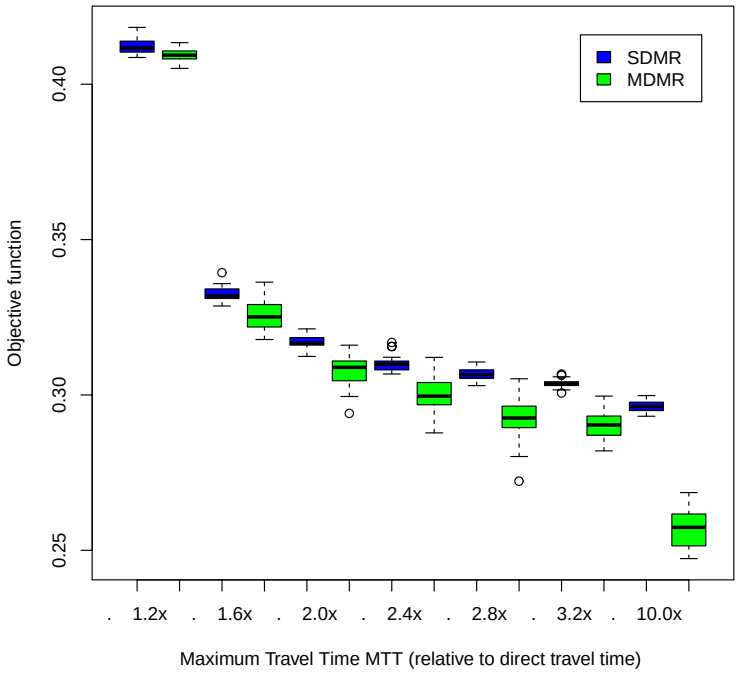


Figure 7.12: Boxplot for the values of the objective function.

The misleading values of the objective function in the SDMR ridematching are because of the normalization we do before calculating it. We normalize the travel time for the riders and drivers by dividing the actual total travel time by the total maximum travel time minus the total minimum travel time (direct travel time). Increasing MTT increases the total maximum travel time and the rest remains constant which results in larger denominator. This means that for larger MTT, we get smaller values of the drivers and riders total travel time and hence lower values of the objective function. Still the objective function comparison between the two ridematching variants for the same MTT is meaningful.

Figure 7.13 shows the runtime of the genetic algorithm. It is clear that the required runtime for MDMR ridematching is more than the required runtime for the SDMR ridematching. The runtime increases with the increase of MTT. The search space is larger for the MDMR ridematching and becomes larger with the increase of the MTT.

Fugure 7.14 shows an example result of selected drivers' trips considering the MDMR ridematching. Drivers meet at the transfer point (red point in the Figure) to exchange riders.

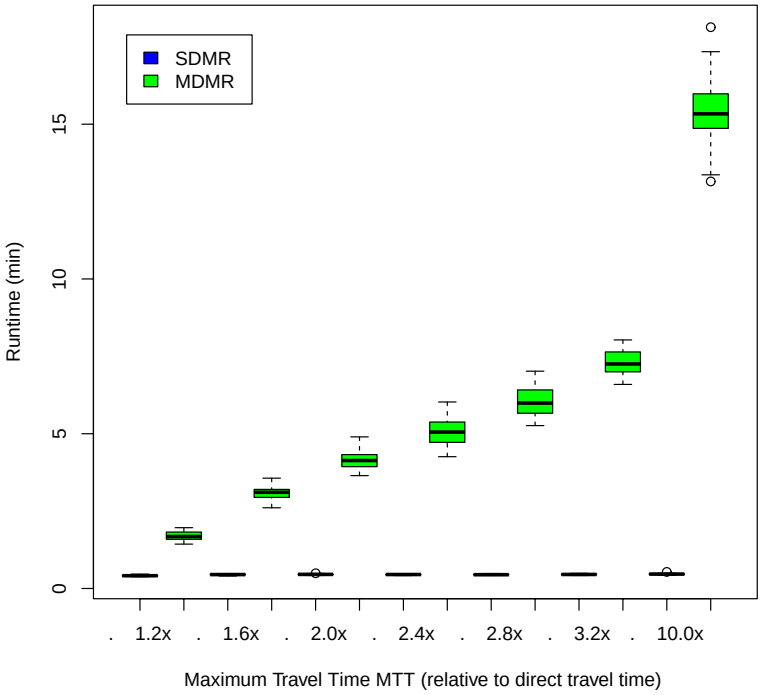


Figure 7.13: Boxplot for runtime of the genetic algorithm

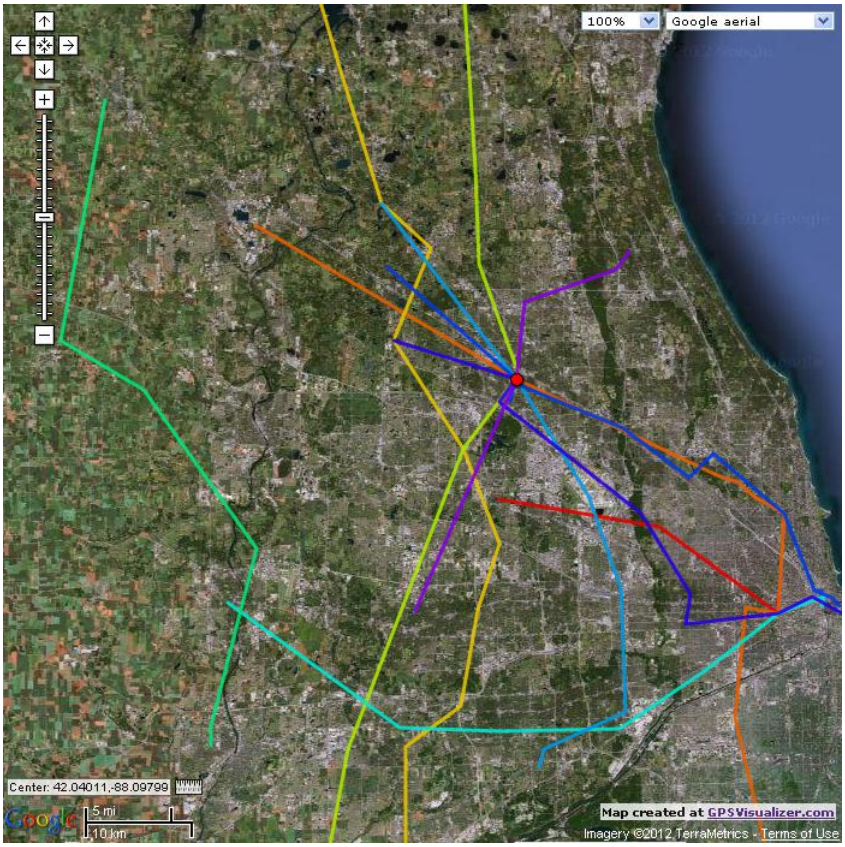


Figure 7.14: Visualization of selected vehicles' trips. The red point is the transfer point where vehicles exchanged riders. The different lines represent the trips of the different drivers from their sources to their destinations including visiting the pickup and delivery points of the served riders. The used GPX visualizer is: http://www.gpsvisualizer.com.

7.4 Summary

In this chapter we have considered the multiple drivers and multiple riders ridematching. The problem is considered with time windows and with a limitation that a rider can be matched with at most two drivers. We have adapted the genetic algorithm proposed in the previous chapter to solve the considered problem in this chapter. Experimentation results on a ridematching problem instance built on real world trip traces indicate that the multiple drivers and multiple riders ridematching could increase the number of served riders' requests as compared with the single driver and multiple riders ridematching on the cost of increasing travel time of the riders and drivers because of the increase in their waiting time.

Chapter 8

Summary

Ridesharing is considered as one of the transportation demand management strategies that can be used to reduce the power consumption, air pollution and traffic congestion. Dynamic ridesharing is a flexible form of ridesharing that is encouraged by the widespread of ubiquitous handheld mobile devices where a rideshare can be arranged on a very short notice. Recently there are many active research areas in dynamic ridesharing including but not limited to security, privacy protection, payment and ridematching. This work is mainly focused on solving the ridematching problem in dynamic ridesharing.

The ridematching problem is to match riders and drivers who wish to participate in ridesharing in the best possible way taking into account the timing and end points of the their trips and other constraints that can be imposed by the participants. The timing of the participants' trips could be points in time or time intervals where the participants specify an earliest and latest departure/arrival times to form what we call time windows. To be able to attack the problem, we consider four different variants of the ridematching problem and attack each variant independently going from the more specific to the more general variant. Given two sets of drivers and riders to be matched for ridesharing, the four different ridematching variants are defined based on whether a rider can be matched with only one driver or more and whether a driver can be matched with only one rider or more.

The first and simplest variant of ridematching is when a rider can be matched at most with one driver and a driver can be matched with at most one rider. This ridematching variant is called single driver and single rider ridematching and it is solved to optimality in the literature considering points in time for timing the trips of the ridesharing participants. This variant becomes harder to solve when considering time windows instead of points in time for the participants' trips. We indirectly solve this variant of ridematching with time windows by solving the third and fourth variants of ridematching with time windows as they are more general.

The second variant of ridematching is the multiple drivers and single rider ridematching where a rider can be matched with multiple drivers and a driver can be matched at most with one rider. We attack this variant considering that the participants define

points in time for their trips (no time windows) to make it easier. Only oversimplified version of this ridematching variant is considered in the literature such that a rider can be matched at most with two drivers and only if a match with a single driver is not found and the waiting time during the rider's trip is not considered. We model and solve this ridematching variant in its general form where a rider can be matched with many drivers at different times of his trip and we consider modeling the waiting times of the riders when sharing trips with multiple drivers. The problem is formulated and modeled as a multiobjective route planning problem for the riders where for a rider's request we return a set of equally good route plans, regarding a set of objectives, for the rider to select among them. The set of objectives to be minimized includes the total trip time, cost and number of drivers in a route plan. The multiobjective route planning is first solved using an off-the-shelf exact algorithm, and then because of the bad scalability of the exact algorithm we propose a genetic algorithm metaheuristic to solve it. The proposed genetic algorithm is then improved considering different solutions' representation and through hybridizing it with local search. All in all, the genetic algorithm provides good quality results in reasonable time and with good scalability compared with the exact algorithm. Finally we compare the performance of the genetic algorithm with the performance of an ant-colony algorithm proposed to solve the same problem where the genetic algorithm showed superiority over the ant-colony algorithm.

In the third variant of ridematching a rider can be matched with at most one driver and a driver can be matched with multiple riders. We call this variant the single driver and multiple riders ridematching with the major tasks to assign riders to drivers and to route the drivers (including timing) among the sources and destinations of the riders in the best possible way. For this variant we consider a harder but more realistic timing of the participants' trips such that a participant defines a time window for his departure/arrival (not points in time). The objectives that define the quality of a ridematching solution are: the maximization of the number of matches (served riders), minimization of the total drivers (vehicles) travel distance and time and the minimization of the total riders travel time where the objectives are combined in a single objective to be minimized. We provide a mathematical model for this variant of the problem and propose a genetic algorithm metaheuristic to solve the ridematching problem and minimizing the objective function. The proposed algorithm is tested on a set of problem instances derived from realistic trip traces from a travel and activity survey for northeastern Illinois. The algorithm is able to find reasonable number of ridematches with a small increase in the total drivers travel distance which results in notable reduction in the total participants travel distance. To solve this variant of the ridematching problem in realistic dynamic ridesharing scenarios, we propose an algorithm that switches between the proposed genetic algorithm and an insertion heuristic and can be tuned to balance between the responsiveness of the algorithm and the quality of the solutions.

We go more general in the last variant of ridematching and allow a rider to be matched with multiple drivers at different times of his trip and also we allow a driver to be matched with multiple riders in the so called multiple drivers and multiple riders ridematching (the fourth ridematching variant). This variant is more general than the first and third ridematching variants but actually not more general than the second variant as we limit

the rider to be matched with at most two drivers. However, we consider time windows for the participants' trips and match a driver with multiple riders. The major tasks in this variant are to assign riders to drivers, route the drivers between the sources and destinations of the riders, provide the timing of the drivers' visits to the different sources and destinations of the riders and to route the riders between the drivers. We consider the same objectives in the third variant and extend its model using the idea of transfer points to be used for this variant and we extend the proposed genetic algorithm for solving the third variant to be used to solve this variant. After that we compare the matching results considering the third and fourth variants of ridematching on the same problem instance. The results show that the fourth variant of ridematching can provide larger number of matches (serving larger number of riders) but requires more tolerance from the participants for longer waiting times to increase the number of matches.

Figure 8.1 summarizes the requirements satisfaction of the different variants of the ridematching problem after our contribution. The requirements satisfaction before our contribution is summarized in Figure 2.7.

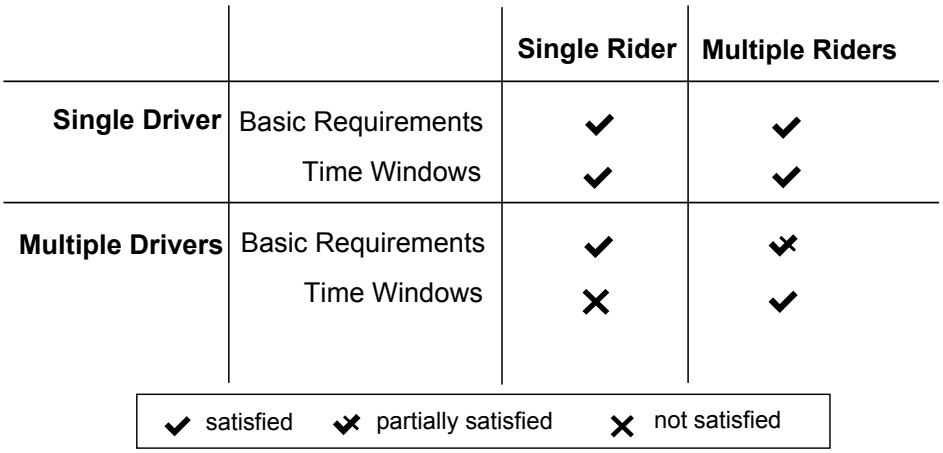


Figure 8.1: Requirements satisfaction of the ridematching variants after our contribution.

8.1 Conclusion

Altogether, this work provided a step forward in solving the ridematching problem in dynamic ridesharing. Our experimentation indicates that the ridematching problem can be efficiently solved in its different variants and a reasonable number of ridematches between drivers and riders can be found which makes the idea of dynamic ridesharing to be appealing and promising. In addition, experimentation indicates that a considerable amount of travel distance can be saved with ridesharing, considering our ridematching implementation, which in turn reduces the fuel consumption and air pollution.

The ridematching problem becomes a very complex problem to model and solve when considering realistic dynamic ridesharing scenarios. On one hand, the ridematching

algorithm has to provide good quality solutions to the ridematching problem in short time, and on the other hand many participants' requirements and constraints have to be modeled and considered by the ridematching algorithm which makes the problem harder and requires more time to solve. To this end, we have considered many constraints that can be imposed by ridesharing participants for realistic ridesharing (for example the time windows and maximum travel distance and time constraints), however other constraints are necessary for more realistic ridesharing as we underline in our outlook.

Because of the computational complexity of the different variants of the ridematching problem in realistic ridesharing scenarios, we conclude that approximate methods (be it heuristic, metaheuristic or any other approximate method) are of a great interest in solving this problem especially for a high demand dynamic ridesharing where the ridematching problem becomes intractable for exact algorithms. We mainly relied on metaheuristics, more precisely genetic algorithms, in this work to solve the ridematching problem which showed success in solving the problem in reasonable time.

8.2 Outlook

There are many directions to build on this work. The first direct extension of this work is to model and solve the multiple drivers and multiple riders ridematching in its general form without limiting the number of drivers, that can be matched with a rider, to two. This extension might increase the number of matches between riders and drivers. Based on our implementation and testing of this variant of ridematching, our intuition says that this extension might require more and more tolerance from the participants for longer waiting times as already discussed, however it worth testing this extension which might give unexpected results.

Among the most interesting directions to extend this work is to consider utilizing the social networks during the ridematching. Social networks are valuable source of information that can improve the quality of the ridematching in all its variants. Matching participants with similar interests could add a great advantage to ridesharing and such information about the interests and hobbies of the participants can be easily found in social networks. In addition, matching participants with social closeness as a priority can also be achieved through the connection with social networks besides accessing the participants' calendars to infer future trips.

Another extension to this work is to consider realistic travel times using existing APIs instead of the theoritical travel times. In addition, considering time dependent travel times between two points would make the ridematching more realistic. During peak times, the travel time between two points might be much more that the travel time between the same two point at times other than the peak times and hence this information can be of great interest to the ridematching algorithm.

In our formulation of the ridematching problem, a driver defines a maximum travel distance that defines a maximum detour distance to serve riders. To this extent, we impose no constraint on the amount of detour distance to serve a rider and the driver can spend his maximum detour distance to serve only single rider. For more realistic

ridematching, limiting the maximum detour distance per rider should be taken into consideration. In addition, an interesting idea is to link the amount of payment and the maximum detour per single rider. The amount that a rider is willing to pay for a driver can define the maximum detour distance that can be made by the driver to serve that rider in a way similar to auction. Riders tend to pay more to motivate drivers to make larger detours and to increase their chances to be served by drivers.

The time windows of the participants' trips are considered hard constraints in our problem formulation that cannot be violated. An interesting thread of research is to study the effect of considering soft time windows on the results of the ridematching. Soft time windows can be violated which requires some constraints violation handling strategy. One approach to handle the violation of the time windows is some sort of penalization in the objective function. There exist other approaches for handling constraints violation which is an active research area in itself. Allowing time windows constraint violation could results in promising solution that would not have been reached with hard time windows constraints.

List of Abbreviations

ACO	Ant Colony Optimization
DMD	Divers with maximum Distance
DTD	Direct Travel Distance
DTT	Direct Travel Time
GA	Genetic Algorithm
GD	Generational Distance
gGA	Generational Genetic Algorithm
GLC	Generalized Label Correcting
GLS	Genetic Local search
HV	Hypervolume
IGD	Inverted Generational Distance
KRR	Known Riders' Requests
$m-ACO$	Multiobjective Ant Colony Optimization
$MDMR$	Multiple Drivers and Multiple Riders
$MDSR$	Multiple Drivers and Single Rider
$MOEA$	Multiobjective Evolutionary Algorithm
$MSPP$	Multiobjective Shortest Path Problem
PSO	Particle Swarm Optimization
RW	Random Walks

$SDMR$	Single Driver and Multiple Riders
$SDSR$	Single Driver and Single Rider
TW	Time Window

Bibliography

[1] Agatz, N., Erera, A., Savelsbergh, M., and Wang, X. Sustainable passenger transportation: Dynamic ride-sharing. Tech. rep., Erasmus Research Inst. of Management (ERIM), Erasmus Uni., Rotterdam, 2010.

[2] Agatz, N. A., Erera, A. L., Savelsbergh, M. W., and Wang, X. Dynamic ride-sharing: A simulation study in metro atlanta. *Transportation Research Part B: Methodological 45*, 9 (2011), 1450–1464.

[3] Alaya, I., Solnon, C., and Ghedira, K. Ant colony optimization for multi-objective optimization problems. In *Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence - Volume 01* (Washington, DC, USA, 2007), ICTAI '07, IEEE Computer Society, pp. 450–457.

[4] Bandyopadhyay, S., Saha, S., Maulik, U., and Deb, K. A Simulated Annealing-Based Multiobjective Optimization Algorithm: AMOSA. *Evolutionary Computation, IEEE Transactions on 12*, 3 (2008), 269–283.

[5] Baos, R., Gil, C., Paechter, B., and Ortega, J. A hybrid meta-heuristic for multi-objective optimization: Mosats. *Journal of Mathematical Modelling and Algorithms 6* (2007), 213–230.

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

[7] Baumal, A., McPhee, J., and Calamai, P. Application of genetic algorithms to the design optimization of an active vehicle suspension system. *Computer Methods in Applied Mechanics and Engineering 163* (1998), 87–94.

[8] BTS. *Highlights of the 2001 National Household Travel Survey.* U.S. Dept. of Transportation., Bureau of Trans. Statistics, Washington, 2003.

[9] Burke, E. K., and Cowling, P. Combining hybrid metaheuristics and populations for the multiobjective optimisation of space allocation problems. In *Pro-*

ceedings of the international conference on Genetic and evolutionary computation conference (2001), GECCO'01, Morgan kaufmann, pp. 1252–1259.

[10] Burke, E. K., and Kendall, G. *Search methodologies : introductory tutorials in optimization and decision support techniques*. Springer, 2005.

[11] CASEY, R. F., Labell, L. N., and Holmstrom, R. Advanced public transportation systems: The state of the art update '96. Tech. rep., Transportation Research Board, Washington, D.C, 1996.

[12] Chaube, V. Understanding and designing for perceptions of trust in rideshare programs. Master's thesis, Virginia Polytechnic Institute and State University, 2010.

[13] Chaube, V., Kavanaugh, A. L., and Perez-Quinones, M. A. Leveraging social networks to embed trust in rideshare programs. In *Proceedings of the 43rd Hawaii International Conference on System Sciences* (Washington, DC, USA, 2010), HICSS '10, IEEE Computer Society, pp. 1–8.

[14] Cheng, R., Gen, M., and Tsujimura, Y. A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: hybrid genetic search strategies. *Computers and Industrial Engineering 36*, 2 (1999), 343–364.

[15] Chitra, C., and Subbaraj, P. A nondominated sorting genetic algorithm solution for shortest path routing problem in computer networks. *Expert Systems with Applications 39*, 1 (2012), 1518–1525.

[16] Cho, C.-W., Wu, Y.-H., Yen, C., and Chang, C.-Y. Passenger search by spatial index for ridesharing. In *Proceedings of the International Conference on Technologies and Applications of Artificial Intelligence* (Washington, DC, USA, 2011), TAAI '11, IEEE Computer Society, pp. 88–93.

[17] Corne, D. W., Jerram, N. R., Knowles, J. D., Oates, M. J., and J, M. PESA-II: Region-based selection in evolutionary multiobjective optimization. In *Proceedings of the international conference on genetic and evolutionary computation conference* (2001), GECCO'01, Morgan Kaufmann Publishers, pp. 283–290.

[18] Costelloe, D., Mooney, P., and Winstanley, A. From random walks to pareto optimal paths. In *Proceedings of the 12th Irish AI and CogScience* (2001), MEditors, Ed., AICS'01.

[19] Czyzzák, P., and Jaszkiewicz, A. Pareto simulated annealing–a metaheuristic technique for multiple-objective combinatorial optimization. *Journal of Multi-Criteria Decision Analysis 7*, 1 (1998), 34–47.

[20] Dailey, D. J., Loseff, D., and Meyers, D. Seattle smart traveler: dynamic ridematching on the world wide web. transportation research 7c, 1999.

[21] Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE TEC 6*, 2 (2002), 182–197.

[22] Dorigo, M. *Optimization, Learning and Natural Algorithms.* PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy, 1992.

[23] Dorigo, M., and Sttzle, T. The ant colony optimization metaheuristic: Algorithms, applications, and advances. In *Handbook of Metaheuristics*, F. Glover, G. Kochenberger, and F. S. Hillier, Eds., vol. 57 of *International Series in Operations Research & Management Science*. Springer New York, 2003, pp. 250–285.

[24] Dorigo, M., and Stützle, T. *Ant Colony Optimization.* MIT press, 2004.

[25] Durillo, J. J., and Nebro, A. J. jMetal: A Java framework for multi-objective optimization. *Adv. Eng. Softw. 42*, 10 (Oct. 2011), 760–771.

[26] Edgeworth, F. Y. *Mathematical physics; an essay on the application of mathematics to the moral sciences.* C. Kegan Paul and Co., 1881.

[27] EEA. Occupancy rates of passenger vehicles (term 029). Tech. rep., European Environment Agency, May 2010.

[28] Eklund, P., Thom, J., Wray, T., and Thomson, M. Location privacy in a digital ecosystem for context-aware applications. In *Proceedings of the International Conference on Management of Emergent Digital EcoSystems* (New York, NY, USA, 2010), MEDES '10, ACM, pp. 137–144.

[29] EPA. Inventory of U.S. greenhouse gas emissions and sinks: 1990 2010. Tech. Rep. EPA 430-R-12-001, U.S. Environmental Protection Agency, 1200 Pennsylvania Ave., N.W., 2012.

[30] Eskandari, H., Geiger, C. D., and Lamont, G. B. FastPGA: a dynamic population sizing approach for solving expensive multiobjective optimization problems. In *Proceedings of the 4th international conference on Evolutionary multi-criterion optimization* (Berlin, Heidelberg, 2007), EMO'07, Springer-Verlag, pp. 141–155.

[31] Geisberger, R., Luxen, D., Neubauer, S., Sanders, P., and Völker, L. Fast detour computation for ride sharing. In *Proceedings of ATMOS* (2010), pp. 88–99.

[32] Gen, M., and Cheng, R. *Genetic Algorithms and Engineering Optimization (Engineering Design and Automation).* Wiley-Interscience, 1999.

[33] Ghelawat, S., Radke, K., and Brereton, M. Interaction, privacy and profiling considerations in local mobile social software: a prototype agile ride share system. In *Proceedings of the 22nd Conference of the Computer-Human Interaction Special Interest Group of Australia on Computer-Human Interaction* (New York, NY, USA, 2010), OZCHI '10, ACM, pp. 376–379.

[34] Gidfalvi, G., Larsen, H. R., and Pedersen, T. B. Instant social ridesharing. G. Gidofalvi, G. Herenyi, and T. B. Pedersen, Eds., ITS World.

[35] Goss, S., Aron, S., Deneubourg, J., and Pasteels, J. Self-organized shortcuts in the Argentine ant. *Naturwissenschaften 76*, 12 (1989), 579–581.

[36] Gruebele, P. Interactive system for real time dynamic multi-hop carpooling. Tech. rep., Global Transport Knowledge Partnership, 2008.

[37] Hartwig, S., and Buchmann, M. Empty seats traveling. Tech. rep., Nokia Research Center, Bochum, 2007.

[38] He, F., Qi, H., and Fan, Q. An evolutionary algorithm for the multi-objective shortest path problem. In *Proceedings of the international Conference on Intelligent Systems and Knowledge Engineering* (2007), ISKE 2007.

[39] Herbawi, W., and Weber, M. Ant colony vs. genetic multiobjective route planning in dynamic multi-hop ridesharing. In *Proceedings of the IEEE 23rd International Conference on Tools with Artificial Intelligence* (Washington, DC, USA, 2011), ICTAI '11, IEEE Computer Society, pp. 282–288.

[40] Herbawi, W., and Weber, M. Comparison of multiobjective evolutionary algorithms for solving the multiobjective route planning in dynamic multi-hop ridesharing. In *Proceedings of the IEEE Congress on Evolutionary Computation* (2011), CEC'11, pp. 2099–2106.

[41] Herbawi, W., and Weber, M. The effect of problem modeling on the evolutionary multiobjective route planning in dynamic multi-hop ridesharing. In *Proceedings of the Metaheuristics International Conference* (2011).

[42] Herbawi, W., and Weber, M. Evolutionary multiobjective route planning in dynamic multi-hop ridesharing. In *Proceedings of the 11th European conference on Evolutionary computation in combinatorial optimization* (Berlin, Heidelberg, 2011), EvoCOP'11, Springer-Verlag, pp. 84–95.

[43] Herbawi, W., and Weber, M. A genetic local search algorithm for multiobjective time-dependent route planning. In *Proceedings of the IEEE Congress on Evolutionary Computation* (june 2012), CEC'12, pp. 1–7.

[44] Herbawi, W., and Weber, M. Modeling the multihop ridematching problem with time windows and solving it using genetic algorithms. In *Proceedings of the 2011 IEEE 24th International Conference on Tools with Artificial Intelligence* (Athens, Greece, 2012), ICTAI '12, IEEE Computer Society.

[45] Herbawi, W., and Weber, M. The ridematching problem with time windows in dynamic ridesharing: A model and a genetic algorithm. In *Proceedings of the IEEE Congress on Evolutionary Computation* (june 2012), CEC'12, pp. 1–8.

[46] Herbawi, W. M., and Weber, M. A genetic and insertion heuristic algorithm for solving the dynamic ridematching problem with time windows. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference* (New York, NY, USA, 2012), GECCO '12, ACM, pp. 385–392.

[47] Holland, J. H. *Adaptation in natural and artificial systems.* MIT Press, Cambridge, MA, USA, 1992.

[48] Huang, Q., Fork, D., PlurKowski, L., and Weinerth, D. System and method for setting a rideshare transaction fee, U.S. Patent US 2011/0196709 A1, 2011.

[49] Ishibuchi, H., and Murata, T. Multi-objective genetic local search algorithm and its application to flowshop scheduling. *IEEE Transactions on Systems, Man and Cybernetics 28*, 3 (1998), 392–403.

[50] Jacobson, S., and King, D. Fuel saving and ridesharing in the U.S.: Motivations, limitations, and opportunities. *Transportation Research Part D: Trans. and Environment 14*, 1 (2009), 14–21.

[51] Jaszkiewicz, A. Genetic local search for multi-objective combinatorial optimization. *European Journal of Operational Research 137*, 1 (2002), 50–71.

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

[53] Jorgensen, R. *Dial-a-Ride.* PhD thesis, Technical University of Denmark, 2002.

[54] Kamar, E., and Horvitz, E. Collaboration and shared plans in the open world: studies of ridesharing. In *International Joint Conference on Artifical intelligence (IJCAI), 2009* (San Francisco, CA, USA, 2009), Morgan Kaufmann Publishers Inc., pp. 187–194.

[55] Kanoh, H., and Hara, K. Hybrid genetic algorithm for dynamic multi-objective route planning with predicted traffic in a real-world road network. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation* (New York, NY, USA, 2008), GECCO '08, ACM, pp. 657–664.

[56] Kleiner, A., Nebel, B., and Ziparo, V. A. A mechanism for dynamic ride sharing based on parallel auctions. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence - Volume Volume One* (2011), IJCAI'11, AAAI Press, pp. 266–272.

[57] Knowles, J., and Corne, D. The pareto archived evolution strategy: A new baseline algorithm for multiobjective optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation* (Piscataway, NJ, 1999), CEC'99, IEEE Press, pp. 9–105.

[58] Kuhn, H. W. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly 2*, 1-2 (1955), 83–97.

[59] Li, X.-Y., Sun, Z., Wang, W., and Lou, W. Cost sharing and strategyproof mechanisms for set cover games. *Journal of Combinatorial Optimization 20*, 3 (2010), 259–284.

[60] Luke, S. *Essentials of Metaheuristics*. Lulu, 2009. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[61] Man, K., Tang, K., and Kwong, S. Genetic algorithms: concepts and applications [in engineering design]. *Industrial Electronics, IEEE Transactions on 43*, 5 (1996), 519–534.

[62] Mann, M., Bradley, R., and Hughes, M. Northern hemisphere temperatures during the past millennium: Inferences, uncertainties, and limitations. *Geophysical Research Letters 26* (1999), 759–762.

[63] Martins, E., and Santos, J. The labeling algorithm for the multiobjective shortest path problem. Tech. rep., University of Coimbra, Portugal, 1999.

[64] Mitchell, M. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA, 1998.

[65] Mooney, P., and Winstanley, A. An evolutionary algorithm for multicriteria path optimization problems. *International Journal of Geographical Information Science 20* (2006), 401–423.

[66] Müller-Hannemann, M., and Schnee, M. *Finding All Attractive Train Connections by Multi-criteria Pareto Search*, vol. 4359/2007. Springer Berlin, 2007, pp. 246–263.

[67] Müller-Hannemann, M., and Weihe, K. Pareto shortest paths is often feasible in practice. In *Algorithm Engineering*, G. Brodal, D. Frigioni, and A. Marchetti-Spaccamela, Eds., vol. 2141 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2001, pp. 185–197.

[68] Ooi, C. H., and Tan, P. Genetic algorithms applied to multi-class prediction for the analysis of gene expression data. *Bioinformatics 19*, 1 (2003), 37–44.

[69] Orda, A., and Rom, R. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *J. ACM 37*, 3 (July 1990), 607–625.

[70] Pangilinan, J., and Janssens, G. Evolutionary algorithms for the multiobjective shortest path problem. *International Journal of Applied Science, Engineering and Technology 4(1)* (2007), 205–210.

[71] Pareto, V. cours d'conomie politique, 1896.

[72] Pyrga, E., Schulz, F., Wagner, D., and Zaroliagis, C. Towards realistic modeling of time-table information through the time-dependent approach. *Electronic Notes in Theoretical Computer Science 92* (2004), 85–103. Proceedings of ATMOS Workshop 2003.

[73] Pyrga, E., Schulz, F., Wagner, D., and Zaroliagis, C. D. Experimental Comparison of Shortest Path Approaches for Timetable Information. In *ALENEX/ANALC* (2004), L. Arge, G. F. Italiano, and R. Sedgewick, Eds., SIAM, pp. 88–99.

[74] Radke, K., Brereton, M., Mirisaee, S., Ghelawat, S., Boyd, C., and Nieto, J. G. Tensions in developing a secure collective information practice - the case of agile ridesharing. In *Proceedings of the 13th IFIP TC 13 international conference on Human-computer interaction - Volume Part II* (Berlin, Heidelberg, 2011), INTERACT'11, Springer-Verlag, pp. 524–532.

[75] Rossi-Doria, O., Sampels, M., Birattari, M., Chiarandini, M., Dorigo, M., Gambardella, L., Knowles, J., Manfrin, M., Mastrolilli, M., Paechter, B., Paquete, L., and Stützle, T. A comparison of the performance of different metaheuristics on the timetabling problem. In *Practice and Theory of Automated Timetabling IV*, E. Burke and P. Causmaecker, Eds., vol. 2740 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2003, pp. 329–351.

[76] Schnee, M. *Fully Realistic Multi-Criteria Timetable Information Systems*. PhD thesis, Technische Universitt Darmstadt, 2009.

[77] Schrank, D., and Lomax, T. The 2007 urban mobility report. Tech. rep., Texas Transportation Inst., Washington, 2007.

[78] Schulz, F., Wagner, D., and Weihe, K. Dijkstra's algorithm on-line: an empirical case study from public railroad transport. *J. Exp. Algorithmics 5* (2000).

[79] Selmic Milica, Macura Dragana, T. D. B. Ride matching using k-means method: Case study of gazela bridge in belgrade, serbia. *Journal of Transportation Engineering-ASCE 138*, 1 (2012), 132–140.

[80] Sinnott, R. W. Virtues of the haversine. *Sky and Telescope 68*, 2 (1984), 159.

[81] Skriver, A., and Andersen, K. A label correcting approach for solving bicriterion shortest-path problems. *Computersand Operations Research 27*, 6 (2000), 507–524.

[82] Solomon, M. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research 35* (1987), 254–265.

[83] Sperling, D., and Cannon, J. Climate and transportation solutions: Findings from the 2009 asilomar conference on transportation and energy policy. Tech. rep., Institute .of Transportaion Studies, University of California, 2010.

[84] Srinivas, M., and Patnaik, L. Genetic algorithms: a survey. *Computer 27*, 6 (1994), 17–26.

[85] Stach, C., and Brodt, A. vHike - A Dynamic Ride-Sharing Service for Smartphones. In *Proceedings of the 2011 IEEE 12th International Conference on Mobile Data Management - Volume 01* (Washington, DC, USA, 2011), MDM '11, IEEE Computer Society, pp. 333–336.

[86] Stüzle, T., and Hoos, H. MAX-MIN ant system. *Journal of future generation computer systems 16* (2000), 889–914.

[87] Talbi, E.-G. *Metaheuristics: From Design to Implementation.* Wiley Publishing, 2009.

[88] Tarapata, Z. Selected multicriteria shortest path problems: An analysis of complexity, models and adaptation of standard algorithms. *International Journal of Applied Mathematics and Computer Science 17*, 2 (June 2007), 269–287.

[89] Teodorovi, D., and Orco, M. Mitigating traffic congestion: Solving the ridematching problem by bee colony optimization. *Transportation Planning and Technology 31*, 0308-1060 (2008), 135–152.

[90] Toth, P., and Vigo, D., Eds. *The vehicle routing problem.* Siam, 2002.

[91] Tsai, J. Y., Kelley, P. G., Cranor, L. F., and Sadeh, N. Location-sharing technologies: Privacy risks and controls. In *Proceedings of Research Conference on Communication, Information and Internet Policy (TPRC)* (2009).

[92] Ulungu, E. L., and Teghem, J. Multi-objective combinatorial optimization problems: A survey. *Journal of Multi-Criteria Decision Analysis 3* (1994), 83–104.

[93] Veldhuizen, D. A. V. *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations.* PhD thesis, Air University, 1999.

[94] Xing, X., Warden, T., Nicolai, T., and Herzog, O. *SMIZE: A Spontaneous Ride-Sharing System for Individual Urban Transit*, vol. 5774/2009 of *Lecture Notes in Computer Science.* Springer Berlin / Heidelberg, 2009, ch. Multiagent System Technologies, pp. 165–176.

[95] Zahn, W. Systems and methods for global transportation, vetting, and payment, U.S. Patent US 2010/0153279 A1, 2010.

[96] Zitzler, E., and Knzli, S. Indicator-based selection in multiobjective search. In *Parallel Problem Solving from Nature - PPSN VIII*, X. Yao, E. Burke, J. A. Lozano, J. Smith, J. J. Merelo-Guervs, J. A. Bullinaria, J. Rowe, P. Tino, A. Kabn, and H.-P. Schwefel, Eds., vol. 3242 of *Lecture Notes in Computer Science.* Springer Berlin / Heidelberg, 2004, pp. 832–842.

[97] Zitzler, E., Laumanns, M., and Thiele, L. SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In *Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems* (Athens, Greece, 2001), K. C. Giannakoglou, D. T. Tsahalis, J. Périaux, K. D. Papailiou, and T. Fogarty, Eds., International Center for Numerical Methods in Engineering, pp. 95–100.

[98] Zitzler, E., and Thiele, L. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE TEC 3*, 4 (1999), 257–271.

[99] Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. M., and Grunert da Fonseca, V. Performance Assessment of Multiobjective Optimizers: An Analysis and Review. *IEEE TEC 7*, 2 (2003), 117–132.