A Distributed Approach for Robust, Scalable, and Flexible Dynamic Ridesharing

by

Hadi Hajari

Submitted to the Graduate Faculty of the

School of Computing and Information in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2021

UNIVERSITY OF PITTSBURGH

SCHOOL OF COMPUTING AND INFORMATION

This dissertation was presented

by

Hadi Hajari

It was defended on

May 19, 2021

and approved by

Dr. Paul Munro, Associate Professor, School of Computing and Information, University of Pittsburgh

Dr. Michael Lewis, Professor, School of Computing and Information, University of Pittsburgh

Dr. Zachary Rubinstein, Principal Project Scientist, Robotics Institute, Carnegie Mellon University

Dissertation Director: Dr. Hassan Karimi, Professor, School of Computing and Information, University of Pittsburgh

A Distributed Approach for Robust, Scalable, and Flexible Dynamic Ridesharing

Hadi Hajari, PhD

University of Pittsburgh, 2021

This dissertation provides a solution to dynamic ridesharing problem, a NP-hard optimization problem, where a fleet of vehicles move on a road network and ridesharing requests arrive continuously. The goal is to optimally assign vehicles to requests with the objective of minimizing total travel distance of vehicles and satisfying constraints such as vehicles' capacity and time window for pick-up and drop-off locations. The dominant approach for solving dynamic ridesharing problem is centralized approach that is intractable when size of the problem grows, thus not scalable. To address scalability, a novel agent-based representation of the problem, along with a set of algorithms to solve the problem, is proposed. Besides being scalable, the proposed approach is flexible and, compared to centralized approach, more robust, i.e., vehicle agents can handle changes in the network dynamically (e.g., in case of a vehicle breakdown) without need to re-start the operation, and individual vehicle failure will not affect the process of decision-making, respectively. In the decentralized approach the underlying combinatorial optimization is formulated as a distributed optimization problem and is decomposed into multiple subproblems using spectral graph theory. Each subproblem is formulated as DCOP (Distributed Constraint Optimization Problem) based on a factor graph representation, including a group of cooperative agents that work together to take an optimal (or near-optimal) joint action. Then a min-sum algorithm is used on the factor graph to solve the DCOP. A simulator is implemented to empirically evaluate the proposed approach and benchmark it against two alternative approaches, solutions obtained by ILP (Integer Linear Programming) and a greedy heuristic algorithm. The results show

that the decentralized approach scales well with different number of vehicle agents, capacity of vehicle agents, and number of requests and outperforms: (a) the greedy heuristic algorithm in terms of solution quality and (b) the ILP in terms of execution time.

Table of Contents

List of Tables

List of Figures

Preface

I would like to thank my advisor, Dr. Hassan Karimi, for his support and allowing me to keep doing research on my idea. He always showed great passion, both in in-person meetings and lab meetings, for discussing different aspects of my dissertation.

I also wish to thank the rest of my dissertation committee, Dr. Paul Munro, Dr. Michael Lewis, and Dr. Zachary Rubinstein for their invaluable help, guidance, and constructive feedback. I had several fruitful meetings with the committee to discuss my work in detail, which helped me a lot shape my research.

I am also grateful to have benefited from constant effort of the staff in School of Computing and Information at the University of Pittsburgh, especially Patricia Markham, Corey James, Brandi Belleau, and Wesley Lipschultz.

Lastly, I am indebted to my family who has always supported and encouraged me and has been my guiding light throughout my life.

1.0 Introduction

As urbanization contributes to the well-beings of societies, more people continually move from rural to urban areas. Nonetheless, urbanization comes with many new challenges in cities such as increased demand on transport infrastructure, increased traffic congestion, increased fuel consumption, and increased level of greenhouse gases (Zheng et al., 2014). These challenges are being addressed by advancements in transportation infrastructures, e.g., intelligent transportation systems (ITS), and vehicle technology, in particular electric cars and autonomous vehicles (AVs).

Next generation of transportation is being envisioned in different ways, including but not limited to autonomy and shared mobility. The technology of AVs has been anticipated to perform in a shared mode similar to sharing commutes with ridesharing (Gerte et al., 2018). Shared autonomous vehicles (SAVs) or driverless taxis, as an innovative transportation mode, is among the new visions focused on mobility that would enhance the future of transportation (Hyland & Mahmassani, 2017; Fagnant et al., 2015). Today, ridesharing is growing in popularity because not only does it have a paramount importance in saving fuel consumption, reducing the need for parking, and improving traffic flow (Kelly, 2007; Morency, 2007; Chan & Shaheen, 2012), but it also fills the gap in places within a city where public transportations are not well supported (Ghoseiri et al., 2011). Fagnant and Kockelman (2018) conducted ridesharing simulations and showed that ridesharing can reduce trip costs, reduce vehicle miles traveled (VMT), and improve service time (ride time plus wait time) for SAV users. Another simulation study confirms that SAVs can reduce fleet size, wait time, operations cost, and CO2 emissions in comparison with a non-sharing strategy (Liu et al., 2018). Ridesharing, thus, improves the efficiency of transportation

systems where people are provided with choices that are beneficial at both individual and society levels.

Besides the benefits that ridesharing offers, ridesharing is becoming an attractive mode of transportation by many people primarily due to computing and technological advancements in intelligent phones, ridesharing apps, and social networks. Applications like UberPOOL and Lyft Carpool developed by ridesharing companies, Uber and Lyft, respectively, have recently attracted a large and growing number of customers. Gerte et al. (2018) highlighted the dramatic growth in ridesharing services and demonstrated the widespread adoption of ridesharing as a key principle of the future mobility management.

Conceptually, ridesharing refers to sharing of empty car seats between individuals who have similar itineraries and time schedules in any means of transportation such as truck, van, vehicle, or taxi to split travel costs (e.g., gas, toll, and parking fees) (Furuhata et al., 2013). In this mode of transportation, usually a company (e.g., yellow cab company) owns a fleet of vehicles to provide rides to the customers rather than independent private cars. Basically, there are two types of ridesharing (Bullo et al., 2011): *static* ridesharing in which the demands are known in advance and matching vehicles, riders, and routes scheduling happen *before* the trips start; *dynamic* ridesharing in which the requests arrive continuously and the system needs to match and arrange the trips with available vehicles in real time. Route-planning and scheduling algorithms in dynamic ridesharing are more complex than in static one because the algorithms must be efficient enough to solve the underlying optimization problem in real time while satisfying the constraints of both the new requests and the requests already confirmed by riders. Route-planning algorithms in static ridesharing are not applicable to real-time matching in dynamic ridesharing since vehicles and

demands are highly dynamic (in space and time) in the latter case (Shen et al., 2016). This dissertation focuses on dynamic ridesharing due to the increased demand for it.

The central idea to solve dynamic ridesharing problem is the development of optimization models and efficient algorithms for *optimally* matching vehicles and riders in real time. The problem can be formulated as other classic problems in operations research (OR) such as dynamic dial-a-ride problem (DDARP) that is NP-hard (Baugh et al., 1998). Dynamic ridesharing problem has been extensively studied, but it is still a challenging research topic because existing approaches do not adequately scale up. Figure 1 shows the number of requests submitted by yellow cabs' riders in New York City in different hours in February 2016 (*New York City Yellow Taxi Trip*, 2016). Assuming that the demands within hours of days are uniformly distributed, the range of submitted requests in rush hours is around 280 to 400 requests per minute which shows a high degree of dynamism in the system. This huge number of requests per minute with hundreds or thousands of service vehicles and the real-time nature of decision making for the problem requires *large-scale* optimization that is a challenging task.

Scalability in dynamic ridesharing is still an open topic in OR and has gained popularity in recent years while the enabling technologies are becoming available (Lowalekar et al., 2019; Schwarting et al., 2018; Agatz et al., 2016; Agatz et al., 2012; Nourinejad & Roorda, 2016; Ota et al., 2015; Mallus et al., 2017; D'Orey et al., 2012). Furthermore, (1.1) proves that the scale of the problem escalates exponentially as dimensions of the problem increase. Problem's dimensions are number of requests coming to the system n, number of vehicles in the fleet v and their capacities c. In (1.1) three different scenarios, where number of requests are equal, greater, or less than number of empty seats, are considered.

Besides not being scalable, the existing approaches are not flexible, i.e., they cannot efficiently handle changes when unexpected situations occur (e.g., in case of adding or removing a vehicle). Also, building fault-tolerant centralized ridesharing systems is more challenging and costly that building fault-tolerant decentralized ridesharing systems in that in the former one single decision-maker is responsible for all the tasks in the operation where the latter for a much smaller set of tasks.

From the above observational evidences, this dissertation states the following hypothesis:

Hypothesis: A decentralized approach to ridesharing problem addresses the issues of scalability, flexibility, and robustness in dynamic ridesharing.

Taking the decentralized approach requires the development of new models and algorithms and testing them by using two metrics: solution quality (e.g., minimizing total travel distance of vehicle agents) and running time. With these two metrics, the main features (Section 1.2) of the proposed approach can be quantified. For example, to test the flexibility of the proposed approach, at the time of decision making more active vehicle agents can be added to the fleet to serve remaining requests in the pool without need to solve the underlying optimization problem from scratch, which usually happens in the centralized approach. Similar to adding vehicle agents to the fleet to test the flexibility, some vehicle agents can be removed from the fleet at the time of decision making and see how it does affect the solution quality and running time. The benchmark (exact solution) for evaluating solution quality in both centralized and decentralized approaches can be obtained by solving the optimization problem using integer linear programming (ILP).

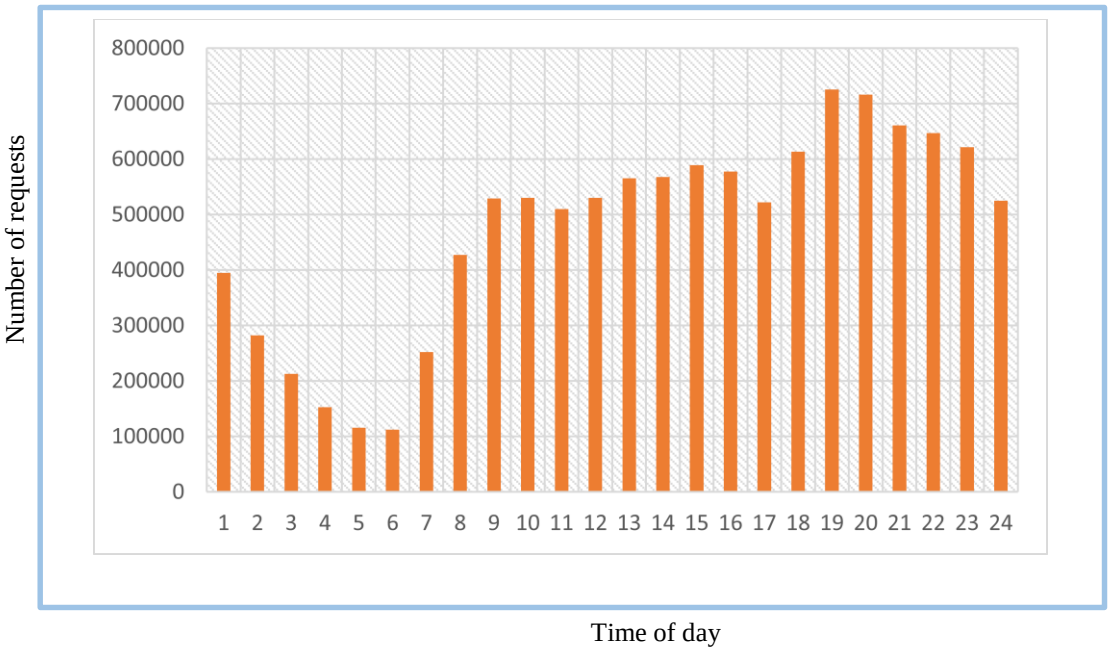


Figure 1 New York City yellow cab taxi records in different hours in February 2016

$$f(n,v,c) =$$

$$\begin{cases} \binom{n}{c}\binom{n-c}{c}\binom{n-2c}{c}\dots\binom{c}{c} = \frac{n!}{c!\times v} \sim O(n^{n-c}), & if\ \frac{n}{c} = v \\ \binom{n}{c}\binom{n-c}{c}\dots\binom{n-(v-1)c}{c} = \frac{n!}{c!\times v\times(n-vc)!} \sim O(n^{vc}),\ if\ \frac{n}{c} > v \\ \binom{v}{m}\binom{n}{c}\binom{n-c}{c}\dots\binom{c}{c} = \frac{v!}{m!\times(v-m)!} \times \frac{n!}{c!\times m} \sim O(v^m n^{n-c}),\ if\ \frac{n}{c} = m < v \end{cases} \quad (1.01)$$

1.1 Terminologies

In this section, I define the terms that I will frequently use in this dissertation.

- *Intelligent vehicle* or *decision maker* is an object that has computing power to do computation independently (e.g., computing bids) and make decisions independently or jointly through interaction with other agents.
- *Capacity* of a vehicle agent is the maximum number of passengers that a vehicle agent can serve.
- *Centralized* or *single-agent* approach refers to a system that only has one decision maker at the heart of the system to complete the given task.
- *Decentralized, distributed,* or *multi-agent* approach refers to a system that consists of multiple decision makers to complete the given task.
- *Customer*, *requester*, *user*, or *rider* is a person who needs to get a ride through submitting a request to the system.

1.2 Motivation

Based on the shortcomings of existing dynamic ridesharing systems and the observational evidence outlined above, this dissertation introduces a novel paradigm in solving dynamic ridesharing problem where a team of agents (decision makers) will be coordinated effectively to service incoming requests. Since the nature of ridesharing problem is dynamic in terms of spatio-temporal location of requests and vehicle agents' trajectories, the features of the proposed approach can properly address the dynamic conditions. These features are:

- Scalability. This dissertation hypothesizes that decentralized ridesharing systems are scalable because multiple decision makers can solve a given task more efficiently than a single agent can when the problem scales up.

- Flexibility. This dissertation hypothesizes that decentralized ridesharing systems are flexible in that decision makers can be added to or removed from the systems dynamically without interruption in the operation. In a centralized approach, adding/removing agents requires a re-start of the system.
- Robustness. This dissertation hypothesizes that decentralized ridesharing systems are more robust than centralized ones. In decentralized ridesharing systems, failure of one single decision maker affects the operation locally and not the overall operation. Building fault-tolerant decentralized ridesharing systems is less challenging and costly than building centralized ridesharing systems; in case of failure, the centralized approach must handle several computational challenges, incurs time delay due to synchronization between nodes, and requires data integrity checks due to redundant and inconsistent data.
- Heterogeneity. Vehicle agents in the multi-agent approach can be heterogeneous with respect to capacity. This heterogeneity does not affect the overall performance of the system unlike the single agent approach in which increased capacity, as one dimension of the problem, would degrade the performance.

Besides the benefits that the multi-agent approach can bring in solving ridesharing problem, it has its own challenges as follows.

- Design of distributed algorithms to effectively solve the ridesharing problem in hand.
- Coordination between vehicle agents in a decentralized manner is a challenge since there is no central controller in the system. Building a graph decomposition technique, considering the density and size of the graph is important. The provided solution must be supported theoretically and scales well as the number of vehicle agents increases.

- Development of an efficient algorithm that guarantees convergence for real-time decision making in each partition.

1.3 Proposed Research

Disruptive technologies in IoT and automotive industry, particularly autonomous vehicles, is the motivation behind proposing a multi-agent approach to solve inherently distributed ridesharing problem. In this respect, this dissertation formulates this spatio-temporal resource allocation problem as a decentralized optimization and design distributed algorithms to obtain optimal or at least near-optimal solutions. This dissertation takes a decentralized optimization approach to address dynamic ridesharing problem where the global objective function is to minimize the total travel distance of the vehicle agents. In the proposed approach, the global objective function is decomposed into a sum of local loss functions where each is only known to one particular vehicle agent. The local constraints of each local objective function are: capacity of each vehicle agent and time windows for each request's pick-up and drop-off locations. It is assumed that in the proposed decentralized approach, the agents have computing resources and can work cooperatively to minimize total travel distance.

There are three different types of agents in the decentralized approach: users, dispatcher, and decision makers. Users send their requests to the dispatcher and the dispatcher passes all the requests to the decision makers. Each decision maker individually computes bid for the requests that they can serve, considering capacity constraint and time windows for existing and new requests. Once the dispatcher receives the bids from decision makers, it constructs the coordination graph between decision makers based on their similarity and decomposes the graph into several

partitions. Then the dispatcher allocates requests to each partition and sends this information to the decision makers in the corresponding partitions. In each partition, decision makers work together jointly to decide which requests should be served by which decision maker. The final assignment in each partition will be issued to the dispatcher and the users will be notified if their requests are accepted or rejected. More detail of the proposed approach along with the set of algorithms will be discussed in Chapter 4.0.

1.4 Contributions

This dissertation addresses the proposed hypothesis (Section 1) by introducing a novel perspective for solving ridesharing problem using decentralized approach with the following contributions:

- An objective function that is suitable for building a distributed optimization model
- An algorithm for decomposing the coordination graph for tasks distribution and coordination between vehicle agents
- An optimization model to allocate requests to each subproblem
- Formulation of subproblems using factor graph
- Design and development of an algorithm to do inference in the factor graph in order to solve each subproblem
- Implementation of a simulation to test the proposed approach

1.5 Structure of the Dissertation

This dissertation presents a novel paradigm to dynamic ridesharing problem where the optimization model along with the algorithms make the ridesharing system scalable, robust, and flexible. Chapter 2.0 explains the centralized ridesharing as it is the dominant approach in the literature, followed by providing a comprehensive background to dynamic ridesharing problem from perspectives of transportation, robotics and operations research in Chapter 3.0. Chapter 4.0 describes the proposed optimization model and the algorithms to solve dynamic ridesharing problem. Chapter 5.0 presents and analyzes the experimental results to show the workability of the proposed approach along with the limitations of this work. Finally, this dissertation concludes in Chapter 6.0 with a summary of this research towards a description of future work.

2.0 Background

Since the centralized approach is the dominant paradigm for solving dynamic ridesharing problem in the literature, Section 2.1 explains its main components, i.e., Systems Objects, Data, Algorithm, and Optimization. Then Section 2.2 analyzes different characteristics of the centralized approach along with its advantages and disadvantages in dealing with a large-scale ridesharing problem.

2.1 Centralized Approach

The centralized approach has a *single-agent* decision maker at the heart of the system and contains these modules: Systems Objects module (Section 2.1.1), Data module (Section 2.1.2), Algorithm module (Section 2.1.3), and Optimization module (Section 2.1.4). Figure 2 illustrates a high-level view of the centralized approach and how its modules interact with each other in a dynamic ridesharing system. In this approach, each vehicle is connected to a common cloud via the Internet or an intranet to take advantage of the high-performance computing and large-storage capacity of the cloud. The cloud hosts all the required resources with no duplication, including the street network database and the program code for optimization, among other functionalities, and is responsible for all computations. As shown in Figure 2, a rider upon submitting a ridesharing request to the system receives a response from the system (decision maker) after schedules and routes for corresponding vehicles are updated. The response to the user comprises the vehicle ID and the estimated pick-up time or a reject response in case the system could not match a vehicle

to the request. We analyze the features of the centralized approach and discuss its shortcomings in the remainder of this chapter.

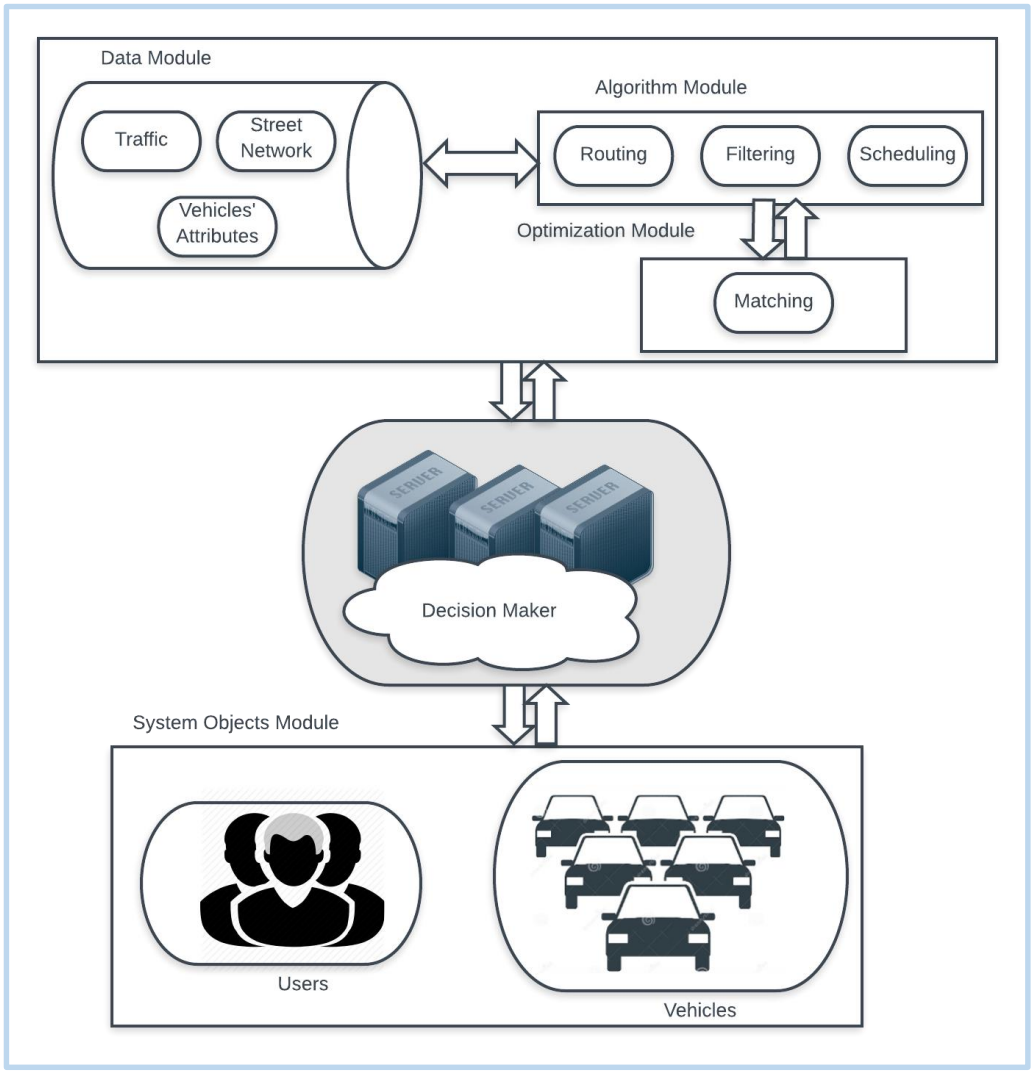


Figure 2 Centralized approach in dynamic ridesharing systems

2.1.1 System Objects Module

In a ridesharing system, there are two sets of objects: users and vehicles. The users use a mobile or web application to submit requests in real time to the system, consisting of pick-up location, drop-off location, number of passengers, time window for pick-up location that defines the time interval when the user should be picked up at the origin and time window for drop-off location that defines the time interval when the user should be dropped off at the destination. The last two items specify the constraints of each request which are required to be met in solving ridesharing problem. Moreover, dynamic ridesharing problem must take into account the number of vehicles servicing over a street network (finding a vehicle for each request) by dispatching vehicles with the purpose of minimizing or maximizing an objective function and satisfying a set of constraints (Section 2.1.4 is devoted to the discussion of objective functions and constraints). It is important to note that the fleet of vehicles can be heterogeneous in terms of having different capacities and/or accessibility for various needs such as individuals using wheelchairs requesting rides. The vehicles upload their time-stamped locations to the system frequently so that the decision maker knows where each vehicle is during a period of time for finding matches between vehicles and riders.

2.1.2 Data Module

The database in the Data module contains all the required data for making ridesharing decisions and includes a street network (represented as a graph) for finding routes, traffic data for handling stochasticity about travel times in the street network, and other static and dynamic data about each vehicle such as ID (static), capacity (static), current location and time (dynamic),

updated schedule and route for new requests (dynamic), and number of empty seats (dynamic). The Algorithm module utilizes the data in this database to perform various functions such as finding optimal paths over the network, selecting candidate vehicles, and updating vehicles' schedules once new requests are submitted.

Travel time is essential for routing and in the absence of traffic information less reliable routes may be found. However, while incorporating real-time traffic information leads to finding more optimal routes, real-time routing is computationally expensive (Ma et al., 2013), which is why most works in the literature take a pre-computed shortest path approach to overcome the time issue.

2.1.3 Algorithm Module

This module encompasses three submodules: routing, filtering, and scheduling. The routing submodule is responsible for computing the optimal travel time between pairs of locations on the street network. Applying existing real-time techniques, e.g., the technique by Delling et al. (2009), can efficiently compute the shortest path in real time and applying non-real-time existing techniques, e.g., the proposed method in T-Share (Ma et al., 2013), can approximate the distance of the shortest path by partitioning the street network into grid cells and determining the shortest path for each anchor node (the nearest node in the road network to the geographical center of the cell) pair; however, the distance accuracy in this technique highly depends on the selected grid size.

The aim of filtering submodule is to efficiently select a set of candidate vehicles that can serve new requests, satisfying the constraints of each candidate vehicle's capacity and time windows for pick-up and drop-off locations. Obviously, going through each vehicle locally to

match a ride request when there are many vehicles is computationally inefficient. To address this inefficiency issue, spatial data structures such as R-tree (Guttman, 1984), KD-tree (Jon Louis Bentley, 1990), R^+-tree (Sellis et al., 1987), R^*-tree (Kriegel et al., 1990), and Quad-tree (Bentley & Finkel, 1974) have been considered. However, these data structures may not be suitable for a large-scale dynamic ridesharing problem because of high cost of handling dynamism associated with vehicles and updating indices (Xia & Prabhakar, 2003; Lee et al., 2003). Research, e.g., see Ma et al. (2013) and Shen et al. (2016), focused on developing speed-up techniques to reduce the search space of the problem by pruning the potential vehicles to pick up a ride request. In such techniques, the street network is partitioned into a set of grid cells in which each cell g_i stores the following static and dynamic data: a list of other ascending temporally-ordered grid cells traveled to g_i (static); a list of other ascending spatially-ordered grid cells traveled to g_i (static); and a list of all vehicles scheduled to enter g_i in the next few hours (dynamic).

After reducing the search space by the filtering submodule and choosing a set of candidate vehicles, to reschedule the route of each candidate vehicle and check the contribution of each vehicle to the objective function, satisfying the constraints (e.g., time windows for pick-up and drop-off locations) of both the new request and the existing rides, the scheduling submodule is invoked. A ride in dynamic ridesharing must start from the pick-up location and move to the drop-off location and the shortest path between each pair of pick-up and drop-off locations is computed by the routing submodule. Figure 3 illustrates the functionality of the scheduling submodule in dynamic ridesharing. In panel (a), a vehicle has a schedule to pick up a passenger C_1 at P_1, drop C_1 at D_1, pick up passenger C_2 at P_2 and drop C_2 at D_2. When a new request with origin P_3 and destination D_3 arrives, rescheduling is needed. Panel (b) shows the result of rescheduling, a new route and the sequence of serving the passengers.

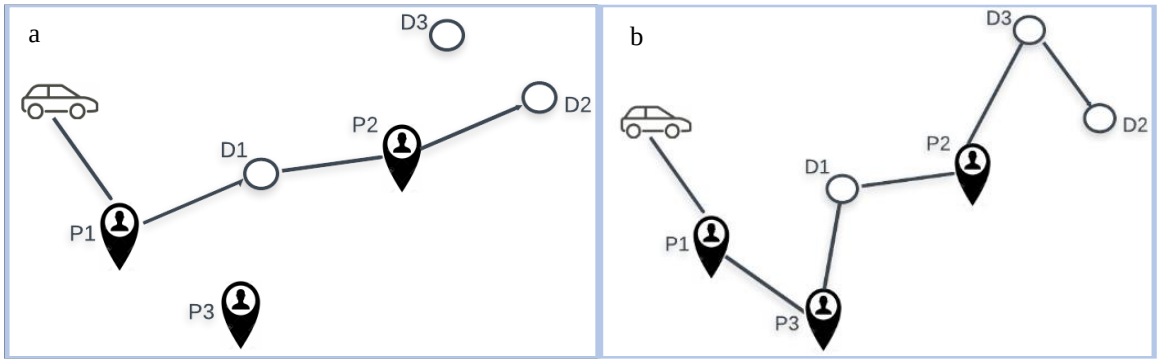


Figure 3 An illustration of rescheduling in a dynamic ridesharing system. (a) current route of a vehicle for serving two passengers and a new request with pick-up location P_3 and drop-off location D_3. (b) new route of the vehicle after rescheduling.

The rescheduling problem can be addressed by one of two methods: (i) insert the new request at any position in the current schedule without altering the order of existing locations; this is known as insertion heuristic method. To insert a new request into the current route with n stops (pick-up and drop-off locations), there are (n+1) (n+2)/2 schedule alternatives. This method is widely used in the literature (e.g., Huang et al., 2013; Coslovich et al., 2006; Jaw et al., 1986) because it is not computationally expensive. (ii) construct an entirely new schedule and solve an instance of the open-loop TSP (Traveling Salesman Problem) with time window that is computationally expensive (with complexity of O(n!)). For vehicles with lower capacity, the problem can be formulated as ILP (Reinelt, 1994) and solved by applying an exhaustive search using a branch-and-bound technique (Kalantari et al., 1985). Most studies have addressed the problem of vehicles with large capacity by developing approximation and heuristic techniques (e.g., insertion heuristics) to provide real-time solutions at the cost of optimality. Example heuristic solutions are Christofides (Christofides & Eilon, 1969), Lin-Kernighan (Lin & Kernighan, 1973),

modified version of Lin-Kernighan (Helsgaun, 2000), tabu search (Tsubakitani & Evans, 1998), or Simulated Annealing (Song et al., 2003).

2.1.4 Optimization Module

The optimization module finds an optimal solution to the problem by identifying the candidate vehicle, among all candidates that the scheduling submodule evaluated for contribution to the objective function, which best meets the request. The optimization module is the core of ridesharing solution by performing a matching function that assigns vehicles to the requests with the aim of optimizing an objective function. The ridesharing models consider one or a weighted combination of the following cost functions when determining rideshare matches (Agatz et al., 2012): minimizing total distances or travel times by all vehicles over the street network; minimizing the vehicles' detours; minimizing cost to the passengers; and maximizing number of successful rideshare requests. These objective functions take a variety of different constraints such as capacity of vehicles, desired departure or drop-off time specified by users, or travel cost.

The assignment task depends on how the optimization problem deals with the incoming requests. Basically, there are two research direction in addressing the assignment problem: queueing (first-come-first-served) and batch assignment. In queueing approach, popular for solving assignment problems, all trip requests are considered in chronological order. Note that this is a greedy strategy and may not provide a global optimal solution since the approach does not consider all the possible combinations of trip requests to be shared. By applying the queuing approach in Figure 4 where request 1 arrives before request 2 and vehicles cannot serve both requests simultaneously, the optimization module assigns vehicle 1 to request 1 with cost of 20 and vehicle 2 to request 2 with cost of 90, providing an overall cost of 110. According to (Ayala

et al., 2017), this type of vehicle-request assignment fits Nash equilibrium (Nash, 1950). In the example in Figure 4, assigning request 1 to vehicle 1 is the best option for vehicle 1 since its travel time to request 2 is longer and vehicle 2 cannot get request 1 to decrease its travel time because vehicle 1 is closer to request 1. Here, the vehicles are acting selfishly for their own benefits instead of trying to optimize the overall performance. While matching vehicles to requests with the queuing approach is efficient in solving the underlying combinatorial optimization problem, it does so at the cost of optimality.

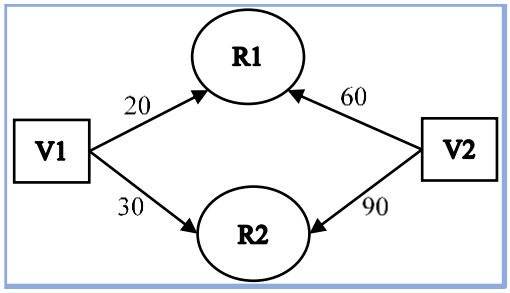


Figure 4 An example of queueing approach in ridesharing problem. The numbers on links are travel times for vehicles-requests (Ayala et al., 2018).

As illustrated in Figure 4, the Nash equilibrium assignment is not optimal because a better solution with total cost of 90 can be obtained by matching vehicle 2 to request 1 and vehicle 1 to request 2. This optimum plan is achievable only if the incoming requests are pooled together and then assigned to all vehicles in a given interval. In the batch assignment approach (Figure 5), given a set of requests and a set of vehicles with many-to-many relationship, the goal is to compute the optimal assignment of requests to vehicles that minimizes or maximizes the objective function. Note that each vehicle might be able to service a set of requests together, so more nodes will be added to the bipartite graph represented in Figure 5. Obtaining close-to-optimal solution in a large-

scale dynamic ridesharing problem is the main motivation behind using the batch assignment approach instead of the queuing approach, though the batch assignment approach is an NP-hard problem. This means that a solution to this large-scale combinatorial optimization problem with reasonable response time can be provided by heuristics and/or approximate approaches (e.g., Liebling, 1987).

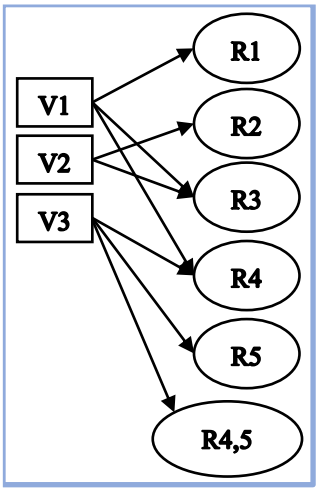


Figure 5 An example of batch assignment

2.2 Analysis

As discussed earlier, in the centralized approach it is assumed that the data module has the most updated information about vehicles, requiring vehicles to share their status, i.e., IDs and spatial locations, with the central system frequently (e.g., every 10 seconds), which is usually costly. This sharing information in the centralized approach can be seen as a trade-off between cost of sharing and loss of efficiency when information is not shared. This information is crucial

for the algorithm module to process incoming requests efficiently and select candidate vehicles that are suitable for serving them.

Like all real-time systems, ridesharing systems must be designed to incur zero downtime, i.e., they must be fault-tolerant. In a centralized ridesharing system, if the *single-agent* decision maker fails, the entire system becomes unavailable interrupting the ridesharing service. Building fault-tolerant ridesharing systems is not trivial as providing extra hardware and multiple versions of the same modules, a common approach in building fault-tolerant systems, leads to time delay due to synchronization between nodes and data integrity checks due to multiple copies of data and redundant data.

The two key questions concerning the time performance of the optimization module in the centralized approach are: (a) Are the algorithms scalable? and (b) Does adding computing resources address the scalability issue? Assuming each vehicle is equipped with sufficient computing power, parallel computation is possible in a way that each vehicle can locally solve the problem in the Scheduling submodule. Nevertheless, dedicating a VM (Virtual Machine) to each vehicle in the cloud makes this assumption unrealistic due to existence of thousands of vehicles in the fleet, which continues to increase steadily in rush hours and may cause the cloud infrastructure to reach its limits. On the other hand, in off-hours when the demand for vehicles is low, many of these VMs with dedicated CPU, memory, and storage would be idle. Obviously, this is not a cost-effective way of utilizing resources in the cloud where detecting these idle VMs for recycling their resources and dynamic VM allocation (Saraswathi et al., 2015; Xiao et al., 2013) will add complexity to the underlying optimization problem.

As a s*ingle-agent* decision maker at the heart of the system, finding a near global optimal solution to the underlying optimization problem is the only option. The optimization problem in

the batch assignment approach can be easily intractable as the input size of the problem, such as adding more requests to the pool or adding vehicles to the fleet, for example, in rush hours, increases. One way to address the scalability issue with a reasonable time performance is to formulate the problem as an ILP and take a parallel computing approach. There are some studies (e.g., Ralphs, 2006; Barreto & Bauer, 2010) that address parallelization of branch-and-bound algorithm for solving ILP. There also exist state-of-the-art ILP solvers such as Mosek (*Mosek optimization solver*, 2019), CPLEX (*CPLEX optimizer*, 2019), or Gurobi (*Gurobi optimizer*, 2019), which can be used as a black box and implemented in parallel computing environments. However, the effect of task decomposition and inter-processor coordination in the design of scalable parallel branch-and-abound algorithms and efficient usage of additional processors may be a barrier to acceptance of this approach in solving the large-scale optimization problem in real time (Bader et al., 2005; Herrera et al., 2017).

3.0 Dynamic Ridesharing

It is now informative to examine in detail existing solution approaches in the literature that can be applied to dynamic ridesharing problem. There are different classes of solution approaches to the vehicle routing problem, including dynamic vehicle routing problem with time window (DVRPTW), dynamic pick-up and delivery problem (DPDP), dynamic dial-a-ride problem (DDARP), and multiple travelling salesman problem with time window (MTSPTW). Table 1 makes an analogy between different classes of VRP and dynamic ridesharing problem. Among these classes of VRP, the solution to class DDARP can be applied to dynamic ridesharing problem due to the similarities in the mathematical formulation of the problem and the constraints which are: a) vehicles with finite capacities; b) passenger's request with a pair of pick-up and drop-off locations; and c) passenger's request with time windows for pick-up and drop-off locations. In the DVRPTW, the vehicles' routes start and end at a depot where the concept of pick-up and delivery point is released (Pillac et al., 2013). The MTSPTW attempts at finding a set of optimal vehicles' routes within a specific time window. In this type of problem, the concept of dynamism does not exist, the capacity constraint of vehicles is released, and similar to DVRPTW, paired pick-up and drop-off locations' constraint is released (Krishnamurti, 2002). DPDP's formulation is suitable for the problems where the requests are placed for objects transportation such as parcels or letters, time windows are not tight when they are present, and there are no capacity constraints (Berbeglia et al., 2010). Basically, in the DDARP, as a special case of DPDP, vehicles start/end their routes from/at different depots (or single depot), users' requests are broadcast during the operation, and users need to be transported between pairs of pick-up and drop-off locations with the aim of finding a set of optimal routes for vehicles that minimize distance with constraints such as time windows

for each request (Parragh et al., 2008). In the DDARP, vehicles can arrive at pick-up locations before the beginning of the time window but not later than the end of the time window. By an analogy, this is also valid for the pick-up and drop-off time windows constraint in dynamic ridesharing problem. Due to these close relationships between dynamic ridesharing problem and the DDARP, this chapter summarizes that part of the literature that addresses the DDARP. In the remainder of this chapter, the representative papers that propose novel mechanisms for solving dynamic ridesharing problem are discussed.

Table 1 Analogy between dynamic ridesharing problem and different VRP variants.

Problem	Dynamism	Vehicle's capacity	Paired pick-up and drop-off	Time window
Dynamic Ridesharing	✓	✓	✓	✓
DVRPTW	✓	✓		✓
MTSPTW		✓		✓
DPDP	✓		✓	
DDARP	✓	✓	✓	✓

3.1 Dynamic Dial-a-Ride Problem (DDARP)

The era of big data, technological advances in communication channels, and rapid rise in computing power have shifted the line of DARP research toward dynamic DARP to address various aspects of real-time planning and decision-making process such as developing efficient

on-line optimization algorithms and improvement of solution qualities. The static version of DARP has been extensively studied in the literature, starting with the work of Jaw et al. (1986). However, only a small body of research in the literature is focused on developing solutions to dynamic DARP with time windows. Similar to other variants of VRP, DDARP is a complex combinatorial optimization problem and finding an optimal solution to it is a NP-hard problem. Current solution approaches range from exact methods such as branch-and-cut algorithm to custom-designed heuristics and meta heuristics such as insertion heuristics and tabu search. The main issue with exact algorithms is that they are computationally expensive even with small size instances of the problem which makes it inapplicable in a dynamic environment with real-time decision making. Having focused the attention on the solution approaches to the dynamic and deterministic DARP, size of the problem at hand, efficiency and applicability of the algorithms in real-world scenarios, the intent is not to provide a comprehensive survey of existing solutions to different variants of DARP (static or dynamic, stochastic or deterministic) and interested readers are referred to the surveys by (Cordeau & Laporte, 2007; Molenbruch et al., 2017; Ho et al., 2018).

Table 2 outlines the detailed classification scheme of all approaches as presented in this dissertation.

Table 2 Classification of the papers with respect to the type of solution approach applied

3.1. **DDARP**
Teodorovic and Radivojevic (2000), Attanasio et al. (2004), Berbeglia et al. (2012), Santos and Xavier (2015), Rubinstein et al. (2012)
3.2. **Agent-based Approach**
Fischer et al. (1996), Perugini et al. (2003), Mes et al. (2007), Kleiner et al. (2011), Nourinejad and Roorda (2016), Coltin & Veloso (2014), Asghari et al. (2016)
3.3. **Other Ridesharing Systems**
Cao et al. (2015), Alarabi et al. (2016), Cici et al. (2015), Agatz et al. (2011), Huang et al. (2013), Schreieck et al. (2016), Gao et al. (2017), Jia et al. (2017), Jung et al. (2016), Ma et al. (2013), Ma et al. (2015), Shen et al. (2016), Shemshadi et al. (2014), Hosni et al. (2014), Najmi et al. (2017), Ota et al. (2015), Mallus et al. (2017), Alonso-mora et al. (2018), Simonetto et al. (2019), Lowalekar et al., 2019

Teodorovic and Radivojevic (2000) developed a fuzzy logic approach to solve the dynamic version of the DARP in which all calculations related to the vehicles and passengers waiting times were performed using fuzzy arithmetic. They proposed nine fuzzy rules based on additional vehicles distance, vehicles waiting time, and dispatcher's preference to insert and assign a new request to one of the vehicles' routes with the greatest value of dispatcher's preference strength. They tested the developed algorithms on several numerical examples with generating 900 requests in the 5 a.m.-11 p.m. interval with a fleet of 30 vehicles and capacity of 10. Coslovich et al. (2006) developed a two-phase heuristic algorithm based on route perturbations to insert a new request into the previously planned route of a vehicle. The new requests follow a priority first-come-first-

served rule. In the first phase (off-line phase), the algorithm creates a feasible neighborhood using 2-opt arc swap. Given a feasible neighborhood of the current solution, the second phase (on-line phase) tries to insert the new request into the current route with the purpose of minimizing the level of dissatisfaction (excess ride time). In their DDARP, there are a predefined set of pick-up and delivery stops and capacity constraint of vehicles is not considered. They tested their algorithms on different instances of the problem where at most 50 new requests were presented during the execution.

Attanasio et al. (2004) proposed several parallel implementations of tabu search heuristic to DDARP with the aim of decreasing the running time of the sequential tabu search algorithm. In their formulation, requests are served in a first-come-first-served fashion and the objective function is to accept as many new requests as possible while satisfying time windows and vehicles' capacity constraints. They tested their approach on synthetic and real data in which around 150 requests were submitted dynamically during the operation. Also, a hybrid algorithm, combining a tabu search heuristic and an exact constraint programming algorithm, was developed in (Berbeglia et al., 2012) to solve the DDARP with the same objective as Attanasio et al. (2004). They tested their algorithm on different sets of synthetic and real-life instances, considering 13 vehicles with maximum capacity of eight and up to 200 requests. Santos and Xavier (2015) incorporated pricing decisions in their objective function besides maximizing the number of served requests when modeling static and dynamic version of the DARP as a combinatorial optimization problem. They proposed a heuristic method called GRASP (greedy randomized adaptive search procedure) with path relinking to solve the underlying optimization problem in a large scale. One issue with this method is that if there are already customers onboard a vehicle, the new request can only be added after the drop-off point of the last passenger. This process is necessary to keep the price invariant

for the existing passengers. They ran a simulation to evaluate the performance of the proposed method on a real data set for the city of São Paulo. In the dynamic version of the problem, 1,333 new vehicles per hour with maximum capacity of four and 54 new requests per minute were considered.

Finally, from a robotics perspective, Rubinstein et al. (2012) solved the dynamic oversubscribed DARP with the aim of minimizing vehicle's travel distance by presenting an iterative repair-based search technique called Generalized Task Swap (GTS) for incrementally integrating new requests into the existing schedules. In the oversubscribed DARP, the service quality constraints (e.g., time window) are relaxed when not all requests can be serviced within their constraints. They conducted several experiments on both synthetic benchmark problem and real-life paratransit scheduling problem to evaluate the performance of GTS. In the synthetic data set, all the instances had at most 96 requests with vehicle capacity of 6 where the maximum CPU time for GTS was around 10 seconds. The real-world problem was at a larger scale consisting of 30 to 50 heterogeneous vehicles with maximum capacity of 14 servicing up to 900 requests for a day.

The aforementioned approaches can be used to find solutions to small and medium size instances of the DDARP in a reasonable amount of time; however, there are some areas for further examination. First, the applicability of the proposed techniques to deal with large-scale optimization problem with a high degree of dynamism, explained earlier, need to be investigated. Second, in all cases, a queuing approach has been used in serving the new requests; see Section 2.1.4 for a discussion of the pros and cons of this approach. Third, the quality of solutions in these approaches is not known, i.e., how far the objective value of a solution is from the optimal value.

3.2 Agent-based Approach

The literature includes several papers that have taken a multi-agent approach in transportation scheduling problem where each vehicle as an agent can schedule its own route, calculate the cost needed to serve a new request, and propose an offer (bid) to a centralized server (auctioneer) who eventually makes the decision and assigns the new request to a specific vehicle. This process can be seen as a market-like negotiation mechanism (auction) in which the auctioneer auctions a new request to all the vehicles to see if any of them is able to add it to its schedule. Each vehicle then checks if the new request can be fit into its schedule, considering the constraints of the problem (time windows and capacity). If so, each vehicle computes its bid (the additional cost of adding the new request into its current schedule) and sends the bid to the auctioneer. Then the auctioneer allocates the new request to the vehicle with the smallest bid, if any. A summary of some multi-agent approaches, where agents play a game and are competitive and self-interested without considering any cooperation between themselves in solving the problem, is given below.

The Contract Net Protocol (CNP) (Smith, 1980), one of the oldest task-sharing protocols in a distributed system, has been used in several studies to cope with dynamic scheduling problem in the transportation application. Fischer et al. (1996) and Perugini et al. (2003) conducted two of the earliest studies in the transportation domain by tackling the dynamic scheduling problem as a multi-agent system and developed an extension of CNP to deal with task decomposition and task allocation in a shipping company system.

Besides CNP, auction-based methods have been widely adopted for use in robotics for task and resource allocation to robots in dynamic environments. Second-price sealed-bid auction (a.k.a. Vickrey auction), the most commonly used single-item auction type among the other three (English auction, Dutch auction, sealed first-price auction), provides a mechanism for allocating a

single item to one of the bidders in a multi-agent system. Bidders simultaneously submit their bid for the item in a sealed envelope (unaware of the others' bids) to win the item. Interested readers are referred to the book by Cramton et al. (2006) for a detailed discussion about the topic.

Mes et al. (2007) developed a multi-agent system using the auction mechanism to address the real-time scheduling problem of allocating trucks to dynamic transportation orders with time windows. They implemented both insertion heuristic and exact algorithms based on TSP as the internal scheduling for each vehicle and used Vickrey auction mechanism to assign an incoming order to a vehicle. The work by Kleiner et al. (2011) is the first to present a multi-agent system for solving dynamic ridesharing problem using Vickrey auction mechanism. In this system, the requests are considered sequentially where only one rider can share a ride with a single driver with the goal of minimizing the total travel distance of vehicles and maximizing the number of ride-matches. They simulated an environment with up to 50 vehicles and 50 customers to test the performance of the system. They computed the optimal solution of the problem by solving maximum weighted bipartite matching problem and used it as a baseline to validate the system. Experimental results showed that the outcome is very close to the optimal solution; however, the work lacks a large-scale experiment to prove the performance of the proposed technique. Nourinejad and Roorda (2016) proposed a decentralized matching model based on agents to reduce extensive computation time and provide a near-optimal solution in a single-driver single-passenger dynamic ridesharing problem. They partitioned the space based on the geographic locations of the participants (riders and drivers). They used Vickrey auction mechanism to assign a driver to the new request and validated the efficiency of the proposed method by the solution obtained from integer programming.

Coltin & Veloso (2014) formulated dynamic ridesharing problem differently with the objective of minimizing total travel distance and transfer cost in which passengers are able to transfer between multiple vehicles, meaning that a driver delivers a passenger to a transfer point and then another vehicle picks the passenger up. They developed three algorithms with different quality and running time to solve the problem: an auction-based algorithm, a greedy heuristic algorithm, and a graph-based search algorithm. They tested the performance of the algorithms on synthetic and real datasets with up to 80 vehicles and 100 passengers, and the result showed that the greedy algorithm outperforms the others.

Asghari et al. (2016) proposed a distributed auction-based framework, called APART (Auction-based Price-Aware Real-time), to solve real-time ridesharing problem with the objective of maximizing the drivers' revenue and satisfying monetary constraints of passengers. In APART, a server plays the role of central auctioneer where bidders (drivers) and goods (requests) participate in a sealed-bid auction, i.e., the drivers submit their bids to the server simultaneously and no driver is aware of other drivers' bids. Finally, the server chooses the bidder with the highest bid as winner and matches the new request with the corresponding driver. The bidding process uses a queue-based method in which the bidding process will be performed once a new request comes in. They tested their framework on New York City taxi dataset with thousands of drivers and hundreds of tasks per second.

As a multi-agent approach, the auction strategy is easy to implement and parallelizable among the vehicles to compute and place their bids. Moreover, in terms of response time of providing service to a new request and scalability with adding drivers, the auction-based approach is effective in dynamic environments. However, in terms of solution quality, the auction strategy

does not guarantee an optimal allocation in competitive environments as long as the incoming requests are served based on first-come-first-served fashion (cf. Section 2.1.4).

3.3 Other Ridesharing Systems

This section introduces state-of-the-art ridesharing systems developed in the last decade and focuses on the techniques each system uses and their scalability. Although several techniques concerning different aspects of the ridesharing problem, such as various algorithms, constraints and objective functions, have been proposed, this strand of research is still in its infancy to create a balance between optimality and tractability. The purpose of this section is to shed sufficient light on the current solution approaches to the large-scale dynamic ridesharing problem.

In a simplified version of a dynamic ridesharing system, two papers by Cao et al. (2015) and Alarabi et al. (2016) presented SHAREK as a scalable ridesharing service for matching a rider to a specific driver who can satisfy the constraints of maximum price and maximum wait time for pick-up. SHAREK processes the incoming requests one-by-one in a temporal sequence and assumes that each driver has known origins and destinations and can share a ride with only one passenger. The drivers do not continually drive on the road and are signed off from the system once they reach their destinations. An environment over the area of San Francisco was simulated by producing a synthetic dataset with 10,000 drivers and 1,000 requests to evaluate the efficiency and scalability of SHAREK. In a similar setting, Cici et al. (2015) designed a dynamic ridesharing system in which the matching problem is formulated as a maximum cardinality matching in a bipartite graph between drivers and new requests.

Agatz et al. (2011) developed a single-ride-single-trip ridesharing system as a multi-objective optimization model with the aim of minimizing total miles traversed by all vehicles as well as individual travel costs. In a centralized environment, they decomposed passengers and cars into two sets of vertices in a bipartite graph and applied the rolling horizon approach to provide high quality solution to dynamic ridesharing problem. They tested the proposed model through a simulation in metropolitan Atlanta.

Huang et al. (2013) developed two sets of algorithms to solve ridesharing problem: branch-and-bound for static version of the problem, and kinetic tree structure for dynamic version of the problem. Branch-and-bound and integer programming techniques do not consider the dynamic nature of ridesharing problem where new requests arrive at the server continuously. The main issue with these techniques is that by inserting new requests into existing pick-up and drop-off locations the process of rescheduling must re-start from the beginning. For this, they take the kinetic tree approach to address dynamic ridesharing problem. A grid-based indexing method is used to filter out the vehicles that cannot provide service to a new request within the wait time constraint. They tested the proposed algorithm on Shanghai dataset with maximum number of 20,000 taxis with capacity of four.

Schreieck et al. (2016) presented an efficient algorithm for matching ride offers and ride requests in dynamic ridesharing problem using inverted data structure. The limitation of this work is that they did not test the algorithm on a realistic dataset to measure the efficiency of the proposed algorithm. Also, their approach uses Google geocoding service which has restriction upon executing certain number of requests per minute, so in real-world applications with large number of rides offers and requests, this approach is not applicable (*Google geocoding API*, 2020).

Gao et al. (2017) presented algorithms to dynamic ridesharing problem with the aim of maximizing average satisfaction. They developed two algorithms: searching and scheduling. The first algorithm uses a binary search strategy to reduce size of candidate cars in a linear time by only looking at pick-up and drop-off time constraint. The second algorithm checks all constraints and finds a car that maximizes average satisfaction and satisfies all constraints (detour and capacity). To improve the computing speed of the shortest path algorithm, they approximated the distance by partitioning the area into grids and measuring the grid distance. This approach works at the cost of accuracy of the estimated travel time. They evaluated the system in Beijing Chaoyang district, containing 33,000 taxis and 101,952 trips per day.

The paper by (Jia et al., 2017) presented algorithms to solve ridesharing problem in both offline and online modes together in a two-sided market, i.e., workers and customers both benefiting from the ridesharing system. Offline mode happens when all travel plans are known in advance while online mode requires to match a car to a request in real time where requests come to the system continuously. Two objective functions are maximized in their model: customers' social welfare and profits of cars. They proposed an approximation algorithm and two heuristic algorithms (nearest drivers and maximum marginal value) for solving the problem in offline and online mode, respectively. In offline mode, the problem is transferred to the multiple disjoint path (MDP) problem with the aim of finding weighted node-disjoint path in a directed acyclic graph. Their offline algorithm provides a good solution, but it cannot be applicable to the online setting due to lack of information about all tasks in advance. In online mode, the platform considers the tasks one by one based on their arrival time and the quality of the solution is not mentioned. They conducted experiments in the city of Porto, Portugal with 442 taxis including their full-year trajectories.

Jung et al. (2016) presented a type of ridesharing system that specifically prevents excessive passenger detours. They developed three different algorithms to solve the dynamic ridesharing problem: Nearest Vehicle Dispatch (NVD) that matches the new request to the closest geographically available car from the new passenger's pick-up location, Insertion Heuristic (IS) that considers all feasible vehicles based on satisfying time window and capacity constraint and then assigns the best available vehicle to the new request, and Hybrid Simulated Annealing (HSA) that considers all new requests at the same time and applies SA to solve the underlying optimization problem. Given the limited computational time in dynamic ridesharing systems, the quality of the solution obtained by the proposed algorithms remains unknown. Through a simulation study in the city of Seoul, considering a fleet of 600 four-seater vehicles and up to 18,000 requests, the results showed that HSA outperforms the other algorithms and is a suitable solution for maximizing the efficiency of dynamic ridesharing systems.

Three studies by (Ma et al., 2013), (Ma et al., 2015), and (Shen et al., 2016) solve a large-scale dynamic ridesharing problem with the aim of vehicles' total travel distance minimization. In the first step, the problem is scaled down by a searching algorithm that returns a list of candidate cars which can satisfy the new ride. Then in the second step, a scheduling algorithm inserts the new request into all candidate cars' schedules to select a car with minimal additional travel distance. The scheduling algorithm in this step satisfies constraints (e.g., pick-up and drop-off time constraint, and detour constraint) of the existing requests as well as the new request. Another work by Shemshadi et al. (2014) proposed a framework, MARS (multi-agent ridesharing system), where the efficiency of taxi searching is improved through a decremented search approach. They tested the proposed approach for the city of Beijing with around 10,000 taxicabs and 6,200 requests and compared their approach's performance with the taxi search algorithm developed in T-Share.

Hosni et al. (2014) formulated the shared-taxi problem as a mixed integer programming model where vehicles can have different capacities and the requests are served one-by-one upon arrival. Then they presented two approaches to solve the optimization problem: a) a Lagrangian decomposition approach which decomposed the problem into T (number of taxis) subproblems that can be solved in parallel and b) a heuristic approach for finding good solutions within a reasonable amount of computational time. The proposed heuristic algorithm finds the minimum cost route for each taxi that includes the existing passengers and the new request; then the new request is assigned to a taxi with the lowest incremental cost. They tested the performance of the proposed approaches on different sets of instances in both dynamic and static settings of the problem. Their experiments in dynamic setting that consisted of 50 taxis with maximum capacity of 4 and 200 passengers showed that approach (a) is not applicable in solving dynamic settings of the problem due to its high computational time; however, the heuristic algorithm was efficient in providing good quality solutions in running within 50s. The solution from approach (a) served as a benchmark to validate the solution quality obtained from approach (b). Najmi et al. (2017) developed a method for static and dynamic ridesharing problem using rolling horizon approach to match drivers to requests in real time. The optimization problem in their work has different objective functions: maximizing total distance proximity, maximizing total number of matches, maximizing total net distance savings, and maximizing total adjusted distance proximity. Also, they present a heuristic method to cluster the requests in order to improve the efficiency of the algorithms in large-scale ridesharing problem. Their proposed method and algorithms were tested for the Melbourne dataset.

Ota et al. (2015) developed a simulation framework combined with parallelization to make it scalable for large-scale taxi ridesharing services and implemented the simulation model with

Hadoop's MapReduce. Their simulation used over 150 million trips and ran in 10 minutes with a 1200-core cluster. The proposed optimization algorithm uses a queueing approach without considering all the possible combinations of trips to be shared, so a global optimal solution cannot be reached. To show the effectiveness of the framework, they applied the model and algorithms to New York City taxi data.

In a study by Mallus et al. (2017), a new platform for dynamic ridesharing problem called CLACSOON was developed. In this platform drivers avoid taking a detour whenever possible in a way that users can walk to reach the driver along his/her route. They proposed a route matching algorithm with three functionalities: temporal matching in which for each new request the system checks whether a time constraint is satisfied for the existing requests in a trip; geographical matching in which a matching between a driver and a rider is evaluated based on the distance from the path; cost function evaluation which calculates the cost for a shared trip between each driver and rider. Through simulations, this platform was tested in the area of Cagliari with up to 2,500 users and time window of 4 hours.

All the existing ridesharing systems described above, with an exception to the third algorithm in Jung et al. (2016), have served the incoming requests based on first-come first-serve scheme. The earlier discussions concisely explained the advantages and disadvantages of dealing with requests in the queuing mode in terms of computation time and solution quality. To the best of this work's knowledge, only three studies, Alonso-mora et al. (2018), Lowalekar et al. (2019) and similarly Simonetto et al. (2019), solve the dynamic ridesharing problem with batch assignment. In Alonso-mora et al. (2018), the authors built a deterministic optimization model and developed algorithms for a large-scale ridesharing problem in order to match large groups of rides to a fleet of shared vehicles in real time. The objective function is to minimize delays of requests,

i.e., minimizing time between drop-off time and the earliest possible time that the destination could be reached as well as minimizing the number of rejected requests. They leveraged pairwise shareability graph for assigning trips to vehicles and formulated the matching problem as an ILP. They used a state-of-the-art solver, MOSEK, to solve the ILP. Their proposed framework also considers the rebalancing problem in a dynamic ridesharing system which means how to distribute the idle vehicles to high demand areas. They conducted a simulation in the area of Manhattan, including 200,000 requests and a fleet of up to 3,000 vehicles of capacity 10. The results showed the efficiency of the algorithms in providing a solution to different settings of the problem (number of vehicles, number of requests, time window constraint); however, the work has some shortcomings as follows. In several parts of the framework, a number of limits are considered to reach a real-time performance but all at the cost of optimal solutions. For example, setting timeout per vehicle to explore candidate trips for each vehicle and add edges in constructing RTV-graph (request-trip-vehicle), stopping the solver in solving the ILP after spending a specific amount of time before convergence of the algorithm, or setting a limit on the number of vehicles eligible for servicing a request. Specifying these timeouts is useful in keeping the running time of the algorithms short, but it negatively affects the solution quality when the input size of the problem (number of requests, number of vehicles and their capacities) increases. Hence, the main issue with this work is its lack of validation, meaning that the optimality of the final solution is not known at different scales of the problem. One way to resolve the scalability issue is using techniques to parallelize the computational workload in a centralized approach (see Chapter 2.0) or employ decentralization of the decision-making process. The mathematical formulation in Lowalekar et al. (2019) is similar to Alonso-mora et al. (2018) in terms of constructing RTV graph, but the objective function is to maximize number of served requests. Similar to these works, this

dissertation processes the requests using batch assignment; yet, the proposed formulation in this dissertation differs from these other works in that the objective function is to minimize total distance travelled by all vehicles to serve the requests in the pool.

In summary, from the above discussions and analysis, the shortcomings of the centralized approach are:

- Inherent to the centralized approach, making decision in real time will be intractable when the size (number of vehicles, capacity of vehicles, number of requests) of the problem increases.
- Due to the high degree of dynamism in dynamic ridesharing problem, the centralized approach is not flexible in handling a large amount of changes in the environment when some vehicles should be added to or removed from the system.
- The centralized approach demands a heavy communication requirement, i.e., the vehicles must frequently (e.g., every 10 seconds) communicate with the central decision maker to update their status (spatio-temporal locations).
- The centralized approach is not robust because if the single agent decision maker fails, the entire system fails.
- In the centralized approach, the developed models assign requests to vehicles based on first-in-first-out scheme which provides a solution quickly but not necessarily optimal. See Section 2.1.4 for more details on this.

Considering the challenges with ridesharing problem and the shortcomings of the current approaches, this dissertation proposes an approach based on cooperative decision making in a decentralized multi-agent system with the aim of tackling scalability, flexibility, and robustness

issues along with finding a tradeoff between optimality of assignment and computational complexity.

4.0 Proposed Research

Disruptive technologies in IoT and automotive industry, particularly autonomous vehicles, is the motivation behind proposing a multi-agent approach to solve inherently distributed ridesharing problem. In this respect, this dissertation formulates this spatio-temporal resource allocation problem as a decentralized optimization, designs and develops a set of algorithms to obtain optimal or at least near-optimal solutions. Note that this multi-agent approach is different from other multi-agent approaches applied to transportation, including those discussed in Section 3.2, in which a *single-agent decision maker* decides which requests should be assigned to which vehicles. This chapter presents a decentralized optimization approach to address dynamic ridesharing problem where the global objective function is to minimize the total travel distance of the agents. In the proposed approach, the global objective function is decomposed into a sum of local loss functions where each is only known to one particular agent. Each local objective function must satisfy a set of local constraints, i.e., capacity of each agent and time windows for each request's pick-up and drop-off locations. All requests are known to all agents in each round (a reference to every new instance of the problem that the algorithms execute) and each agent will exchange and align its decision with the agents identified as *similar*.

The proposed multi-agent approach consists of *multiple cooperative decision-makers,* homogeneous and/or heterogeneous agents in terms of capacity, and has the following characteristics: (a) each vehicle as an agent has computing resources to perform computation on its own; (b) each agent receives a portion of the entire information for decision making; (c) the agents communicate with each other and exchange messages; and (d) there is no central controller

and the decision-making process is completely decentralized among the agents in which they jointly solve the given matching task.

The proposed decentralized approach has three types of agents:

- ***User*** is a person requesting a ride by submitting a request to the *dispatcher agent*.
- ***Dispatcher*** is an automated unit with computational and communication resources, performing two major roles. One is acting as a mediator between the *user* and the *vehicles,* i.e., it receives requests from users, keeps them in the pool based on a hyperparameter defined in the system (a specific amount of time or a specific number of requests) and dispatching them to the decision makers. After solving the given matching problem by the decision makers, it notifies the users of success or rejection of their requests. Another is acting as a coordinator between the decision makers, by constructing a *coordination graph* between the decision makers based on their *similarity*, to identify which decision makers should coordinate their actions. The action space in the coordination graph is exponential in the number of requests, vehicles and their capacities; therefore, to address scalability, a graph decomposition is proposed to reduce the complexity of the matching problem to find efficient solution.
- ***Vehicle*** is an agent with computational and communication resources and performs two sets of tasks: local and cooperative.

  Local. Each vehicle performs its own local tasks including scheduling and routing upon receipt of new requests from the *dispatcher*. Each vehicle computes its own utility values (e.g., travel distance by serving a request) for each request or combination of requests that they can serve, taking into account the time window and capacity constraints.

Cooperative. After constructing the coordination graph by the *dispatcher agent* based on the *similarity* between the decision makers, each vehicle knows the other vehicles with which they should interact to fulfil their joint tasks. With this, all the linked vehicles jointly solve the matching problem and assign user(s) to vehicle(s). This is accomplished by optimizing an objective function, e.g., minimizing travel distance by vehicles. After the final assignment, the vehicles submit their decisions to the *dispatcher agent*.

4.1 Assumptions

This dissertation makes the following assumptions:

- the entire operation (global and local decisions) is automated and there is no human involved in decision making;
- each agent is equipped with sufficient computing resources to performing its own computation; and
- each agent is able to communicate with other agents, when needed, using a reliable communication mechanism.

4.2 Problem Formulation and Definitions

Given a road network as a graph $G(V, E, W)$, consisting of vertices, edges, and weights for each edge, function $c(p, d)$ computes the shortest path from p to d on the road network.

A *request* is a tuple of $r = (p, d, tw_p, tw_d, q)$, where p is pick-up location, d is drop-off location, tw_p is hard time window for pick-up, tw_d is hard time window for drop-off, and q is number of passengers.

A *schedule* is an ordered set of locations on the road network, $s = \{l, p_3, p_2, d_1, d_3, d_2\}$, where the first element shows the current location of the vehicle agent and the rest of the elements indicate the schedule of the vehicle agent for picking up and/or dropping off passengers.

A *path* is an ordered set of vertices on the road network, $p = \{l, v_1, v_2, v_3, v_4\}$, indicating the set of vertices (from shortest path) that a vehicle agent needs to traverse based on its schedule.

Similarity between two vehicle agents, as seen in (4.1), is a reference to a situation where after satisfying the problem's constraints, such as time window and capacity by each decision maker, there is at least one request that can be served by the two vehicle agents. Suppose B_i and B_j are two row vectors with an arbitrary size representing the requests that can be served by vehicle agents i and j, respectively. The similarity between these two vehicle agents is defined as:

$$similarity(i,j) = \exists\ b\ in\ B_i | b \in B_j = |B_i \cap B_j| \geq 1 \qquad (4.1)$$

Consider a network of m homogeneous and/or heterogeneous agents labeled by $A = \{1,2,\dots,m\}$, each with a capacity of $c = \{cap_1, cap_2, \dots, cap_m\}$, the global objective function is to minimize total travel distance of the agents D_i on the road network (4.2). Consider a graph G(R,A) where R is the set of requests as nodes and A is the set of edges associated with travel cost between each pair of requests.

$$Min\ F(x) = \sum_{i\in m}\sum_{j\in R}\sum_{k\in R} x_{ijk}e_{jk} = \sum_{j\in R}\sum_{k\in R} x_{1jk}e_{jk} + \cdots + \sum_{j\in R}\sum_{k\in R} x_{mjk}e_{jk}$$

$$= F_1(x) + \cdots + F_m(x) \qquad (4.2)$$

The aim is to decompose the objective function in (4.2) into a set of m loss functions $\{F_1(x), F_2(x), \dots, F_m(x)\}$ where each loss function as shown in (4.3) is equivalent to an instance of open-loop TSP problem. For example, in Figure 6 a group of agents (six agents) cooperate with each other to solve the underlying optimization problem.

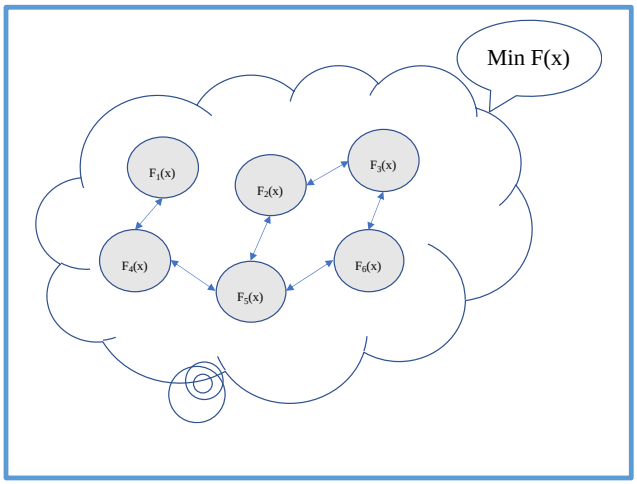


Figure 6 An overall illustration of the global objective function's decomposition

The constraints that each vehicle agent must locally satisfy, as formulated in (4.4), are: number of passengers that should be less than or equal to its capacity; each request is served exactly one time; pick-up time and drop-off time for the existing requests and new requests that should be served within the specified time window. Finding an optimal value for $F_i(x)$ depends on the vehicle agent's capacity. For vehicle agents with a small capacity, an optimal value can be obtained by an exhaustive search, but for larger capacities, efficient algorithms such as heuristic at the cost of optimality, are needed. In (4.4) and (4.5), t_r, t_p, t_d, t_d^* indicate time of the request, pick-up time, and drop-off time, respectively; and t_d^* is the earliest possible time at which the drop-off location

would be reached. Note that t_r, t_p, t_d, and t_d^* are absolute times while tw_p and tw_d are relative times. c(p,d) is the travel cost between pick-up and drop-off location of a request.

$$Min\ F_i(x) = \sum_{j \in R} \sum_{k \in R} x_{jk} e_{jk} \qquad (4.3)$$

$$S.T. \begin{cases} \sum_{j \in R} \sum_{k \in R} x_{jk} \leq cap \\ \sum_{j \in R} x_{jk} = 1 \quad k \in R \\ \sum_{k \in R} x_{jk} = 1 \quad j \in R \\ t_r \leq t_p \leq t_r + tw_p \\ t_d^* \leq t_d \leq t_d^* + tw_d \\ x_{jk} \in \{0,1\} \end{cases} \qquad (4.4)$$

$$t_d^* = t_r + c(p,d) \qquad (4.5)$$

This dissertation uses batch assignment mode (for more detail, see Section 2.1.4), i.e., given a set of requests $R = \{r_1, r_2, \dots, r_n\}$ in the pool and a network of vehicle agents A with regard to their current schedule and path, with the goal of optimally assign requests to vehicle agents in a way that minimizes the cost functions $F_i(x)$ and satisfies constraints in (4.4). In each round, some requests may be rejected due to lack of enough empty seats or to inability to satisfy the requests' constraints. The system keeps all the rejected requests in the pool and reconsiders them in the next.

4.3 Methodology

This section explains the proposed multi-agent approach in detail and discusses a set of algorithms to solve the optimization problem introduced in the previous section. Figure 7 illustrates a high-level view of the proposed decentralized approach in which the vehicles agents form a mobile multi-agent network.

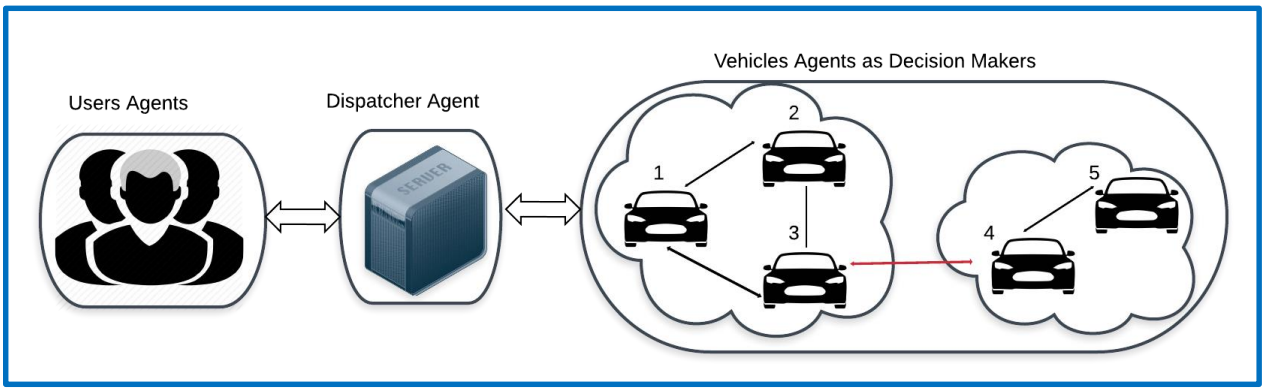


Figure 7 Overview illustration of the decentralized approach

4.3.1 Decentralized Approach

Given a set of requests R in the pool and a set of vehicle agents A at their current schedule, in each round the proposed distributed solution follows these steps to assign requests to agents:

1) Dispatcher agent sends requests R in the pool to all Vehicles agents A.
2) Each Vehicle agent A_i, considering its capacity and time constraints of the new and existing requests, solves an instance of open-loop TSP for all requests and returns bids to Dispatcher agent.

3) Dispatcher agent aggregates bids, forms coordination graph between Vehicles agents and analyzes it to decompose the overall set of Vehicles agents into subsets.
4) Dispatcher agent solves an instance of bin packing or multiple knapsack problem to allocate requests to each subset.
5) Dispatcher agent sends a set of requests to the corresponding Vehicles agents in each subset.
6) Each subset of Vehicles agents solves its own subproblem through negotiation and sends the final decisions back to Dispatcher agent. At the end of negotiation, each Vehicle agent updates its schedule accordingly.
7) Dispatcher agent notifies users whether their requests have been accepted or rejected and if accepted, which Vehicle agent is assigned to serve it.

4.3.2 Coordination Graph and Decomposition

This section thoroughly explains Step 3 of the proposed approach, which is one of the major decisions. Each vehicle agent locally solves the scheduling problem with one of the heuristic or exact algorithms discussed in Section 2.1.3, satisfying the constraints of capacity and time window of the onboard passengers and the new requests. This work computes an exact solution to open-loop TSP. The time complexity for computing the bids for each agent is $O(R^c(p!))$ where R, c, and p are the number of requests in the pool, the number of vehicle agent's empty seats, and the number of points in the schedule of the vehicle agent, respectively. Dispatcher agent receives the bids from each vehicle agent and constructs the agent_request matrix. For instance, in Table 3 vehicle agent 1 can provide a service to request 1, request 2, request 3, and both requests 1 and 2. Dispatcher agent constructs a coordination graph in the form of a *hypergraph* where each node

represents a decision maker, and an hyperedge indicates which vehicle agents can serve a request (Figure 8).

Table 3 An example of agent_request matrix

B =

	req1	req2	req3	req4	req1,2	req5
A1	a_1	a_2	a_3		a_4	
A2	a_5	a_6	a_7		a_8	
A3			a_9	a_{10}		
A4			a_{11}	a_{12}		a_{13}
A5				a_{14}		a_{15}

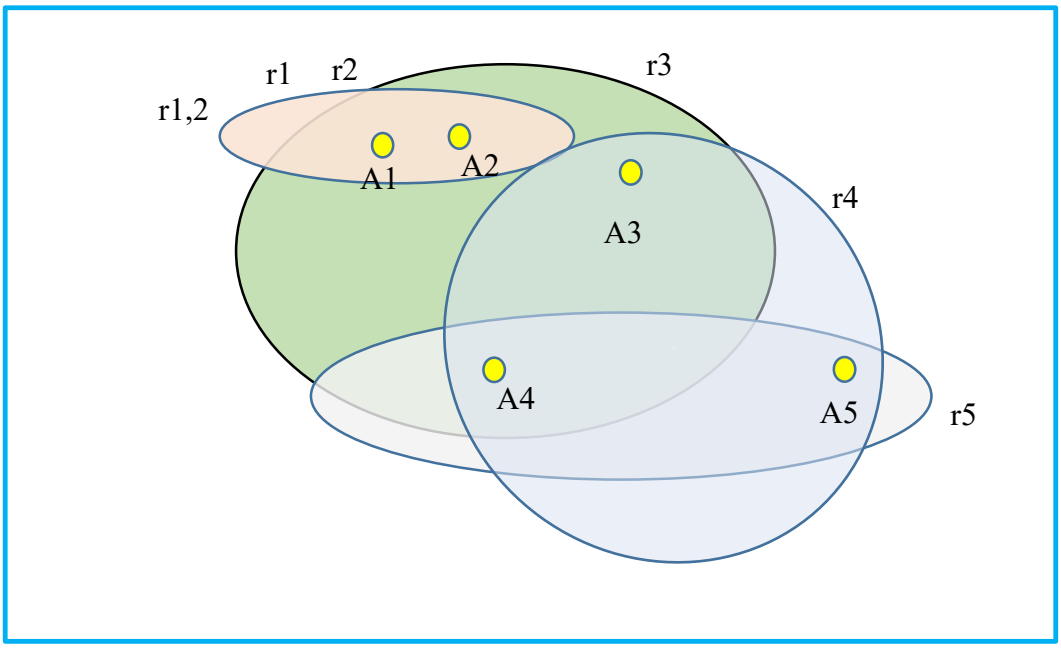


Figure 8 Hypergraph representation of agent-request

The decomposition algorithm that I propose here is to convert the hypergraph into a simple graph and then construct Laplacian matrix from it and apply spectral clustering theory for

decomposition. After forming the agent_request matrix, dispatcher agent constructs the similarity matrix between agents by calculating cosine similarity between each pair of vehicle agents. Similarity graph is an undirected weighted graph and accordingly similarity matrix is symmetric with size of $m * m$ where m is the number of vehicle agents. Figure 9 shows the main steps of the decomposition algorithm which takes agent_request matrix as input and divides the vehicle agents into subsets.

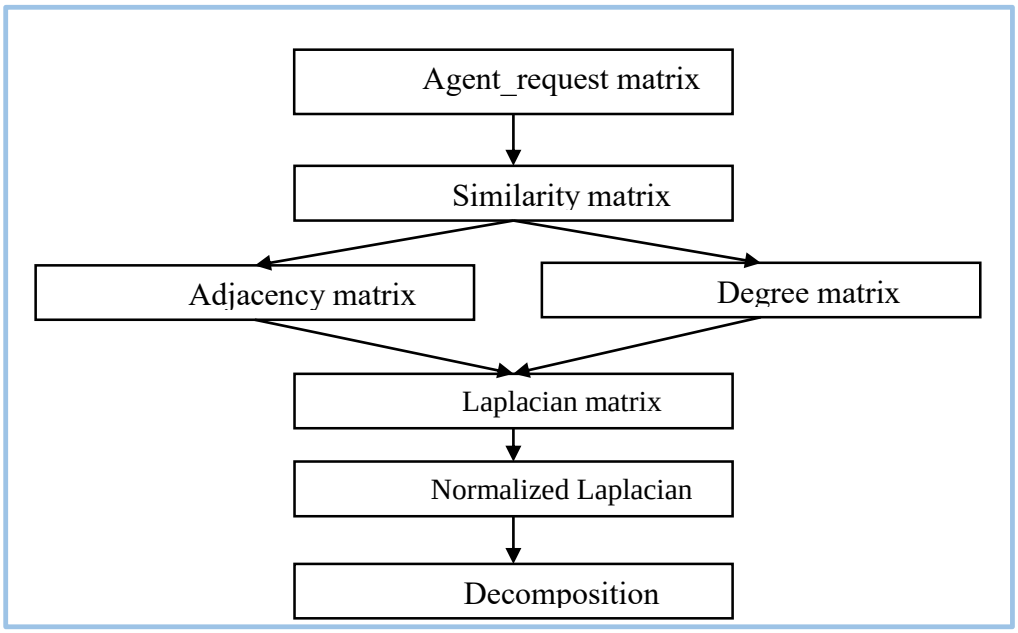


Figure 9 Overview of the decomposition algorithm

$$B_{ij} = \begin{cases} a & a \in R^{+} \\ 0 & otherwise \end{cases} \text{ j} \in \{1,2,\dots,n\}, i \in \{1,2,\dots,m\} \qquad (4.6)$$

$$0 \leq \cos(\theta_{ij}) = \frac{B_i . B_j}{|B_i|\,|B_j|} \leq 1 \qquad (4.7)$$

Each element in B_{ij} (4.6) shows the cost that vehicle agent i serves request j. Each row vector in matrix B (agent_request matrix) has size of $n = \sum_{k=1}^{c} \binom{R}{k} = O(R^c)$ that is equal to total possible number of combinations of requests. R and c are the number of requests in the pool and the number of vehicle agent's empty seats, respectively. The height of matrix B is the number of vehicle agents in the system. Since similarity matrix is symmetric, dispatcher agent needs to perform $(m^2 - m)/2$ comparisons between vehicle agents and calculate similarity by (4.7). Note that the value of similarities is in [0,1] because all values in matrix B are non-negative, i.e., in R^{+}.

The dynamic of the environment continuously changes over time due to changes in both the spatio-temporal distribution of the requests and vehicle agents' trajectories. Consequently, the structure of the hypergraph dynamically changes as well once the system processes a new set of requests in the pool. Analyzing different decomposition results based on different structures of the coordination graph is a big challenge and an active area of research (Gottlob & Greco, 2013). The hypergraph must be appropriately decomposed so that computational efficiency, convergence and quality of the solution in each subgraph are theoretically guaranteed. Obviously, this decomposition provides a suboptimal solution to the global optimization problem but significantly reduces the complexity of the problem and makes it tractable.

4.3.3 Spectral Graph Theory

This section describes how this dissertation benefits from graph *Laplacian* and spectral clustering method, which is a well-established method for graph decomposition in terms of theory and practicality, to identify group of vehicle agents with high similarity; for further details see (Von Luxburg, 2007). The intuition behind the similarity graph and spectral clustering is that given a set of vehicle agents $A = \{A_1, A_2, \dots A_m\}$ and notion of similarity between each pair of vehicle agents, the goal is to partition the vehicle agents into several subsets in such a way that vehicle agents in the same subset are similar and vehicle agents in different subsets are dissimilar to each other.

Let define coordination graph as $G = (A, E)$ with vertex set as vehicle agents and the edge set carries a non-negative weight between vertices. The weighted *adjacency* matrix of the graph is the matrix $W = (w_{ij})_{i,j=1,2,\dots,m}$. $w_{ij} = 0$ means that there is no similarity between vehicle agent i and vehicle agent j. As the coordination graph is undirected, $w_{ij} = w_{ji}$.

The degree of a vehicle agent in the coordination graph is defined as:

$$d_i = \sum_{j \in \{1,2,\dots,m\} \setminus i} w_{ij} \qquad (4.8)$$

Then the *degree* matrix D is defined as the diagonal matrix with the degrees $d_1, d_2, \dots, d_m$ on the diagonal.

Graph *Laplacian* matrix is the main tool for spectral clustering; for further details on spectral graph theory see (Chung and Graham, 1997). The rest of this section explains how the vehicle agents in the coordination graph will be decomposed from graph Laplacians.

The *unnormalized* graph Laplacian matrix L is defined as:

$$L = D - W \qquad (4.9)$$

The most important facts needed for spectral clustering are summarized by the following propositions.

Proposition 1: The matrix L satisfies the following properties (Chung and Graham, 1997):

a) For every vector $f \in R^m$ we have

$$f^T L f = \frac{1}{2} \sum_{i,j=1}^{m} w_{ij} (f_i - f_j)^2 \geq 0 \qquad (4.10)$$

which substantiates that L of every graph is positive semidefinite. In linear algebra, f, the eigenvector of the second smallest eigenvalue of L is called Fiedler eigenvector.

b) L is symmetric and positive semi-definite.

c) The smallest eigenvalue of L is 0 and the corresponding eigenvector is the constant one vector.

d) L has m non-negative, real-valued eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_m$.

Proposition 2: Number of connected components k in G is equal to the number of eigenvalues 0 of L (Chung and Graham, 1997).

Without loss of generality, in the case of k connected components, the Laplacian matrix L has as many eigenvalues 0 as there are connected components where the corresponding eigenvectors represent the indicator vectors of the connected components.

The graph Laplacian matrix can be normalized as (Shi & Malik, 2000) which is called *random walk* normalized Laplacian:

$$L_{nor} = D^{-1} L = I - D^{-1} W \qquad (4.11)$$

or (Ng et al., 2002) which is called *symmetric* normalized Laplacian:

$$L_{sym} = D^{-1/2} L D^{-1/2} = I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}} \qquad (4.12)$$

So, the question is: Why is this work interested in normalizing the Laplacian matrix? According to (Von Luxburg, 2007), the normalized spectral clustering considers both objectives of minimizing the between-partition similarity and maximizing the within-partition similarity while unnormalized algorithm only considers the first objective. Moreover, performing statistical analysis on both normalized and unnormalized spectral clustering algorithms with infinite sample size proves that unnormalized spectral clustering cannot converge to good solutions, resulting in unbalanced partitions with a significant difference between number of data points in the partitions (Von Luxburg, 2007).

From the above explanations, the steps of the proposed decomposition algorithm are:

Input: Agent_request matrix $B \in R^{m \times n}$, number of partitions k

Output: divides set of vehicle agents into k partitions

1) Construct similarity matrix S from B
2) Compute Degree matrix D and Adjacency matrix W from S
3) Compute Laplacian matrix L from D and W
4) Normalize Laplacian matrix L_{nor} or L_{sym} if D is not singular; otherwise, use L. In the case of using unnormalized Laplacian, skip this step.
5) Compute the first k eigenvectors of L or L_{nor} or L_{sym} as $C \in R^{m \times k}$
6) Cluster the points in C with k-means algorithm into k partitions $C_1, C_2, \ldots, C_k$

Algorithm 1: Step 1: Construct S

Input: Agent_request Matrix B

Output: Similarity Matrix S

For $d_i \in B$ {each row of B}

For $d_j \in B$ {each row of B}

$S_{i,j} = dot\_product(d_i,d_j)/l^2\_norm(d_i).l^2\_norm(d_j)$

End For

End For

return S

The Java library (The Apache Commons Mathematics Library) is used to compute l^2 norm and dot product between two vectors.

Algorithm 2: Step 2: Construct D

Input: Similarity Matrix S

Output: Degree Matrix D

For $i \in S$ {each row of S}

sum = 0

For $j \in S$ {each column of S}

sum += $S_{i,j}$

End For

$D_{i,i}$ = sum

End For

return D

Time complexity is $O(m^2)$ where m is the number of vehicle agents.

Algorithm 3: Step 2: Construct W

Input: Similarity Matrix S

Output: Adjacency Matrix W

W = S

For i ∈ W {each row of W}

$W_{i,i} = 0$

End For

return W

Time complexity is O(m) where m is the number of vehicle agents.

Algorithm 4: Step 3: Construct L

Input: Degree Matrix D, Adjacency Matrix W

Output: Laplacian Matrix L

L = subtract W from D

return L

The Java library (The Apache Commons Mathematics Library) is used to do subtraction of the two input matrices.

Algorithm 5: Step 4: Construct L_{nor}

Input: Laplacian Matrix L, Degree Matrix D

Output: Random Walk Normalized Laplacian Matrix L_{nor}

D’ = inverse D if D is not singular

L_{nor} = multiply(D’,L)

return L_{nor}

The Java library (The Apache Commons Mathematics Library) is used to do multiplication of the input matrices and inverse the degree matrix.

Algorithm 6: Step 4: Construct L_{sym}

Input: Adjacency Matrix W, Degree Matrix D

Output: Symmetric Normalized Laplacian Matrix L_{sym}

L_{sym} = multiply(multiply($D^{-1/2}$,W),$D^{-1/2}$)

return L_{sym}

The Java library (The Apache Commons Mathematics Library) is used to do multiplication of the input matrices.

Algorithm 7: Steps 5 and 6: Partitioning using L or L_{nor}

Input: Random Walk Normalized Laplacian Matrix L_{nor} or Laplacian Matrix L, number of clusters k

Output: corresponding vehicle agents in k clusters

Compute eigenvalues of L or L_{nor} and sort ascendingly

Compute the first k (smallest) corresponding eigenvectors $C \in R^{m \times k}$

Do K_means clustering on C

return vehicle agents in each cluster C_i

The Java library (The Apache Commons Mathematics Library) is used to compute eigenvalues, eigenvectors and do k-means clustering.

Algorithm 8: Steps 5 and 6: Partitioning using L_{sym}

Input: Symmetric Normalized Laplacian Matrix L_{sym}, number of clusters k

Output: corresponding vehicle agents in k clusters

compute eigenvalues of L_{sym} and sort ascendingly

Compute the last (largest) k corresponding eigenvectors $C \in R^{m \times k}$

C_n = Normalize each row of C

Do K_means clustering on C_n

return vehicle agents in each cluster C_i

The Java library (The Apache Commons Mathematics Library) is used to compute eigenvalues, eigenvectors and do k-means clustering.

The above algorithm takes number of partitions k as input. Deciding what k should be used depends on the density of the subgraph in each partition and the complexity of the algorithm that is going to be designed to solve the local problem (subproblem) in each partition. Assuming that the algorithm (Section 0), which solves each local problem, can provide a good solution in a reasonable amount of time for a complete subgraph with maximum g nodes, the number of partitions could be found by a naive method as $k = m/g$, where m is the number of vehicle agents in the system. It is important to note that the algorithm should be applied only to the connected parts of the coordination graph. Obviously, the structure of the coordination graph does heavily affect the value of k. If there are isolated nodes in the coordination graph, then k increases accordingly and the algorithm put the isolated nodes in separate partitions.

Step 6 of the decomposition algorithm applies the spectral clustering algorithms using (4.9) (unnormalized Laplacian), (4.11) (random walk normalized Laplacian), and (4.12) (symmetric normalized Laplacian). In the case of using unnormalized Laplacian, the algorithm considers the eigenvectors that correspond to the first k smallest eigenvalues of L. Also, according to (Shi & Malik, 2000), the algorithm considers eigenvectors with the smallest eigenvalues of L_{nor}; however, in the case of using (4.12), according to (Ng et al., 2002), the algorithm considers the k largest eigenvectors of W. There is one extra normalization step when the algorithm applies spectral clustering algorithm introduced in (Ng et al., 2002). After stacking the eigenvectors in columns, it is needed to renormalize each row vector to make sure that each row has unit length.

After partitioning the coordination graph which determines vehicles agents that need to cooperate on a set of requests, dispatcher agent needs to allocate requests to each partition (subproblem). The next section formulates this allocation problem as another optimization problem similar to bin packing or multiple knapsack problem. Then it presents a greedy algorithm to solve the underlying optimization problem.

4.3.4 Allocation of Requests to Subproblems

After partitioning vehicle agents into k subsets, in the next step dispatcher agent distributes requests between subsets. In this step, this work considers two different scenarios: (a) number of requests greater than or equal to number of empty seats; (b) number of requests less than number of empty seats. The first scenario is formulated as multiple knapsack optimization problem with the objective function defined in (4.13). The system can assign a weight to each request, e.g., requests from users with disability can have higher weight for serving. The goal is to maximize number of requests that are assigned to each partition. The constraints as defined in (4.14) satisfy

the capacity of each partition and ensure that each request is assigned to maximum one partition. Note that the capacity of each partition equals to the sum of empty seats of vehicle agents belongs to that partition.

$$Max \sum_{k} \sum_{i} w_i r_{ik} \qquad (4.13)$$

$$S.T. \begin{cases} \sum_{i} r_{ik} = C_k, \ \ i \in Requests\{1,2,\dots,n\} \\ \sum_{k} r_{ik} \leq 1, \quad k \in partitions\{1,2,\dots,k\} \\ r_{ik} \in \{0,1\} \end{cases} \qquad (4.14)$$

In the second scenario, the optimization problem is formulated as bin packing problem as seen in (4.15). The goal is to minimize number of bins (partitions) for assigning requests. The constraints in (4.16) are to ensure that number of requests in each bin does not go beyond the capacity of each partition and each request is assigned maximum one partition.

$$Min \sum_{k} B_k \qquad (4.15)$$

$$S.T. \begin{cases} \sum_{i} r_{ik} \leq C_k, \ \ i \in Requests\{1,2,\dots,n\} \\ \sum_{k} r_{ik} = 1, \quad k \in partitions\{1,2,\dots,k\} \\ r_{ik}, B_k \in \{0,1\} \end{cases} \qquad (4.16)$$

To solve the above optimization problems, a greedy algorithm is designed. Before going deep into the details of the algorithm, it is worthy of note that there are two different measurements while computing the capacity of each partition from agent_request matrix: number of *actual* empty

seats and number of *functional* empty seats. The first measurement refers to the actual number of empty seats that a vehicle agent has in a specific timestamp. On the other hand, the second measurement refers to how many requests a vehicle agent can serve at the same time, regarding its number of empty seats. For example, from the agent_request matrix showed in Table 3, the number of actual empty seats for vehicle agent3 is four, but its number of functional empty seats is two because it can only serve two requests at the same time. Following the above discussion and the structure of agent_request matrix in Table 3, there are two observations: the number of functional empty seats is less than or equal to the number of actual empty seats; different vehicle agents can have similar actual number of empty seats (or capacity), but dissimilar number of functional empty seats. The algorithm designed in this section considers the second measurement to compute the capacity of each partition.

Algorithm 9: Allocation Algorithm: Allocation of requests to partitions

---

Input: Partitions containing their vehicle agents P, agent_request matrix AR

Output: allocation of requests to each partition

Sort requests in AR based on their weights and dependency -> R

Compute total number of empty seats from P by summing the capacity of each partition -> ES

while (ES > 0 and |R| > 0)

    Allocate the request to the largely dependent P_i

    Update capacity of P_i and ES based on length of the request

    Remove R_j from R

end while

return set of requests needed to be served in each partition

Time complexity is $O(n^2m)$ where m and n are the number of vehicle agents and the number of columns (all combinations of requests can be served by the vehicle agents) in the agent_request matrix, respectively.

The algorithm in the first step sorts the requests in agent_request matrix based on their dependency and weights in descending order. The reason for giving higher priority to dependent requests instead of serving them separately is that a better solution might be obtained in the first case. As an example (see Figure 10), there are two requests (blue and orange) with pick-up and drop-off locations with the cost of 50 and 90 for serving each of them, respectively. These requests can be served by cost of 140 and 90 if they are considered separately and together, respectively.

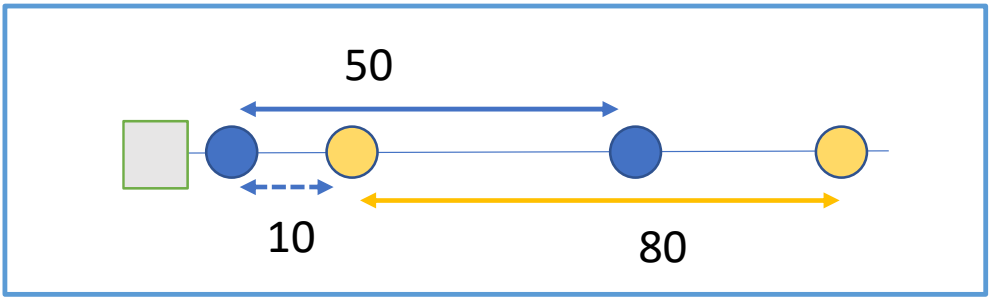


Figure 10 Serving dependent requests together with cost of 90 versus separately with cost of 140

In the second step the algorithm computes total number of empty seats from partitions, considering the functional number of empty seats in each vehicle agent. In the third step the algorithm allocates each request to the *largely dependent* partition, i.e., there are higher number of vehicle agents in that partition which can serve that request. After allocating the request to a partition, if there are empty seats in the partitions, the algorithm in the fourth step updates total

capacity of partitions and the corresponding partition P_i. In the fifth step, the request is removed from the list. Steps three through five will be iterated until the list of requests is empty or the partitions are full.

An example is illustrated in Figure 11 to explicate how the algorithm allocates the requests to the partitions. Suppose the list of requests in the agent_request matrix is $R=\{r_1,r_2,r_3,r_4,r_5,r_6,r_7,r_8,r_9,r_{10},r_{3,5}\}$ and the partitions are $P=\{P_1,P_2,P_3,P_4\}$. As we can see in panel (a), there are ten requests and nine empty seats (i.e., one request should be rejected): three vehicle agents with the total capacity of four in P_1, three vehicle agents with total capacity of three in P_2, one vehicle agent with total capacity of one in both P_3 and P_4. The algorithm sorts the requests in R based on the criteria (weight and dependency) already explained where the list of requests will be $R=\{r_8,r_7,r_4,r_{3,5},r_1,r_2,r_6,r_9,r_{10}\}$. The algorithm allocates r_3 and r_5 to P_1 because of their dependency along with r_1 and r_4. Requests r_8, r_7, and r_6 are allocated to P_2, r_9 to P_3 and r_{10} to P_4. Note that r_2 is rejected because it only belongs to P_1 and before reaching this request in R, the corresponding partition is full. Panel (b) shows the final output of allocating requests to the partitions.

To summarize, the greedy algorithm in each iteration allocates a request to the largely dependent partition if the partition has empty seat. The algorithm considers two conditions in the loop to satisfy the conditions of knapsack as seen in (4.13) and (4.14) and bin packing problems as seen in (4.15) and (4.16). If $|R|>>ES$ the algorithm repeats the steps in the loop ES times; on the other hand, if $ES>>|R|$ the algorithm repeats the steps in the loop $|R|$ times.

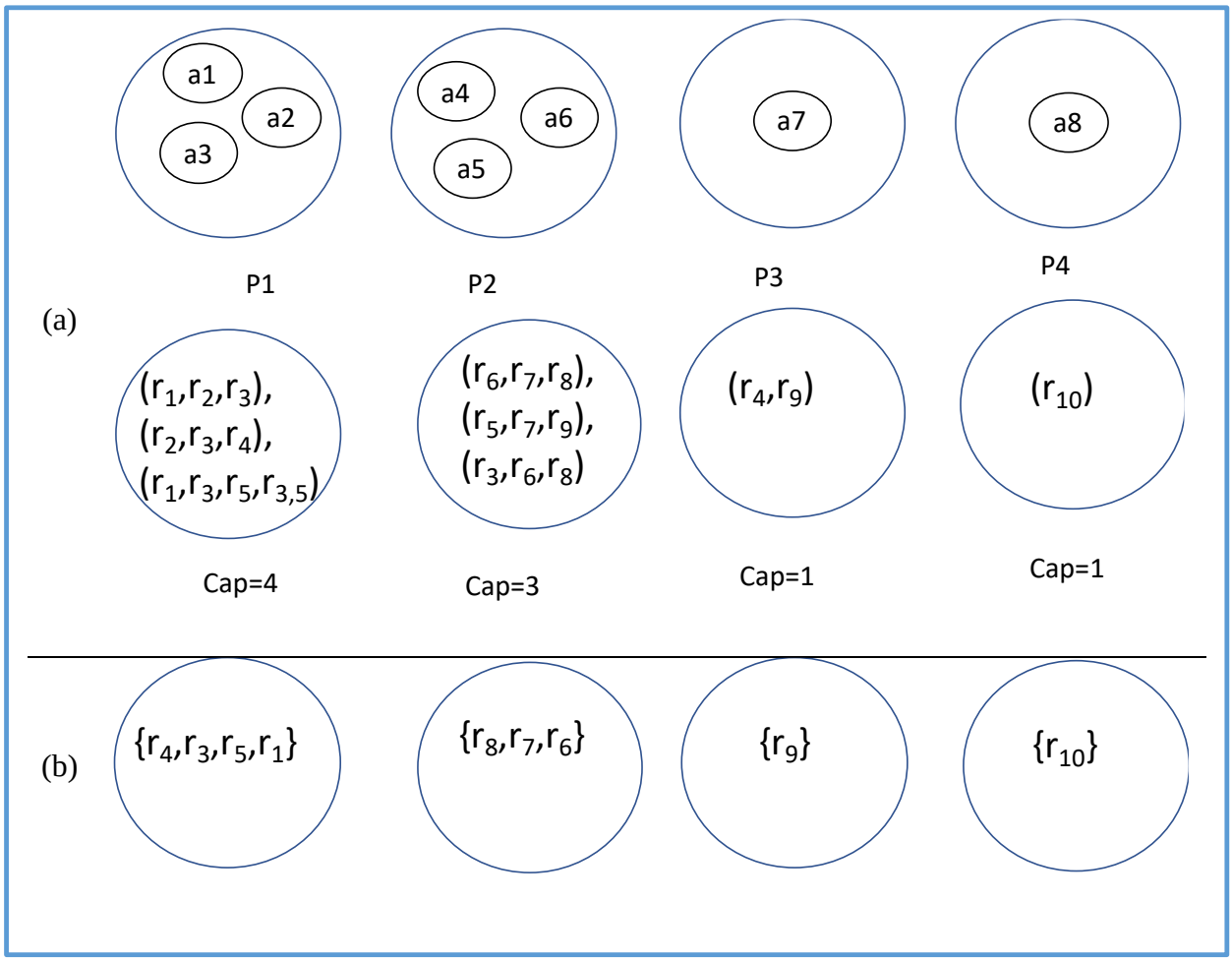


Figure 11 An example of allocating requests to partitions

4.3.5 Solving Subproblems

So now the question is: How do the decision makers in each partition work together locally in order to make a sequence of joint decisions with the aim of optimizing an objective function (e.g., in this dissertation minimizing total travel distance)? Figure 12 illustrates an example with three vehicle agents in a partition where there is a similarity between each pair of vehicle agents. To address the above question, there are three popular formalizations among others in the literature (Farinelli et al., 2014). Decentralized Markov Decision Process (MDP) (Bernstein et al., 2000) is

one technique that can be used to find an optimal policy for taking actions between agents, but it is not practical in real-world scenarios due to its inherent complexity. A second technique is Partially Observable MDP (Velagapudi et al., 2011) that can be scaled to hundreds or thousands of agents, but it only scales to multi-agent systems where agents' interactions are very sparse. Another technique is Distributed Constraint Optimization Problem (DCOP) (Modi et al., 2005; Yeoh & Yokoo, 2012) that shows to be promising in solving large-scale multi-agent problems. This dissertation formulates the above question as classical DCOP (Modi et al., 2005) where agents in each partition are fully cooperative, have deterministic behavior, and need to coordinate their actions in a decentralized manner, in order to optimize their objective functions. The next section presents how the problem in each partition is formulated as DCOP.

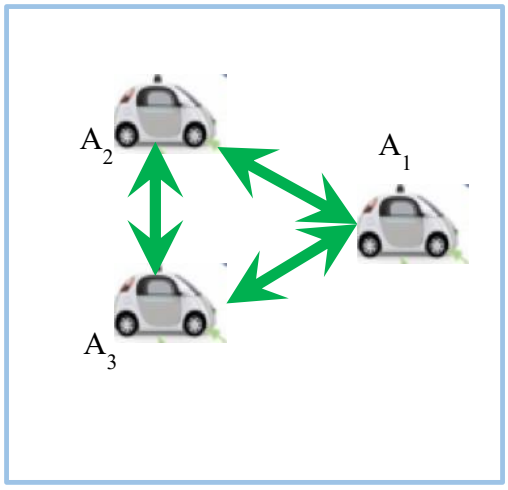


Figure 12 Cooperative agents to solve subproblem in each partition

4.3.5.1 Intra-Partition Problem Formulation

This section presents a formal definition of DCOP for solving optimization problem in each partition. A classical DCOP is formulated as a tuple <A, X, D, F> where $A = \{a_1, a_2, \ldots, a_m\}$ is

a finite set of agents (decision makers), $X = \{x_1,x_2,\ldots,x_m\}$ is a finite set of variables, each agent owns exactly one variable in this work, $D = \{D_1,D_2,\ldots,D_m\}$ is a set of discrete and finite variable domains, each variable owns one domain, and $F = \{f_1,f_2,\ldots,f_m\}$ is a finite set of cost functions describing the constraints among variables. In this work, size of each domain D_i is equal to the number of dimensions of agent i in the agent_request matrix, e.g., if agent i can serve two requests separately and together then $|D_i|=3$. Each function depends on a set of variables that shows the arity of the function. For example, if the function depends on two variables, it is a binary function. The goal in DCOP is to find a variable assignment that maximize/minimize the sum of constraints as defined in (4.17). In this work, the goal is to minimize total travel distance between agents in each partition. Note that finding an optimal solution for DCOP in each subproblem is still a combinatorial optimization problem and NP-hard (Modi et al., 2005).

$$\boldsymbol{Goal}: argmax_x \sum_i f(x_i) \quad \text{or } argmin_x \sum_i f_i(x_i) \qquad (4.17)$$

There are three dominant ways of DCOP representation from agent coordination and algorithmic perspectives, considering the following assumptions (Fioretto et al., 2018):

- Each agent controls a variable which its domain is known to both the owner agent and neighboring agents.
- Each agent knows about the values of the cost function of at least one of its local variables.
- Each agent knows about its neighboring agents. This information is issued to the agents by the dispatcher agent after the decomposition step.

One way to represent DCOP is via a *constraint graph* as $G_p = (X,E_c)$ where X represents the agents nodes and an undirected edge ϵ E_c between two nodes exists if and only if there is a similarity between those nodes. *Pseudo-Tree* is another way to represent DCOP in which there is

a partial ordering among the agents. The nodes in a pseudo-tree representation are arranged as a subgraph T_p of G_p such that T_p is a spanning tree of G_p. A third way to represent DCOP is through a *factor graph* where a bipartite graph is used to represent the factorization of a function. Given the objective function in (4.17), the bipartite graph comprises two types of nodes: variable nodes and function nodes, depicted by circles and squares, respectively. There is a link between a function node and a variable node in the bipartite graph if the function node depends on the variable node. In a factor graph, a variable node represents the actions that an agent can take, and a function node computes utility values for all possible actions based on the dependency between the function node and the variable nodes. There are several distinct factor graph representations of the same problem, e.g., interaction-based factor graph and utility-based factor graph, where the choice of each impacts the performance of the algorithm inferencing in the factor graph. More details about this topic is discussed in (Farinelli et al., 2014). This dissertation applies utility-based factor graph representation in solving the underlying DCOP since it has been proven to be a powerful technique in decentralized coordination (e.g., Zhang & Zhao, 2014; Delle Fave et al., 2012; Yedidsion et al., 2014; Zivan & Peled, 2012; Stranders et al., 2009; Farinelli et al., 2008; Kok & Vlassis, 2006;). Figure 13 illustrates three different DCOP representations of the diagram showed in Figure 12 with three agents where there is a similarity between each pair of agents.

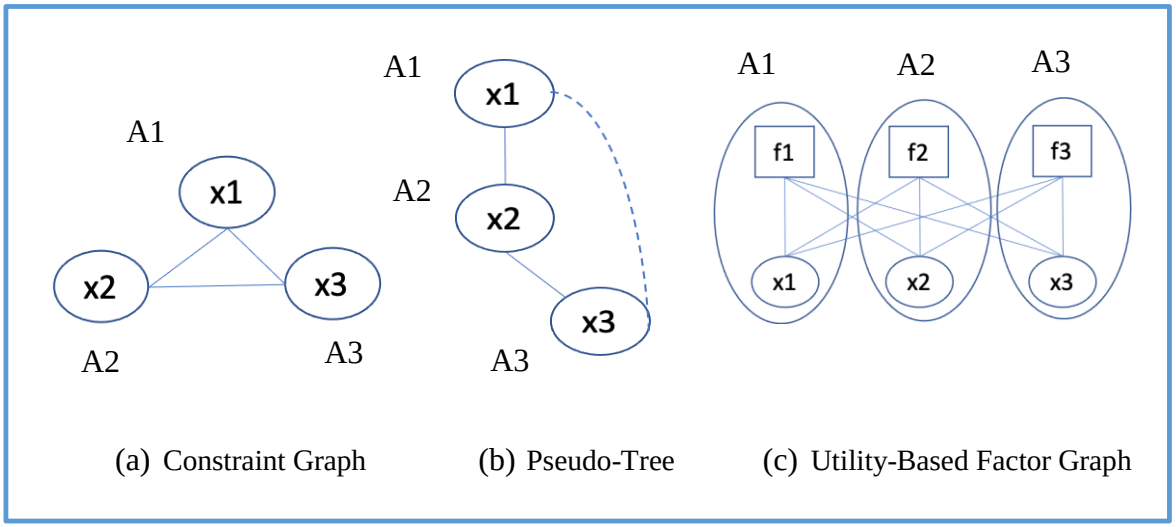


Figure 13 Different DCOP representations

Now, the question is: How to solve the underlying DCOP? Basically, there exist two classes of algorithms for solving the DCOP: *complete* where the optimality of the solution is guaranteed; and *incomplete* where a near-optimal solution can be obtained, i.e., the algorithm executes in a shorter time at the expense of solution quality. Each class of algorithms can be categorized as *search-based* algorithms or *inference-based* algorithms. In search-based algorithms, popular search techniques (e.g., best-first search and depth-first search) will be applied to explore the solution space; on the other hand, inference-based algorithms apply belief propagation technique in which the agents reduce the size of the problem by exploiting the structure of the constraint graph and aggregating costs form their neighbors. Figure 14 illustrates a taxonomy of algorithms along with examples for solving the classical DCOP.

Synchronous branch-and-bound (Hirayama & Yokoo, 1997) is a complete search-based algorithm and can be considered as a distributed version of branch-and-bound algorithm. Another example of complete search-based algorithm is ADOPT (Asynchronous Distributed OPTimization) (Modi et al., 2005) that can be considered as a distributed version of a memory-

bounded best-first search algorithm. DPOP (Distributed Pseudo-tree Optimization Procedure) (Petcu & Faltings, 2005) is an example of complete inference-based algorithm that uses a depth-first search to search the pseudo-tree constructed by the agents. On the other hand, DSA (Distributed Stochastic Algorithm) (W. Zhang et al., 2005) is an incomplete search-based algorithm in which each agent stochastically decides to take a value with maximum gain or other values with smaller gains. Finally, Max-Sum (Farinelli et al., 2008) is an incomplete inference-based algorithm, performing inference on a factor graph using belief propagation approach. It iteratively exchanges messages between variable nodes and factor nodes on a factor graph to optimize an objective function. Interested readers are referred to the survey paper by (Fioretto et al., 2018) for a detailed discussion about the characteristics of these algorithms.

While complete algorithms guarantee to find a global optimal solution to the underlying optimization problem, they grow exponentially in size when solving a large-scale optimization problem. As already mentioned, DCOP is an NP-hard problem; hence complete algorithms are not appropriate for solving it, especially when size of the problem is large. Also, complete algorithms such as ADOPT or DPOP require some preprocessing steps (e.g., constructing pseudo-tree) before executing the algorithm, which means these algorithms are not suitable for dynamic environments where agents are added to or removed from the network because they need to construct the pseudo-tree again and re-solve the optimization problem. Contrarily, incomplete algorithms are applicable to real-time decision-making for large-scale DCOP by providing good quality solutions, but there is no guarantee on the solution quality, and executing in a short run time. Applying max-sum algorithm on utility-based factor graph has been proven to be a powerful technique for solving DCOP along with handling dynamic environments where agents' neighbors can change over time (Waldock et al., 2008; Wang et al., 2009; Farinelli et al., 2014; Yedidsion et al., 2014). This is the

reason that this dissertation adopts max-sum algorithm for solving DCOP that is already formulated and represented by a factor graph. The next section explains how vehicle agents coordinates their actions for serving requests in each subproblem using max-sum algorithm. Before going into the details of the algorithm, it is worthy of note that this dissertation calls this algorithm *min-sum* because the goal is to *minimize* a cost function instead of maximizing a gain that is the case in max-sum algorithm.

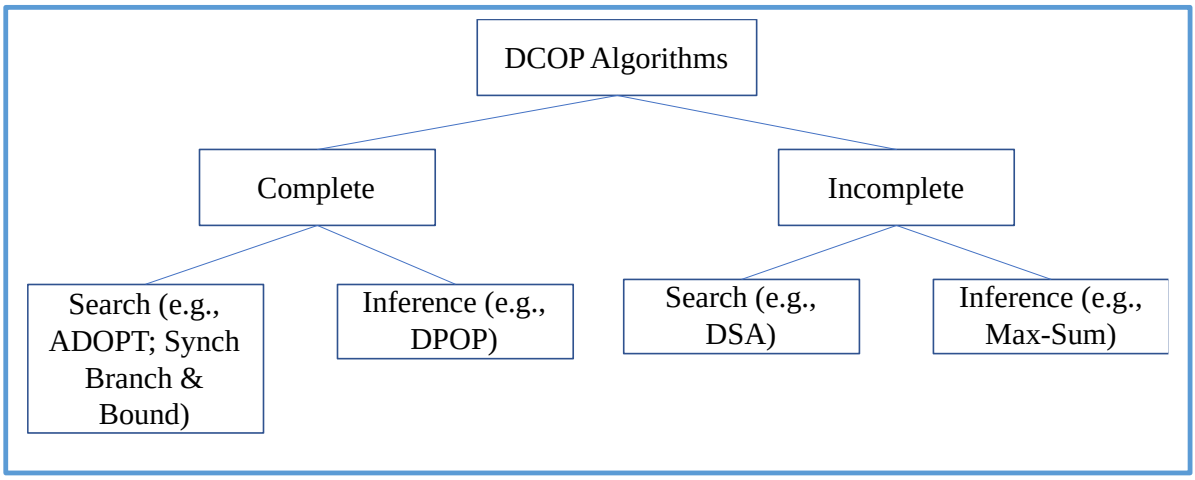


Figure 14 Taxonomy of DCOP algorithms

4.3.5.2 Min-Sum Algorithm

The goal in each subproblem is to find a set of variable assignments that minimizes the objective function (total travel distance in each partition). Making use of factor graph, the objective function in (4.17) is decomposed into *g* factors (functions) where g is the number of vehicle agents in each subproblem. Each individual function (in this work, open-loop TSP) represents the utility of an agent and the sum of the functions shows the objective function. For example, (4.18) is the factorization of the objective function between three agents in Figure 13 (c) in which each factor

depends on all three variables (x_1,x_2,x_3) since there is a similarity between each pair of agents as shown in Figure 13 (a).

In min-sum algorithm, each agent owns one variable node and one function node to perform computation. In other words, each agent is responsible for allocating values to its own variable (e.g., allocating values to variable x_1 by agent a_1), receiving messages from its function and variable nodes, and updating messages that flow out of its function and variable nodes. In min-sum algorithm, each agent continuously negotiates with its neighbors to decide about the best possible joint action that minimizes the sum of the agents' functions.

$$F(x_1,x_2,x_3) = f_1(x_1,x_2,x_3) + f_2(x_1,x_2,x_3) + f_3(x_1,x_2,x_3) \quad (4.18)$$

To ensure that the neighboring agents do not serve the same request in each partition, (4.17) is reformulated to (4.19) as sum of *utility* functions where each agent's utility is defined as follows (4.20) and (4.21):

$$argmin_x \sum_{k \in g} U_k(x_k) \quad (4.19)$$

$$U_k(x_k) = f_k(x_k) + \sum_{i \epsilon N_k \backslash k} x_k \otimes x_i \quad (4.20)$$

Where:

$$x_j \otimes x_i = \begin{cases} \infty & if\, x_j = x_i \\ 0 & otherwise \end{cases} \quad (4.21)$$

(4.19) finds a set of variable assignment between agents that minimizes the sum of agents' utility functions; in other words, the joint action between agents (which request should be served by which agent) that minimizes the total travel distance in each partition. In (4.20), N_k is the set of agents that are neighbors of agent k.

After formulating the decomposable objective function in each partition, as seen in (4.19)-(4.21) and representing it as a factor graph, the min-sum algorithm will be applied on the factor graph to find an optimal joint action between agents by passing messages from variable nodes to

function nodes, as seen in (4.22), and from function nodes to variable nodes, as seen in (4.23). These messages are defined as:

$$q_{i\to j}(x_i) = \sum_{k\epsilon M_i\setminus j} r_{k\to i}(x_i) \qquad (4.22)$$

$$r_{j\to i}(x_i) = min_{x_j\setminus i}[U_j(x_j) + \sum_{k\epsilon N_j\setminus i} q_{k\to j}(x_k)] \qquad (4.23)$$

In (4.22), M_i is a set of function nodes that are connected to variable node i. In (4.23), N_j is a set of variable nodes that are connected to function node j. Note that U_j in (4.23) is computed from (4.20). As an example based on the factor graph represented in Figure 13 (c), the message from function node 3 to variable node 3 is computed by

$$r_{3\to 3}(x_3) = max_{x_1,x_2}[U_3(x_3) + \sum_{k\epsilon\{x_1,x_2\}} q_{k\to j}(x_j)]$$

and the message from variable node 1 to function node 1 is computed by

$$q_{1\to 1}(x_1) = r_{2\to 1}(x_1) + r_{3\to 1}(x_1).$$

Algorithm 10 presents the operations that each vehicle agent performs to implement min-sum algorithm. Each vehicle agent receives messages from neighboring vehicle agents (Q and R), computes the messages accordingly (variable to function and function to variable according to (4.22) and (4.23), respectively), sends the messages to the neighboring vehicle agents, and finally updates its current value. Each vehicle agent repeats the former steps for a number of iterations or until there is no change in the current value of x (variable assignment for the vehicle agent). The main reason for iteration is that the environment might be dynamic in such a way that some requests might be cancelled, or some vehicle agents might be incapable of serving requests during the decision-making process. Algorithms 10-12 are adopted from (Farinelli et al., 2014).

Algorithm 10: Assignment Algorithm: Min-Sum Algorithm

Input: the set of received variable to function message Q, the set of received function to variable message R

Output: Assigning a value to x

Q <- {}

R <- {}

while termination condition is not met do

for $k \in N_j$ do

$r_{i->k}(x_k)$ <- compute message from function node i to variable node k (Algorithm 11)

send message $r_{i->k}(x_k)$ to neighboring vehicle agent a_k

end for

for $k \in M_i$ do

$q_{i->k}(x_i)$ <- compute message from variable node i to function node k (Algorithm 12)

send message $q_{i->k}(x_i)$ to vehicle agent a_k

end for

Q <- get message from neighboring function nodes

R <- get message from neighboring variable nodes

Update current value of x

end while

return x

Time complexity is $O(td^g)$ (Fioretto et al., 2018) where d, g, and t are the size of the largest domain in each partition, maximum number of vehicle agents in each partition, and number of iterations in the algorithm, respectively.

Algorithm 11: Compute Message from Function Node j to Variable Node i According to Eq. 4.23

Input: the receiver's variable x_i, the sender's utility function U_j, Q as the current set of variable to function messages received by the sender j

Output: $r_{j->i}(x_i)$ that is a column vector with the size of D_i

$r_{j->i}(x_i) <- \infty$

for $d_i \in D_i$ {all joint assignments of x_i} do

  $t <- U_j(d_i)$

  for $d_k \in D_{Nj}$, $(k \neq i)$ do

    $t <- t + q_{k->j}(d_k)$ { $q_{k->j} \in Q$}

    $t <- U_j(d_i)$

  end for

  $r_{j->i}(d_i) <-$ min t {$t \in D_{Nj}$}

end for

return $r_{j->i}(x_i)$

Algorithm 12: Compute Message from Variable Node i to Function Node j According to Eq. 4.22

Input: the sender's variable x_i, the receiver's function U_j, R as the current set of function to variable messages received by the sender i

Output: $q_{i->j}(x_i)$ that is a column vector with the size of D_i

$q_{i->j}(x_i) <- 0$

for $r_{k->i} \in R$ $(k \neq j)$ do

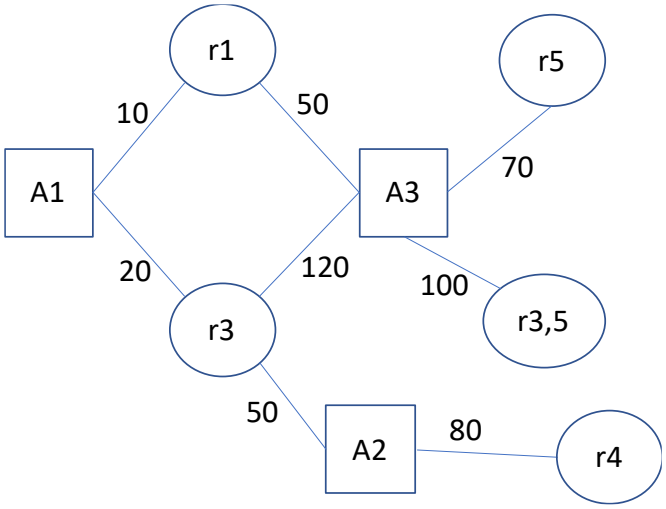
  $q_{i->j}(x_i) <- q_{i->j}(x_i) + r_{k->i}(x_i)$

end for

return $q_{i->j}(x_i)$

To summarize this section, an example with three interacting agents is presented to illustrate how the Algorithms 10-12 operate to compute the messages and assign a value to each agent's variable. Figure 15 shows the assignment problem in a partition where there are three interacting agents, Agent 1 and Agent 2 with functional capacity of one and Agent 3 with functional capacity of two, and four requests. The aim is to find a joint action (maybe optimal) that minimizes the total travel distance, $\sum_{k=1}^{3} U_k(x_k) = U_1(x_1, x_2, x_3) + U_2(x_1, x_2, x_3) + U_3(x_1, x_2, x_3)$, for serving the requests in this partition. Note that the utility of each agent (4.20) and (4.21) depends on the owner's agent and two other agents because there is a similarity between each pair of agents. Based on the DCOP formulation presented in the previous section, the domain of three variables and function values for the agents are specified as shown in Figure 15 panels (b) and (c).

As an illustration of how Algorithms 10-12 compute the messages, Table 4 presents some messages exchanged by the agents at Time 0 and Time 1. The messages are computed by (4.22) and (4.23).



(a) The Assignment Problem	(b) Variables' Domains	(c) Function Values
	$D_1 = \{r_1, r_3\}$ $D_2 = \{r_3, r_4\}$ $D_3 = \{r_1, r_3, r_5, r_{3,5}\}$	$f_1(x_1) = [10,20]$ $f_2(x_2) = [50,80]$ $f_3(x_3) = [50,120,70,100]$

Figure 15 (a) An example of assignment problem in a partition (b) domain of each agent's variable (c) function values of each agent

Table 4 An example of exchanging messages from function to variable and from variable to function

Time 0	Time 1
$q_{1\to1}(x_1) = q_{2\to1}(x_1) = q_{3\to1}(x_1) = \begin{bmatrix}0\\0\end{bmatrix}$ $q_{1\to2}(x_2) = q_{2\to2}(x_2) = q_{3\to2}(x_2) = \begin{bmatrix}0\\0\end{bmatrix}$ $q_{1\to3}(x_3) = q_{2\to3}(x_3) = q_{3\to3}(x_3) = \begin{bmatrix}0\\0\\0\\0\end{bmatrix}$ $r_{1\to2}(x_2) = min_{x_1,x_3}[U_1(x_1,x_2,x_3) + q_{1\to1}(x_1) + q_{3\to1}(x_3)] = \begin{bmatrix}10\\20\end{bmatrix}$	$q_{2\to1}(x_2) = \begin{bmatrix}50\\80\end{bmatrix}$ $q_{3\to1}(x_3) = \begin{bmatrix}50\\120\\70\\100\end{bmatrix}$ $r_{1\to1}(x_1) = min$ $\begin{bmatrix}10+\infty+50+50, 10+\infty+50+120,\\ 10+50+70, 10+\infty+50+100,\\ 10+\infty+80+50, 10+80+120,\\ 10+80+70, 10+80+100;\\ 20+\infty+50+50, 20+\infty+50+120,\\ 20+\infty+50+70, 20+\infty+50+100,\\ 20+80+50, 20+80+120,\\ 20+80+70, 20+\infty+80+100\end{bmatrix}$

The variable assignments for agent 1 computed by $r_{1->1}$ at Time 1 would be as follows:

x1=r1 x2=r3 x3=r1**,** x1=r1 x2=r3 x3=r3, x1=r1 x2=r3 x3=r5, x1=r1 x2=r3 x3=r3,5, x1=r1 x2=r4 x3=r1, x1=r1 x2=r4 x3=r3, x1=r1 x2=r4 x3=r5, x1=r1 x2=r4 x3=r3,5;

x1=r3 x2=r3 x3=r1**,** x1=r3 x2=r3 x3=r3, x1=r3 x2=r3 x3=r5, x1=r3 x2=r3 x3=r3,5, x1=r3 x2=r4 x3=r1, x1=r3 x2=r4 x3=r3, x1=r3 x2=r4 x3=r5, x1=r3 x2=r4 x3=r3,5

From all possible actions computed in $r_{1->1}$, it can be observed that best action, shown in green, with total travel distance of 190 would be as follows: serving request 1 by agent 1, request 4 by agent 2, and requests 3 and 5 by agent 3. Note that two other joint actions, shown in blue, are also possible, which have total travel distance less than the selected joint action, but they are rejected. The reason is that the joint actions in blue are serving only three requests out of four while in the selected joint action all requests are served. At the end, each agent by executing Algorithms 10-12 converges to this assignment.

5.0 Evaluation

After proposing the decentralized approach, formulating the main optimization problem, designing algorithms for decomposing the main problem and allocating requests to each subproblem, formulating each subproblem as DCOP and designing algorithms for solving each subproblem in Chapter 4, now this chapter presents an empirical evaluation of the formalism and the designed algorithms through a simulation. This dissertation considers two metrics for evaluating the performance of the proposed decentralized approach: (a) solution quality in terms of measuring total travel distance of vehicle agents and (b) running time of the algorithms by measuring CPU time. These two metrics are used to benchmark the proposed approach against two alternative approaches: (a) in the first approach, a lower bound to the original optimization problem is obtained by formulating the problem as ILP and providing an exact solution via CPLEX and (b) a greedy heuristic algorithm is designed to solve the original optimization problem.

5.1 Data for Experiments

The proposed decentralized approach is tested on a synthetic data rather than real data because at this time the synthetic data meets the needs of this work in terms of testing the developed models and algorithms and the theories behind them. Furthermore, with synthetic data, coordination graph with different structures (best case, average case, worst case) can be created and examined while in using real-world data, there might not exist such flexibility.

Five variables are considered in creating test cases to test the performance of the proposed approach: (a) number of vehicle agents (b) capacity of each vehicle agent (c) number of requests (d) maximum number of vehicle agents in each subproblem(g), and (e) time interval, elapsed time from the initialization. The simulator (Section 5.2) considers these five variables and creates five sets of test cases that represent variant structures of the coordination graph. These sets are:

- 50 instances of the problem with 10, 15, 20, 25, and 50 random requests where number of vehicle agents is 5, capacity of each vehicle agent is 3, time interval is 155 seconds, and g is 2.
- 30 instances of the problem with vehicle agents' capacity of 2, 3, and 4 where number of vehicle agents is 15, time interval is 155 seconds, g is 5, and number of random requests is 20.
- 40 instances of the problem with 50, 100, 150, and 300 vehicle agents where capacity of each vehicle agent is 2, time interval is 155 seconds, g is 5, and number of random requests is 110.
- 70 instances of the problem with setting g to 5, 10, 15, 20, 25, 30, and 50 where number of vehicle agents is 300, capacity of each vehicle agent is 2, time interval is 155 seconds, and number of random requests is 100.
- 40 instances of the problem with time interval of 30, 50, 90, and 150 seconds where number of vehicle agents is 300, capacity of each vehicle agent is 2, g is 10, and number of random requests is 100.

5.2 Simulator

A simulator is developed using Java programming language on a laptop with 8 GB RAM and 2.9 GHz Core i5 CPU. In the simulator, a number of vehicle agents move on a grid with a specific speed (Figure 16). At time zero (initialization time), all vehicle agents are at the origin of the grid that is at the bottom left of the grid. Vehicle agents are allowed to take one of the four actions (right, left, up, down) in each movement if they stay in the grid. Each vehicle agent has the following properties in the system: capacity, occupancy with number of onboard passengers, speed, path as a set of cells that the vehicle agent should traverse on the grid, and schedule as a set of pick_up and/or drop_off cells.

The shortest distance on the grid is computed by Manhattan distance. A specific number of requests are generated randomly on the grid in each round with the following properties: pick_up cell, drop_off cell, time window for both cells, current time (time of request). To compute bids in each round, each vehicle agent finds an exact solution to open-loop TSP problem, satisfying the constraints of the new requests and onboard passengers. In other words, each vehicle agent is responsible for optimizing its own local schedule by solving an instance of open-loop TSP. Since the capacity of vehicle agents in this simulation is at most four, an exact solution is obtained to the TSP problem. By increasing vehicle agents' capacity, heuristic algorithms can be applied for solving open-loop TSP.

To be consistent with the works reported in the literature, for example, the models and algorithms developed for DARP, this dissertation experimented with a grid size of 20 x 20 cells. Also, in the simulator, the speed of each vehicle agent is 1m/s and time window for pick-up and drop-off is 130s.

The following assumptions are made for the experiments in this simulator:

- There is no wall in the grid.
- Number of passengers in each request is one.
- Number of requests and number of empty seats at time zero are equal.
- Vehicle agents remain idle at their location when they are done with serving the requests.

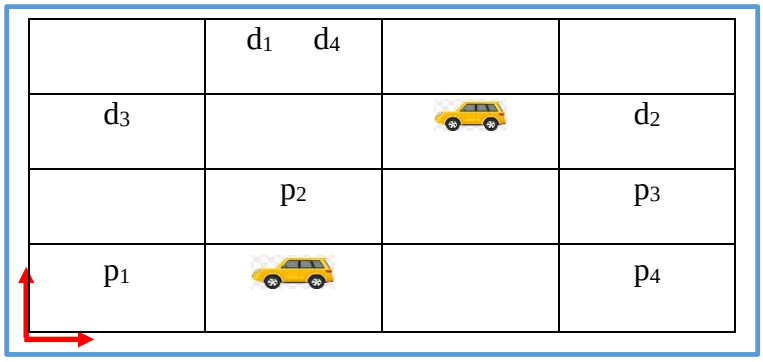


Figure 16 An example of a grid with two agents and four requests

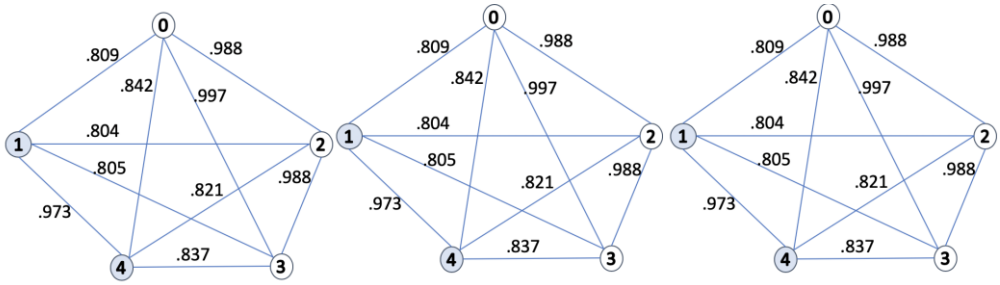
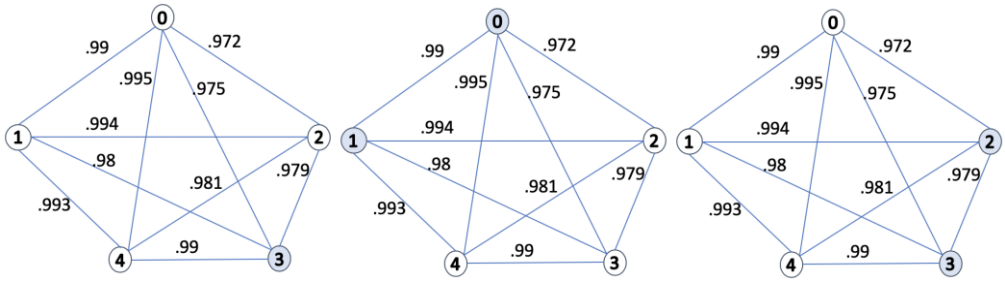
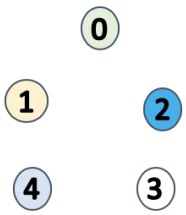
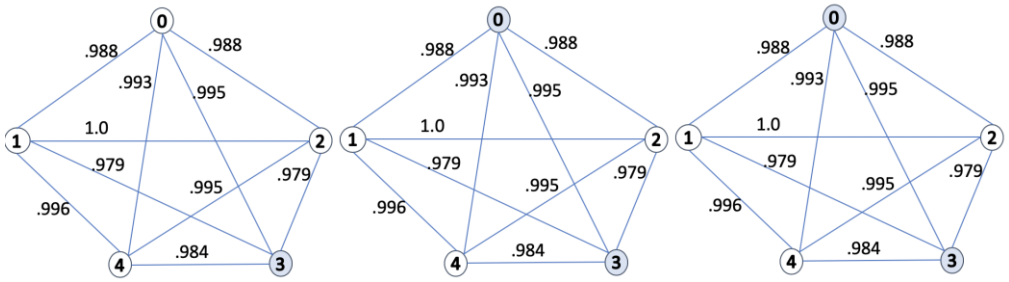
5.3 Validation and Evaluation Results

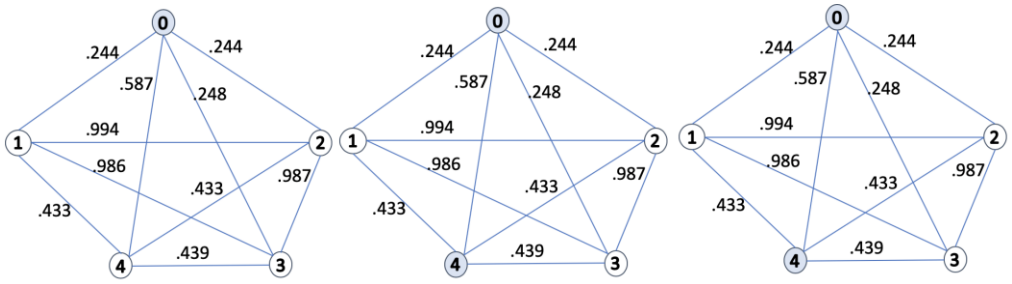
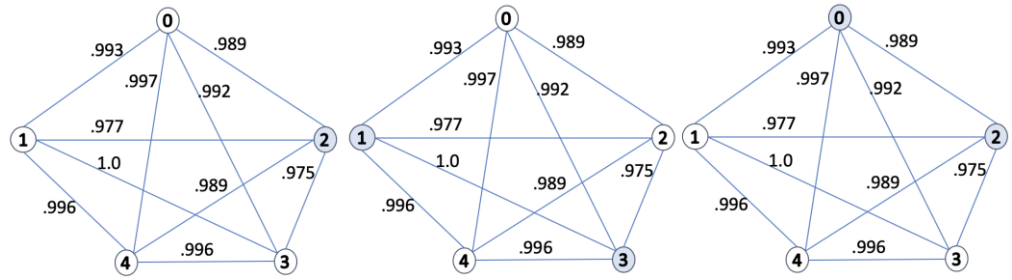
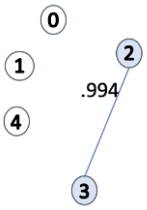
Before discussing the algorithms designed for evaluation and presenting the empirical results to confirm the workability of the proposed multi-agent approach, it is important to see how different values of time interval in the simulator affect the structure of the coordination graph. The shorter the time interval, the sparser the coordination graph. The intuition behind this is that when the time interval is too short, most or all vehicle agents are full, so they cannot serve any new request and there is no similarity between vehicle agents. i.e., there are isolated nodes in the graph.

Table 5 illustrates coordination graphs with different structures (complete graph, sparse graph, edgeless graph) when two variables, time interval and number of requests, are changed in the simulator. The range of time interval is between 13s and 223s where the number of requests is

5 and 10. In this part of simulation, the number of partitions is set to two (K=2) since there are only five vehicle agents in the fleet. The decomposition algorithm divides vehicle agents based on three different spectral clustering algorithms (See Chapter 4). Firstly, the algorithm decomposes the graph from an unnormalized Laplacian matrix (the antepenultimate graph). Secondly, the algorithm considers normalized Laplacian for decomposition, inspired by the algorithm presented in (Shi & Malik, 2000) (the penultimate graph). Thirdly, the decomposition algorithm uses normalized adjacency matrix and divides vehicle agents based on the algorithm developed by (Ng et al., 2002) (the ultimate graph). An isolated node in a graph means that there are no similarities between the vehicle agents and the algorithm considers it as a separate partition.

Table 5 Decomposition of five vehicle agents with different structures and similarities and with varying time intervals and number of requests. (TI=Time Interval, CG=Coordination Graph)

Req	TI (s)	CG
5	113	
	43	
	13	
	223	

10	73	
	153	
	33	

5.3.1 Validation

As mentioned before, this dissertation considers two metrics for validation: solution quality and running time. Using these two metrics, the set of proposed algorithms in the decentralized approach is benchmarked against two alternative solution approaches: optimum solution obtained by ILP and greedy heuristic solution.

To formulate the original optimization problem as ILP, this dissertation adopts the mathematical formulation presented by (Alonso-mora et al., 2018), which is defined as follows:

$$Min\ [\sum_{i\in\{agents\}}\sum_{j\in\{dimensions\}} a_i x_{ij} + \sum_{k\in\{requests\}} c y_k] \quad (5.1)$$

$$S.T. \begin{cases} y_{k\in\{requests\}} + \sum_{i\in\{agents\}}\sum_{j\in\{dimensions\}} x_{ij} = 1 & (5.2) \\ \sum_{j\in\{dimensions\}} x_{ij} \leq 1 & (5.3) \\ x_{ij} \in \{0,1\} & (5.4) \\ y_k \in \{0,1\} & (5.5) \end{cases}$$

The goal in the objective function (5.1) is to minimize both the total travel distance of the vehicle agents and number of rejected requests. The set *dimensions* in the objective function refers to the columns of the agent_request matrix presented in Table 3. a_is show the elements of the agent_request matrix and c is a constant (c>0). The constraints of this ILP formulation are to ensure that: each request is served by exactly one vehicle agent or rejected, as seen in (5.2); each vehicle agent serves at most one dimension (5.3); and decision variables are binary. The presented ILP is solved by CPLEX 12.10 to obtain an optimum solution to the problem.

Error! Reference source not found. shows an overview of how the experiments are c onducted in the simulator. There are four variables (number of vehicle agents, number of requests, vehicle agents' capacity, time interval) in the simulator on which the agent_request matrix are constructed. The agent_request matrix is the input to two different scenarios as follows:

- The agent_request matrix is an input to ILP, as formulated in (5.1)-(5.5), and the result would be an optimal solution to the problem.

- The agent_request matrix is an input to the decentralized approach in which the matrix is decomposed into k partitions by using three different decomposition methods presented in Chapter 4 (unnormalized Laplacian, normalized Laplacian presented by Shi & Malik, 2000, and normalized Laplacian presented by Ng et al. 2002). Then requests are allocated to each partition using Algorithm 9. In the next step, the assignment problem in each partition will be solved using min-sum algorithm (Algorithms 10-12). Then the final solutions form partitions are aggregated and compared to the solution provided by ILP.

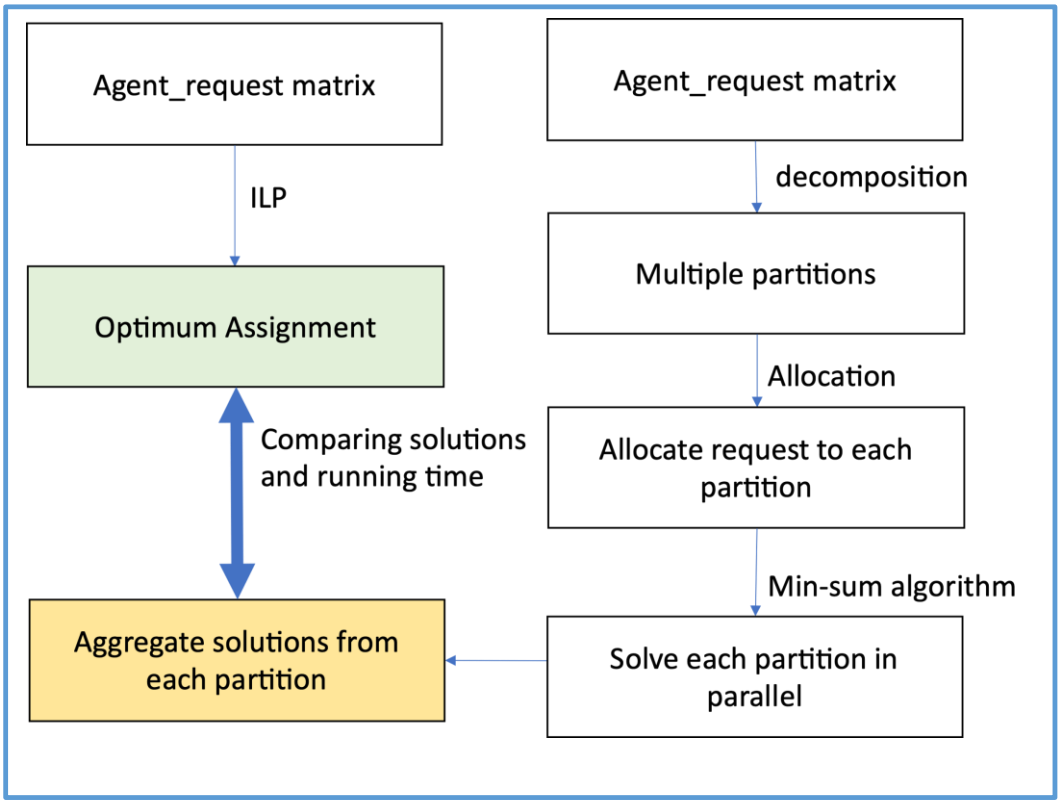


Figure 17 An overview of the experiment design

The second approach for validation is performed by developing a greedy heuristic algorithm as presented in Algorithm 13. The algorithm processes the requests in the pool individually based on first-come-first-served scheme. There are alternatives to processing the requests one-by-one in a chronological order, such as selecting one request randomly at each time. Since Algorithm 13 considers all vehicles in each round (serving a new request) unless a vehicle is full (i.e., a vehicle can serve more than one request if it has enough empty seat), the order of processing requests does not affect the final solution, neither accuracy nor time performance. In another variant of processing requests one-by-one, each vehicle can serve only one request. In this case, the approaches for serving the requests (e.g., first-come-first-served or random selection) do affect the final solution.

Algorithm 13 computes the bid for each vehicle agent sequentially to serve a request. If there is no vehicle agent that can serve the request because the vehicle agents are full or the constraints of the optimization problem (e.g., time window) cannot be satisfied, the request will be rejected. Otherwise, the request will be assigned to the vehicle agent with the minimum bid. This process will be repeated until there is no request in the pool.

Algorithm 13: Greedy Assignment

Input: set of requests R, set of vehicle agents with current trajectories and schedules A

Output: assignment of vehicle agents to requests for serving them B

```
for each request in R
  best bid <- positive infinity
  for each vehicle agent in A
    current bid <- compute bid for the vehicle agent if it is eligible for serving the request
```

```
        if current bid is less than the best bid
          update best bid
        end if
      end for
      if best bid is less than positive infinity
        add best bid to B
      end if
    end for
    return B
```

Time complexity is O(Rm(p!)) where R, m, and p are the number of independent requests in the pool, the number of vehicle agents, and the number of points in the schedule of a vehicle agent, respectively. Note that the reason that the time complexity has a factorial term is that an exact solution is provided to compute each bid as an instance of open-loop TSP.

In Algorithm 13, as one variant of the centralized approach, the vehicles can compute the bids in parallel and independently without knowing about the other vehicles' bids. This parallelization makes it easy to handle dynamism, i.e., adding/removing vehicles to/from the system during the decision-making process, in the ridesharing system to some extent. However, finding a solution to the assignment problem (which vehicle should serve which request) in the centralized approach requires solving the optimization problem from scratch. On the other hand, any change (e.g., add/remove vehicles) in the proposed decentralized approach does not require solving the optimization problem from scratch; instead, it only needs to repair the initial solution in the partition (subproblem) within which the change has happened. It is worth mentioning that this capability is a reference to the flexibility feature of the proposed approach.

5.3.2 Evaluation Results

This section presents the experiments performed in the simulator using the five sets of test cases described earlier. The empirical results are summarized in Table 6. Since requests are generated randomly on the grid, to have a better estimation of the metrics, i.e., objective value (*OV*) and running time (*RT*), each test case (each row in the table) is run 10 times in the simulator. In this experiment, number of vehicle agents, vehicle agents' capacity, time interval, parameter g, and number of requests are the variables that take different values. The reason for choosing different values for g is to observe how different number of nodes in each partition affects the solution quality and execution time of the proposed decentralized approach. Columns *OPT* and GR show the results from ILP and greedy heuristic algorithm, respectively. Columns *DM1*, *DM2*, and *DM3* show the results from three different decomposition methods, i.e., unnormalized Laplacian, normalized Laplacian presented by Shi & Malik (2000), and normalized Laplacian presented by Ng et al. (2002).

Table 6 Experimental Results (TI=Time Interval, GR=Greedy, RT=Running Time, OV=Objective Value)

Agents	Cap	TI	g	Req	→	OPT	DM1	DM2	DM3	GR
5	3	155	2	10	RT	1.0	0.75	0.73	0.73	0.18
					OV	130	183	190	190	376
				15	RT	1.83	1.56	1.52	1.52	0.34
					OV	200	288	286	283	597
				20	RT	3.79	3.2	3.15	3.15	0.23
					OV	235	293	295	295	620

				25	RT	6.56	5.02	4.99	4.97	0.26
					OV	182	263	268	268	617
				50	RT	137.35	55.6	57.1	57.01	0.24
					OV	117	291	293	291	605
15	2	155	5	20	RT	0.71	0.39	0.35	0.33	0.23
					OV	253	375	405	396	571
	3				RT	8.53	7.13	7.08	7.07	0.19
					OV	226	361	371	370	612
	4				RT	1519.21	1518	1518.17	1518	0.27
					OV	68	107	107	105	663
50	2	155	5	110	RT	79.55	22.65	23.14	22.96	0.33
					OV	1028	2065	2166	2161	3049
100					RT	170.8	46	46.83	46.4	0.38
					OV	1098	2055	2303	2326	3062
150					RT	246.33	66.37	67.72	67.32	0.46
					OV	1105	1960	2329	2356	2957
300					RT	531.68	171.79	143.83	142.8	0.66
					OV	1077	1966	2256	2239	2760
300	2	155	5	100	RT	358.12	106.02	108.31	107.07	0.66
					OV	982	1739	2013	2030	2475
			10		RT	357	106.57	109.73	106.63	0.72
					OV	956	1712	2071	2094	2605
			15		RT	361.72	113.59	114.63	135.02	0.67

					OV	968	1677	2124	2160	2642
			20		RT	354.41	105.83	107.99	107.27	0.64
					OV	956	1673	2120	2135	2647
			25		RT	345.55	107.73	109.71	107.76	0.63
					OV	967	1673	2119	2079	2593
			30		RT	337.7	103.66	105.49	106.14	0.62
					OV	983	1685	2049	2052	2565
			50		RT	345.52	107.65	108.71	108.7	0.65
					OV	943	1518	1794	1732	2606
300	2	30	10	100	RT	6.62	6.62	6.36	5.83	0.58
					OV	2370	2628	2921	3469	2954
		50			RT	64.96	22.43	21.95	22.27	0.83
					OV	1044	2008	1958	1650	3046
		90			RT	313.75	97.54	100.22	98.82	0.66
					OV	971	1702	2093	2071	2521
		150			RT	356.65	107.89	110.21	109.7	0.69
					OV	984	1733	2145	2111	2534

Figure 18 and Figure 19 respectively illustrate the objective value (total travel distance) and run time of the algorithms proposed in the decentralized approach using three different decomposition methods, ILP, and greedy heuristic algorithm. In each graph, the values of one variable out of five change to see how this variable affects the solution quality and run time.

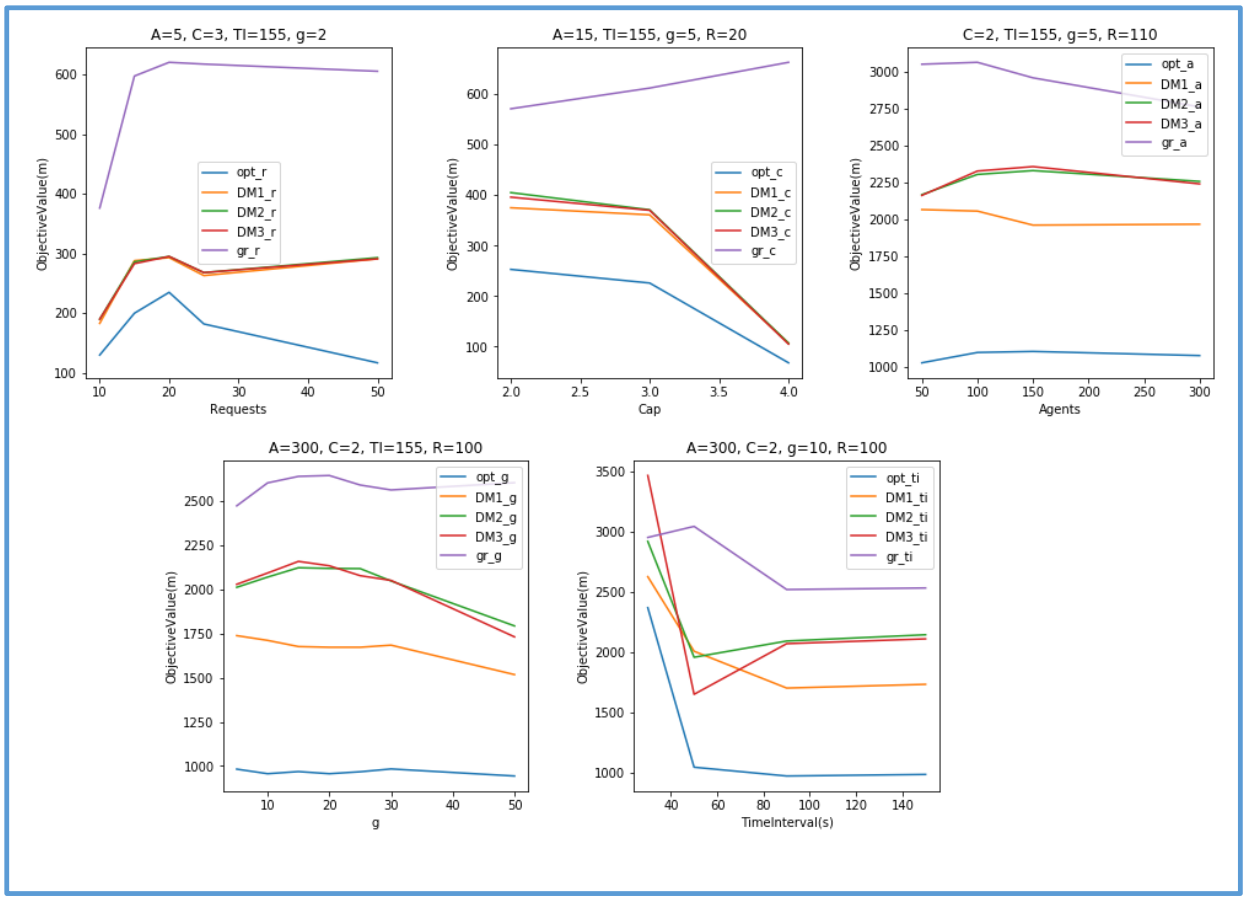


Figure 18 Objective values obtained from ILP, Greedy and three decomposition methods

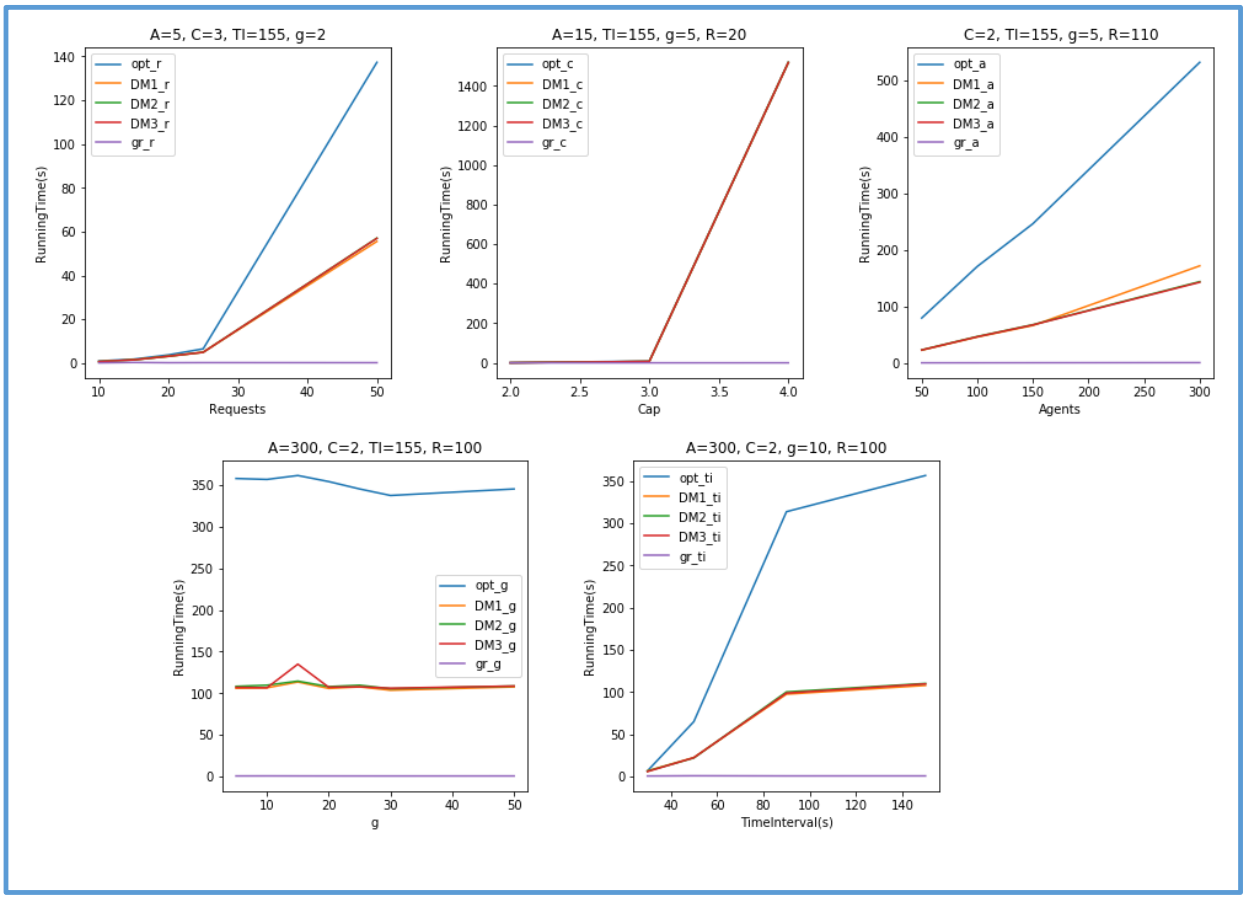


Figure 19 Running time obtained from ILP, Greedy and three decomposition methods

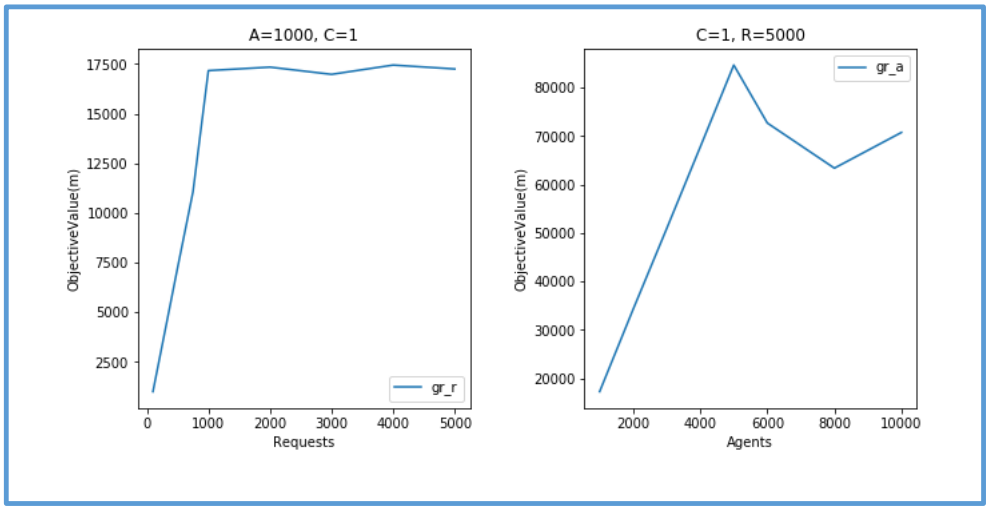


Figure 20 Objective value obtained from Greedy algorithm with different number of requests and agents

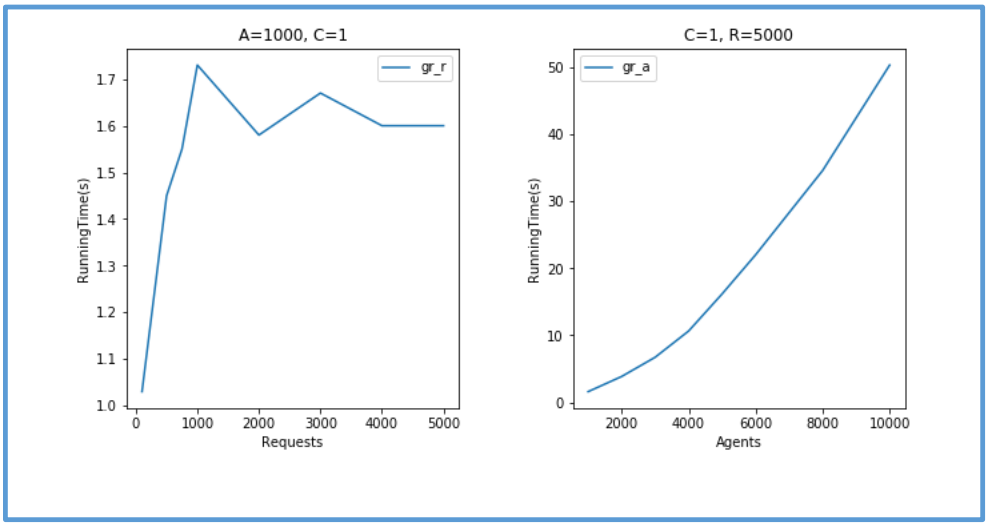


Figure 21 Running time obtained from Greedy algorithm with different number of requests and agents

Based on the five sets of test cases created in the simulator, the greedy heuristic algorithm has a very fast execution time (almost constant) as shown in Figure 19. To have a more precise estimation of the performance of the greedy heuristic algorithm, different test cases (as shown in Figure 20 and Figure 21) with larger number of requests and vehicle agents are tested in the simulator. It can be observed that the execution time of the greedy algorithm grows as number of vehicle agents or number of requests changes from a few hundreds to a few thousands.

5.4 Analysis of Evaluation Results

This section analyses the results of the experiments conducted in the previous section and illustrated in Table 6, Figure 18, and Figure 19.

a) The running time in the distributed approach is less than the centralized one in all experiments. The bigger the size of the problem (by increasing capacity, number of

vehicle agents, number of requests), the more significant the gap between running times of the two approaches (decentralized and ILP). However, by increasing size of the problem, the gap between the optimal solution and local solution increases.

b) According to (Von Luxburg, 2007), the statistical analysis on both normalized and unnormalized Laplacian substantiated that normalized Laplacian provides better solution. However, in my experiment, it can be observed that DM1 (unnormalized Laplacian) outperforms the other two decomposition methods in most test cases in terms of providing better solutions while the running times of different decomposition methods are almost similar. It should be noted that the running time in the decentralized approach is the highest running time between the running times of all subproblems.

c) It can be observed that by increasing g, better solution can be obtained. This observation ratifies the intuition behind it, i.e., when g has a bigger value, the number of partitions decreases (less decomposition) and the search space is more similar to the original search space before decomposition.

d) Decreasing time interval allows us to have a combination of isolated nodes and connected nodes in the coordination graph. However, if the time interval is too short, it is highly probable that the system does not provide any service because all the vehicle agents are full.

e) Almost all the above test cases can be considered as worst-case scenarios (see the values of time interval in Table 6 and the discussion of Section 5.2), i.e., there is a high level of dependency between vehicle agents which means there are a lot of similar requests that can be served between different vehicle agents in different partitions.

f) The running time includes construction of agent_request matrix and solving the assignment problem using ILP. Note that in the distributed paradigm, the running time for constructing agent_request matrix is much lower than the above experiment. This is because based on the proposed model, agent_request matrix is constructed by the vehicle agents in parallel, but in the above experiment, the matrix is constructed in a sequential manner.

g) In the last set of test cases where the variable TI (time interval) changes, it is observed that the more isolated nodes in the coordination graph (shorter time interval), the better solution can be obtained. More isolated (dissimilar) nodes can be created by decreasing the time interval. It verifies that the distributed paradigm performs better in average-case scenarios (sparse coordination graph, i.e., combination of isolated nodes and connected nodes) in comparison with worst-case scenarios (complete coordination graph).

h) In all test cases, the greedy algorithm performs very fast, but its solution quality falls behind the one obtained by two other approaches. The main reason is that in the greedy approach, the requests are processed separately, but better solution can be obtained when considering requests in batch assignment (see Section 2.1.4 for more detail).

i) There could be an extreme case where similarity between all vehicles agents is the same. If this case occurs, mathematically speaking, the cosine similarity between all vehicles agents in the agent_request matrix is one, which means all vehicles agents have exactly similar trajectory and schedule, and serve new requests with the same utility values. This case is not simulated in this dissertation because the occurrence of such a case is extremely rare in real world. However, if this case occurs, the

decomposition algorithm partitions the vehicles agents randomly into K partitions because there is no specific criterion for distinguishing vehicles agents from each other.

5.5 Limitation

In the second scenario of the experiment design as shown in **Error! Reference source not found.**, min-sum algorithm solves the assignment problem in each partition in parallel. This is the ideal materialization of the decentralized approach proposed in this dissertation. However, in this dissertation this implementation has not been done due to lack of resources in parallel computing, extra challenges imposed by parallel programming, and steep learning curve. Alternatively, in the second scenario, instead of using min-sum algorithm for solving each subproblem, ILP formulation similar to (5.1)-(5.4) is implemented. Algorithm 14 presents how this implementation has been done in the simulator.

There are some pros and cons between the ideal approach (min-sum algorithm) and the alternative approach (ILP) in solving each subproblem. The advantage with min-sum algorithm is that it can handle changes in the environment dynamically, e.g., some requests might be canceled, or some vehicle agents might be out of order while in the ILP, the problem should be re-solved once a change occurs in the environment.

On the other hand, considering all combinations of variables' assignments in min-sum could be a hurdle in getting an optimal solution when number of vehicle agents and/or size of the domains grows in each partition. However, some pruning techniques can be applied in min-sum to reduce the fraction of the joint action space in each subproblem. Size of the sub-graph (increasing number of partitions) can be another solution to bypass the above hurdle, but it may

come at expense of global optimality. Table 7 compares ILP and min-sum algorithm in terms of formulation, solution quality, and implementation aspect.

Table 7 ILP vs min-sum algorithm

Technique	Formulation	Solution Quality	Implementation
ILP	(5.1)-(5.4)	Exact solution can be obtained	Can be done using existing optimization tools such as Cplex
Min-Sum Algorithm	Algorithms 10-12	Based on the proposed formulation, exact solution can be obtained by considering all combinations of variables' assignments	Can be implemented in a multi-agent environment such as JADE (Java Agent DEvelopment Framework)

Algorithm 14: Optimal Assignment

Input: set of partitions P including a set of vehicle agents and allocated requests in each partition

Output: assignment of vehicle agents to requests in each partition S

for each partition in P

  add the solution obtained by ILP to S

end for

return S

6.0 Summary, Conclusion, and Future Research Direction

6.1 Summary

This dissertation proposed a novel approach to solve dynamic ridesharing problem in a decentralized way. In the proposed approach, the dispatcher agent plays an important role in the system where it constructs the coordination graph between decision makers, decompose the task between the vehicle agents, and allocates requests to each subproblem. Also, there are a set of cooperative agents work together to take an optimal or near-optimal joint action in each subproblem to decide which requests should be served by which vehicle agent. Three different decomposition techniques, using spectral graph theories and graph Laplacian, are considered in the proposed approach. The experiments showed that the decomposition using unnormalized Laplacian outperforms the other two techniques by providing better solution (near optimal solution), but the execution time of the three techniques is similar. As a proof of concept, a simulator was implemented in Java and five sets of test case designed and ran using this simulator. To evaluate the proposed approach, two other approaches, ILP and greedy heuristic, are used where the solution from ILP provided a lower bound to the underlying optimization problem. The greedy heuristic algorithm performed extremely fast in the five sets of test cases designed in the experiments, but its solution quality fell behind the proposed approach and obviously the ILP.

6.2 Conclusion

Dynamic ridesharing, centralized or decentralized, involves real-time decision making. A ridesharing system must be: (a) scalable with respect to different numbers of passengers and vehicles (b) fault-tolerant (robust), and (c) flexible when an unexpected disturbance happens (e.g., vehicle breakdown) in the system. Gaining advantage from computational and communication capacity of each vehicle, the proposed multi-agent approach has all the above features. Firstly, due to the availability of computational resources in each vehicle and the distributed nature of computation, the decentralized approach has the potential to scale up to much larger scenarios in solving the ridesharing problem, i.e., when the number of agents (users and vehicles) and/or their capacities (number of passengers per requests and larger vehicles) within the network increases. Secondly, the process of decision making in the decentralized approach is more robust in comparison with a single agent decision maker in the centralized approach. The dispatcher agent in the decentralized approach might fail, but building fault-tolerant decentralized systems are not as challenging as building fault-tolerant centralized systems. Building fault-tolerant centralized ridesharing systems must handle some computational challenges, incurs time delay due to synchronization between nodes, and requires to check data integrity due to redundant and inconsistent data. The existence of *multiple decision makers* allows us to decompose a large-scale ridesharing problem, which is computationally complex and expensive to be solved by a single agent, into smaller tractable subproblems, and to solve each subproblem by the corresponding decision makers independently. Thirdly, the decentralized approach is flexible, allowing vehicle agents in each subproblem to adjust their decisions locally when an unexpected situation occurs to a vehicle agent, rather than forcing the single-agent decision maker in the centralized approach to re-compute the global solution. Lastly, properly decomposing the coordination graph as well as

designing efficient algorithms to solve the given assignment task in each subproblem make obtaining a good solution in a reasonable amount of time possible.

Table 8 provides an overview comparing the characteristics of the centralized and decentralized approaches.

Table 8 Comparison of the centralized and decentralized approaches.

Features	Centralized approach	Decentralized approach
Vehicle to vehicle communication	No	Mandatory
Robustness	System is down if the central server fails to operate.	System operates even if some decision makers locally fail to function
Scalability	Adding more computing resources with parallelization might help	Highly scalable since each vehicle has computing resources
latency in Exchanging messages	Might happen when vehicles exchange messages to the central server	Might happen when exchanging messages between decision makers or between vehicles agents and dispatcher agent
Flexibility	Highly sensitive to information updates due to costly re-computation	Flexible owing to the computational and communication resource of each decision maker

6.3 Future Research Direction

This dissertation formulated the ridesharing problem as a distributed optimization problem and designed a set of algorithms for task decomposition, request allocation, and decentralized

coordination between cooperative agents, and finally successfully implemented the proposed approach. The work presented in this dissertation can be extended in three main directions. First, the formulation needs to be extended to address the underlying optimization problem in deciding how idle vehicle agents should move in the environment. In other words, should the vehicle agents be idle after serving their requests, or should they move to high-demand areas? In both cases the objective is to serve more requests; however, in the latter case, a negotiation between vehicle agents is needed to prevent some areas from getting overcrowded with vehicle agents while some other areas are vacant.

Second, cast deterministic formulation of the optimization problem presented in this dissertation to a stochastic optimization problem to consider uncertainty in decision making. More precisely, handling action uncertainty in the proposed large-scale DCOP is challenging. One example of uncertainty is when vehicle agents in each subproblem are not completely sure to serve a new request due to lack of fuel.

From the empirical results (Table 6) it can be observed that in every instance of the problem where |R|<=ES, DM1 can find a solution within a factor $\alpha = 2$ of the optimum solution (for DM2 and DM3, this factor is $\alpha = 2 + \varepsilon$); however, there is no guarantee to keep the solution quality within this range for other instances of the problem. Hence, the third direction is to prove theoretically that the proposed approach guarantees this solution quality and as a result, this dissertation would be the pioneer in providing an approximation algorithm to the underlying optimization problem that is NP-hard.

Lastly, the simulator developed in this dissertation can be extended to test the performance of the proposed model and algorithms on real data, e.g., New York city yellow cab data.

Bibliography

Agatz, N. A. H., Erera, A. L., Savelsbergh, M. W. P., & Wang, X. (2011). Dynamic ride-sharing: A simulation study in metro Atlanta. *Transportation Research Part B: Methodological*, *45* `(9), 1450–1464. https://doi.org/10.1016/j.trb.2011.05.017

Agatz, N., Bazzan, A. L. C., Kutadinata, R., Mattfeld, D. C., Sester, M., Winter, S., & Wolfson, O. (2016). Autonomous car and ride sharing: flexible road trains. *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems - GIS '16*, 1–4. https://doi.org/10.1145/2996913.2996947

Agatz, N., Erera, A., Savelsbergh, M., & Wang, X. (2012). Optimization for dynamic ride-sharing: A review. *European Journal of Operational Research*, *223*(2), 295–303. https://doi.org/10.1016/j.ejor.2012.05.028

Alarabi, L., Cao, B., Zhao, L., Mokbel, M. F., & Basalamah, A. (2016). A demonstration of SHAREK: an efficient matching framework for ride sharing systems. *Sigspatial/Gis*, 95:1-95:4. https://doi.org/10.1145/2996913.2996983

Alonso-mora, J., Wallar, A., Frazzoli, E., Rus, D., Alonso-mora, J., Samaranayake, S., Wallar, A., Frazzoli, E., & Rus, D. (2018). Correction for Alonso-Mora et al., On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, *115*(3), E555–E555. https://doi.org/10.1073/pnas.1721622115

Asghari, M., Deng, D., Shahabi, C., Demiryurek, U., & Li, Y. (2016). Price-aware real-time ride-sharing at scale: an auction-based approach. *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems - GIS '16*, 1–10. https://doi.org/10.1145/2996913.2996974

Attanasio, A., Cordeau, J. F., Ghiani, G., & Laporte, G. (2004). Parallel Tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem. *Parallel Computing*, *30*(3), 377–387. https://doi.org/10.1016/j.parco.2003.12.001

Ayala, D., Wolfson, O., Dasgupta, B., Lin, J., & Xu, B. (2018). Spatio-Temporal Matching for Urban Transportation Applications. *ACM Transactions on Spatial Algorithms and Systems*, *3*(4), 1–39. https://doi.org/10.1145/3183344

Bader, D. A., Hart, W. E., & Phillips, C. A. (2005). Parallel Algorithm Design for Branch and Bound. In H. J. G (Ed.), *Tutorials on Emerging Methodologies and Applications in Operations Research: Presented at Informs 2004, Denver, CO* (pp. 5–44). Springer New York. https://doi.org/10.1007/0-387-22827-6_5

Barreto, L., & Bauer, M. (2010). Parallel branch and bound algorithm - A comparison between serial, openMP and MPI implementations. *Journal of Physics: Conference Series*, *256*(1). https://doi.org/10.1088/1742-6596/256/1/012018

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

Bentley, J. L., & Finkel, R. a. (1974). Quad trees a data structure for retrieval on composite keys. *Acta Informatica*, *4*(1), 1–9.

Bentley, Jon Louis. (1990). *K-d trees for semidynamic point sets*. 187–197.

Berbeglia, G., Cordeau, J. F., & Laporte, G. (2010). Dynamic pickup and delivery problems. *European Journal of Operational Research*, *202*(1), 8–15. https://doi.org/10.1016/j.ejor.2009.04.024

Berbeglia, G., Cordeau, J. F., & Laporte, G. (2012). A hybrid tabu search and constraint

programming algorithm for the dynamic dial-a-ride problem. *INFORMS Journal on Computing*, *24*(3), 343–355. https://doi.org/10.1287/ijoc.1110.0454

Bernstein, D. S., Givan, R., Immerman, N., & Zilberstein, S. (2000). The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, *27*(4), 819–840. https://doi.org/10.1287/moor.27.4.819.297

Blumrosen, L., & Nisan, N. (2007). Combinatorial auctions. *Algorithmic Game Theory*, *9780521872*, 267–300. https://doi.org/10.1017/CBO9780511800481.013

Bullo, F., Frazzoli, E., Pavone, M., Savla, K., & Smith, S. L. (2011). Dynamic vehicle routing for robotic systems. *Proceedings of the IEEE*, *99*(9), 1482–1504. https://doi.org/10.1109/JPROC.2011.2158181

Cao, B., Alarabi, L., Mokbel, M. F., & Basalamah, A. (2015). SHAREK: A Scalable Dynamic Ride Sharing System. *Proceedings - IEEE International Conference on Mobile Data Management*, *1*, 4–13. https://doi.org/10.1109/MDM.2015.12

Chan, N. D., & Shaheen, S. A. (2012). Ridesharing in north America: Past, present, and future. *Transport Reviews*, *32*(1), 93–112. https://doi.org/10.1080/01441647.2011.621557

Christofides, N., & Eilon, S. (1969). An algorithm for the vehicle-dispatching problem. *Journal of the Operational Research Society*, *20*(3), 309–318.

Chung, Fan RK and Graham, F. C. (1997). *Spectral graph theory*. American Mathematical Soc.

Cici, B., Markopoulou, A., & Laoutaris, N. (2015). Designing an on-line ride-sharing system. *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems - GIS '15*, 1–4. https://doi.org/10.1145/2820783.2820850

Coltin, B., & Veloso, M. (2014). Ridesharing with passenger transfers. *IEEE International Conference on Intelligent Robots and Systems*, *Iros*, 3278–3283.

https://doi.org/10.1109/IROS.2014.6943018

Cordeau, J. F., & Laporte, G. (2007). The dial-a-ride problem: Models and algorithms. *Annals of Operations Research*, *153*(1), 29–46. https://doi.org/10.1007/s10479-007-0170-8

Coslovich, L., Pesenti, R., & Ukovich, W. (2006). A two-phase insertion technique of unexpected customers for a dynamic dial-a-ride problem. *European Journal of Operational Research*, *175*(3), 1605–1615. https://doi.org/10.1016/j.ejor.2005.02.038

CPLEX optimizer. (2019). https://www.ibm.com/analytics/cplex-optimizer

D'Orey, P. M., Fernandes, R., & Ferreira, M. (2012). Empirical evaluation of a dynamic and distributed taxi-sharing system. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, 140–146. https://doi.org/10.1109/ITSC.2012.6338703

Delle Fave, F. M., Rogers, A., Xu, Z., Sukkarieh, S., & Jennings, N. R. (2012). Deploying the max-sum algorithm for decentralised coordination and task allocation of unmanned aerial vehicles for live aerial imagery collection. *Proceedings - IEEE International Conference on Robotics and Automation*, 469–476. https://doi.org/10.1109/ICRA.2012.6225053

Delling, D., Sanders, P., Schultes, D., & Wagner, D. (2009). Engineering and augmenting route planning algorithms. *Algorithmics of Large and Complex Networks*, *5515*, 117–139. papers2://publication/uuid/7F722816-1187-4A7A-B2EF-249C3DDFCE9D

Fagnant, D. J., & Kockelman, K. M. (2018). Dynamic ride-sharing and fleet sizing for a system of shared autonomous vehicles in Austin, Texas. *Transportation*, *45*(1), 143–158. https://doi.org/10.1007/s11116-016-9729-z

Fagnant, D. J., Kockelman, K. M., & Bansal, P. (2015). Operations of shared autonomous vehicle fleet for Austin, Texas, market. *Transportation Research Record*, *2536*, 98–106. https://doi.org/10.3141/2536-12

Farinelli, A., Rogers, A., & Jennings, N. R. (2014). Agent-based decentralised coordination for sensor networks using the max-sum algorithm. *Autonomous Agents and Multi-Agent Systems*, *28*(3), 337–380. https://doi.org/10.1007/s10458-013-9225-1

Farinelli, A., Rogers, A., Petcu, A., & Jennings, N. R. (2008). Decentralised coordination of low-power embedded devices using the max-sum algorithm. *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, *2*(Aamas), 630–637.

Fioretto, F., Pontelli, E., & Yeoh, W. (2018). Distributed constraint optimization problems and applications: A survey. *Journal of Artificial Intelligence Research*, *61*, 623–698. https://doi.org/10.1613/jair.5565

Fischer, K., Müller, J. P., & Pischel, M. (1996). Cooperative transportation scheduling: an application domain for dai. *Applied Artificial Intelligence*, *10*(1), 1–34. https://doi.org/10.1080/088395196118669

Furuhata, M., Dessouky, M., Ordóñez, F., Brunet, M.-E., Wang, X., & Koenig, S. (2013). *Ridesharing: The state-of-the-art and future directions*. 1–44. https://doi.org/10.1016/j.trb.2013.08.012

Gao, J., Wang, Y., Tang, H., Yin, Z., Ni, L., & Shen, Y. (2017). An Efficient Dynamic Ridesharing Algorithm. *IEEE CIT 2017 - 17th IEEE International Conference on Computer and Information Technology*, 320–325. https://doi.org/10.1109/CIT.2017.33

Gerte, R., Konduri, K. C., & Eluru, N. (2018). Is there a limit to adoption of dynamic ridesharing systems? Evidence from analysis of uber demand data from new york city. *Transportation Research Record*, *2672*(42), 127–136. https://doi.org/10.1177/0361198118788462

Ghoseiri, K., Haghani, A., & Hamedi, M. (2011). Real-Time rideshare matching problem. *Thesis*.

Google geocoding API. (2020). https://developers.google.com/maps/documentation/geocoding/usage-and-billing#set-caps

Gottlob, G., & Greco, G. (2013). Decomposing combinatorial auctions and set packing problems. *Journal of the ACM*, *60*(4), 1–39. https://doi.org/10.1145/2505987

Gurobi optimizer. (2019). https://www.gurobi.com/

Guttman, A. (1984). R-trees: A dynamic index structure for spatial searching. *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data - SIGMOD '84*, 47. https://doi.org/10.1145/602259.602266

Helsgaun, K. (2000). Effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, *126*(1), 106–130. https://doi.org/10.1016/S0377-2217(99)00284-2

Herrera, J. F. R., Salmerón, J. M. G., Hendrix, E. M. T., Asenjo, R., & Casado, L. G. (2017). On parallel Branch and Bound frameworks for Global Optimization. *Journal of Global Optimization*, *69*(3), 547–560. https://doi.org/10.1007/s10898-017-0508-y

Hirayama, K., & Yokoo, M. (1997). Distributed partial constraint satisfaction problem. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *1330*, 222–236. https://doi.org/10.1007/bfb0017442

Ho, S. C., Szeto, W. Y., Kuo, Y. H., Leung, J. M. Y., Petering, M., & Tou, T. W. H. (2018). A survey of dial-a-ride problems: Literature review and recent developments. *Transportation Research Part B: Methodological*, *111*, 395–421. https://doi.org/10.1016/j.trb.2018.02.001

Hosni, H., Naoum-Sawaya, J., & Artail, H. (2014). The shared-taxi problem: Formulation and solution methods. *Transportation Research Part B: Methodological*, *70*, 303–318. https://doi.org/10.1016/j.trb.2014.09.011

Huang, Y., Jin, R., Bastani, F., & Wang, X. S. (2013). *Large Scale Real-time Ridesharing with Service Guarantee on Road Networks*. *7*(14), 2017–2028. https://doi.org/10.14778/2733085.2733106

Hyland, M. F., & Mahmassani, H. S. (2017). Taxonomy of shared autonomous vehicle fleet management problems to inform future transportation mobility. *Transportation Research Record*, *2653*, 26–34. https://doi.org/10.3141/2653-04

Jaw, J.-J., Odoni, A. R., Psaraftis, H. N., & Wilson, N. H. M. (1986). A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. *Transportation Research Part B: Methodological*, *20*(3), 243–257. https://doi.org/10.1016/0191-2615(86)90020-2

Jia, Y., Xu, W., & Liu, X. (2017). An optimization framework for online ride-sharing markets. *Proceedings - International Conference on Distributed Computing Systems*, 826–835. https://doi.org/10.1109/ICDCS.2017.185

Jung, J., Jayakrishnan, R., & Park, J. Y. (2016). Dynamic shared-taxi dispatch algorithm with hybrid-simulated annealing. *Computer-Aided Civil and Infrastructure Engineering*, *31*(4), 275–291. https://doi.org/10.1111/mice.12157

Kalantari, B., Hill, A. V., & Arora, S. R. (1985). An algorithm for the traveling salesman problem with pickup and delivery customers. *European Journal of Operational Research*, *22*(3), 377–386. https://doi.org/10.1016/0377-2217(85)90257-7

Kelly, K. (2007). Casual carpooling-enhanced. *Journal of Public Transportation*, *10*(4), 119–130. https://doi.org/10.5038/2375-0901.10.4.6

Kleiner, A., Nebel, B., & Ziparo, V. A. (2011). A mechanism for dynamic ride sharing based on parallel auctions. *IJCAI International Joint Conference on Artificial Intelligence*, 266–272.

https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-055

Kok, J. R., & Vlassis, N. (2006). Collaborative multiagent reinforcement learning by payoff propagation. *Journal of Machine Learning Research, 7*, 1789–1828.

Kriegel, H. P., Seeger, B., Schneider, R., & Beckmann, N. (1990). The R-tree: an efficient and robust access method for points and rectangles. *GIS for the 1990s. Proc. National Conference, Ottawa, 1990*, 448–455.

Krishnamurti, R. (2002). The multiple traveling salesman problem with time windows : Bounds for the minimum number of vehicles. *Time*, 1–16.

Lee, M. L., Hsu, W., Jensen, C. S., Cui, B., & Teo, K. L. (2003). Supporting frequent updates in R-Trees: A bottom-up approach. *Proceedings 2003 VLDB Conference*, 608–619. https://doi.org/10.1016/b978-012722442-8/50060-4

Liebling, T. M. (1987). Large scale combinatorial optimization problems: randomized exchange heuristics. *IFAC Proceedings Volumes*, *20*(9), 73–80. https://doi.org/10.1016/s1474-6670(17)55683-9

Lin, S., & Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations Research, 21*(2), 498–516. https://doi.org/10.1287/opre.21.2.498

Liu, Z., Miwa, T., Zeng, W., & Morikawa, T. (2018). An agent-based simulation model for shared autonomous taxi system. *Asian Transport Studies*, *5*(1), 1–13. https://doi.org/10.11175/eastsats.5.1

Lowalekar, M., Varakantham, P., & Jaillet, P. (2019). ZAC: A zone path construction approach for effective real-time ridesharing. *Proceedings International Conference on Automated Planning and Scheduling, ICAPS, Icaps*, 528–538.

Ma, S., Zheng, Y., & Wolfson, O. (2013). T-Share : A Large-Scale Dynamic Taxi Ridesharing.

Proceedings - International Conference on Data Engineering, 410–421. https://doi.org/10.1109/ICDE.2013.6544843

Ma, S., Zheng, Y., & Wolfson, O. (2015). Real-time city-scale taxi ridesharing. *IEEE Transactions on Knowledge and Data Engineering*, *27*(7), 1782–1795. https://doi.org/10.1109/TKDE.2014.2334313

Mallus, M., Colistra, G., Atzori, L., Murroni, M., & Pilloni, V. (2017). Dynamic carpooling in urban areas: Design and experimentation with a multi-objective route matching algorithm. *Sustainability (Switzerland)*, *9*(2). https://doi.org/10.3390/su9020254

Mes, M., van der Heijden, M., & van Harten, A. (2007). Comparison of agent-based scheduling to look-ahead heuristics for real-time transportation problems. *European Journal of Operational Research*, *181*(1), 59–75. https://doi.org/10.1016/j.ejor.2006.02.051

Mills-tettey, G. A., Stentz, A., & Dias, M. B. (2007). The Dynamic Hungarian Algorithm for the Assignment Problem with Changing Costs. *Naval Research Logistics Quarterly*, *July*, 83–87.

Modi, P. J., Shen, W. M., Tambe, M., & Yokoo, M. (2005). Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, *161*(1–2), 149–180. https://doi.org/10.1016/j.artint.2004.09.003

Molenbruch, Y., Braekers, K., & Caris, A. (2017). Typology and literature review for dial-a-ride problems. *Annals of Operations Research*, *259*(1–2), 295–325. https://doi.org/10.1007/s10479-017-2525-0

Morency, C. (2007). The ambivalence of ridesharing. *Transportation*, *34*(2), 239–253. https://doi.org/10.1007/s11116-006-9101-9

Mosek optimization solver. (2019). https://www.mosek.com/

Najmi, A., Rey, D., & Rashidi, T. H. (2017). Novel dynamic formulations for real-time ride-

sharing systems. *Transportation Research Part E: Logistics and Transportation Review*, *108*(September), 122–140. https://doi.org/10.1016/j.tre.2017.10.009

Nash, J. F., & others. (1950). Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, *36*(1), 48–49.

New York City Yellow Taxi Trip. (2016).

Ng, A. Y., Jordan, M. I., & Weiss, Y. (2002). On spectral clustering analysis and an algorithm. In *Advances in neural information processing systems* (pp. 849–856). MIT press Cambridge. https://doi.org/10.1.1.19.8100

Nourinejad, M., & Roorda, M. J. (2016). Agent based model for dynamic ridesharing. *Transportation Research Part C: Emerging Technologies*, *64*, 117–132. https://doi.org/10.1016/j.trc.2015.07.016

Ota, M., Vo, H., Silva, C., & Freire, J. (2015). A scalable approach for data-driven taxi ride-sharing simulation. *Proceedings - 2015 IEEE International Conference on Big Data, IEEE Big Data 2015*, 888–897. https://doi.org/10.1109/BigData.2015.7363837

Parragh, S. N., Doerner, K. F., & Hartl, R. F. (2008). A survey on pickup and delivery problems. *Journal Fur Betriebswirtschaft*, *58*(1), 21–51. https://doi.org/10.1007/s11301-008-0033-7

Perugini, D., Lambert, D., Sterling, L., & Pearce, A. (2003). A distributed agent approach to global transportation scheduling. *IEEE/WIC International Conference on Intelligent Agent Technology, 2003. IAT 2003.*, 18–24. https://doi.org/10.1109/IAT.2003.1241043

Petcu, A., & Faltings, B. (2005). A scalable method for multiagent constraint optimization. *IJCAI International Joint Conference on Artificial Intelligence*, 266–271.

Pillac, V., Gendreau, M., Guéret, C., & Medaglia, A. L. (2013). A review of dynamic vehicle routing problems. *European Journal of Operational Research*, *225*(1), 1–11.

https://doi.org/10.1016/j.ejor.2012.08.015

Ralphs, T. K. (2006). Parallel branch and cut. *Parallel Combinatorial Optimization*, 53–101. https://doi.org/10.1002/9780470053928.ch3

Reinelt, G. (1994). *The traveling salesman: computational solutions for TSP applications*. Springer-Verlag.

Rubinstein, Z. B., Smith, S. F., & Barbulescu, L. (2012). Incremental management of oversubscribed vehicle schedules in dynamic dial-a-ride problems. *Proceedings of the National Conference on Artificial Intelligence*, *3*, 1809–1815.

Santos, D. O., & Xavier, E. C. (2015). Taxi and ride sharing: A dynamic dial-a-ride problem with money as an incentive. *Expert Systems with Applications*, *42*(19), 6728–6737. https://doi.org/10.1016/j.eswa.2015.04.060

Saraswathi, A. T., Kalaashri, Y. R. A., & Padmavathi, S. (2015). Dynamic resource allocation scheme in cloud computing. *Procedia Computer Science*, *47*(C), 30–36. https://doi.org/10.1016/j.procs.2015.03.180

Schreieck, M., Safetli, H., Siddiqui, S. A., Pflügler, C., Wiesche, M., & Krcmar, H. (2016). A Matching Algorithm for Dynamic Ridesharing. *Transportation Research Procedia*, *19*(June), 272–285. https://doi.org/10.1016/j.trpro.2016.12.087

Schwarting, W., Alonso-Mora, J., & Rus, D. (2018). Planning and Decision-Making for Autonomous Vehicles. *Annual Review of Control, Robotics, and Autonomous Systems*, *1*(1), annurev-control-060117-105157. https://doi.org/10.1146/annurev-control-060117-105157

Sellis, T. K., Roussopoulos, N., & Faloutsos, C. (1987). The R+-Tree: A Dynamic Index for Multi-Dimensional Objects. *Proceedings of the 13th International Conference on Very Large Data Bases*, 507–518.

Shemshadi, A., Sheng, Q. Z., & Zhang, W. E. (2014). A decremental search approach for large scale dynamic ridesharing. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *8786*, 202–217. https://doi.org/10.1007/978-3-319-11749-2_16

Shen, B., Huang, Y., & Zhao, Y. (2016). Dynamic ridesharing. *SIGSPATIAL Special*, *7*(3), 3–10. https://doi.org/10.1145/2876480.2876483

Shi, J., & Malik, J. (2000). Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *22*(8), 888–905. https://doi.org/10.1109/34.868688

Simonetto, A., Monteil, J., & Gambella, C. (2019). Real-time city-scale ridesharing via linear assignment problems. *Transportation Research Part C: Emerging Technologies*, *101*, 208–232. https://doi.org/10.1016/j.trc.2019.01.019

Smith, R. G. (1980). The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, *C–29*(12), 1104–1113. https://doi.org/10.1109/TC.1980.1675516

Song, C. H., Lee, K., & Lee, W. D. (2003). Extended simulated annealing for augmented TSP and multisalsemen TSP. *Proceedings of the International Joint Conference on Neural Networks*, *3*, 2340–2343.

Stranders, R., Farinelli, A., Rogers, A., & Jennings, N. R. (2009). Decentralised coordination of mobile sensors using the max-sum algorithm. *IJCAI International Joint Conference on Artificial Intelligence*, 299–304.

Teodorovic, D., & Radivojevic, G. (2000). *A fuzzy logic approach to dynamic Dial-A-Ride problem*. *116*, 23–33.

Tsubakitani, S., & Evans, J. R. (1998). Optimizing tabu list size for the traveling salesman problem. *Computers and Operations Research*, *25*(2), 91–97. https://doi.org/10.1016/S0305-0548(97)00030-0

Velagapudi, P., Varakantham, P., Scerri, P., & Sycara, K. (2011). Distributed model shaping for scaling to decentralized POMDPs with hundreds of agents categories and subject descriptors. *International Conference on Autonoumous Agents and MultiAgent Systems (AAMAS 11)*, 955–962.

Von Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and Computing*, *17*(4), 395–416. https://doi.org/10.1007/s11222-007-9033-z

Waldock, A., Nicholson, D., & Rogers, A. (2008). Cooperative control using the max-sum algorithm. *Computer*, 65–70. http://eprints.soton.ac.uk/265456/

Wang, J., Wang, T., Wang, X., & Meng, X. (2009). Multi-robot decision making based on coordination graphs. *2009 IEEE International Conference on Mechatronics and Automation, ICMA 2009*, 2393–2398. https://doi.org/10.1109/ICMA.2009.5246091

Xiao, Z., Song, W., & Chen, Q. (2013). Dynamic resource allocation using virtual machines for cloud computing environment. *IEEE Transactions on Parallel and Distributed Systems*, *24*(6), 1107–1117. https://doi.org/10.1109/TPDS.2012.283

Yedidsion, H., Zivan, R., & Farinelli, A. (2014). Explorative max-sum for teams of mobile sensing agents. *13th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2014*, *1*, 549–556.

Yeoh, W., & Yokoo, M. (2012). Distributed problem solving. *AI Magazine*, *33*(3), 53–65. https://doi.org/10.1609/aimag.v33i3.2429

Yuni Xia, & Prabhakar, S. (2003). Q+Rtree: efficient indexing for moving object databases. *Eighth*

International Conference on Database Systems for Advanced Applications, 2003. (DASFAA 2003). Proceedings., 175–182. https://doi.org/10.1109/DASFAA.2003.1192381

Zhang, W., Wang, G., Xing, Z., & Wittenburg, L. (2005). Distributed stochastic search and distributed breakout: Properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence*, *161*(1–2), 55–87. https://doi.org/10.1016/j.artint.2004.10.004

Zhang, Z., & Zhao, D. (2014). Clique-based cooperative multiagent reinforcement learning using factor graphs. *IEEE/CAA Journal of Automatica Sinica*, *1*(3), 248–256. https://doi.org/10.1109/JAS.2014.7004682

Zheng, Y., Capra, L., Wolfson, O., & Yang, H. (2014). Urban Computing: concepts, methodologies, and applications. *ACM Transactions on Intelligent Systems and Technology*, *5*(3), 1–55. https://doi.org/10.1145/2629592

Zivan, R., & Peled, H. (2012). Max/min-sum distributed constraint optimization through value propagation on an alternating DAG. *11th International Conference on Autonomous Agents and Multiagent Systems 2012, AAMAS 2012: Innovative Applications Track*, *1*(June), 280–287.