Technical University of Crete
School of Electrical and Computer Engineering

Preference Aggregation in the Ridesharing Domain

Diploma Thesis
by

Antigoni Asproudi

Thesis committee

Supervisor : Georgios Chalkiadakis, Professor
Michail Lagoudakis, Professor
Panagiotis Partsinevelos, Professor (School of MRE)

Chania, December 2023

Πολυτεχνειο Κρητης
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Συνάθροιση Προτιμήσεων για το Πρόβλημα Κοινωνικού Διαμοιρασμού Διαδρομών

Διπλωματική Εργασία

της

Αντιγόνης Ασπρούδη

Επιτροπή

Επιβλέπων : Γεώργιος Χαλκιαδάκης, Καθηγητής

Μιχαήλ Λαγουδάκης, Καθηγητής

Παναγιώτης Παρτσινέβελος, Καθηγητής (Σχολή ΜΗΧΟΠ)

Χανιά, Δεκέμβριος 2023

Acknowledgement

I would like to thank everyone who contributed to making this thesis possible.

First and foremost, I would like to thank my supervisor, Professor Georgios Chalkiadakis, for his guidance throughout my thesis. I would also like to thank Errikos Streviniotis for offering his help and advice when needed and Manolis Pagkalos, whose work I utilized in my implementation.

Last, but not least, I would like to express my gratitude to my family and friends, who supported me all these years.

Abstract

Traditional ridesharing approaches involve the matching of drivers and passengers based on their itineraries and schedules. The objective is to share the trip cost, while also minimizing time delays and the extra distance covered by the driver, when accommodating the passengers. It serves as a flexible and economical alternative to traditional means of transportation, offering environmental and congestion-friendly benefits, as with more individuals sharing rides, the number of vehicles on the streets is reduced.

In this thesis, we approach the ridesharing problem considering spatial criteria, while also incorporating agents' preferences regarding the characteristics of their co-riders. This includes matching between passengers through preference aggregation in order to determine which ones should share each vehicle. Preference satisfaction is essential in our work, enhancing the overall ride experience and contributing to riders' sense of safety by taking into account their preferences regarding potential co-riders.

Rooted in game-theoretic concepts, our approach produces core-stable matches and kernel-stable payment allocations of the trip cost. Hypergraphs are employed to identify passengers located in the same area as the driver. Through experiments involving variations in the shape of the area from which the driver can accommodate passengers, we assess its impact on the extra distance covered. Employing the Gale-Shapley algorithm, we match drivers with passengers, ensuring outcomes are in the core. To plan the coalition's route, we apply a Branch and Bound algorithm to find a good sequence of stops the driver has to make to minimize the cost of the trip. Dijkstra's algorithm connects these stops to form the final path. Subsequently, the trip cost is divided fairly among commuters in each vehicle using the "kernel" cooperative game-theoretic solution concept.

The performance of our work is tested in simulations run in the city of Chania, for a range of 800 to 2500 agents with 35% of those being drivers in each case. The results indicate that our implementation outperforms previous work in terms of preference satisfaction. Moreover, it efficiently matches agents, with an average of three passengers per vehicle, assuming a maximum capacity of four passengers, contributing to the minimization of the volume of vehicles in the streets. Ultimately, the rise in the extra distance

covered by the driver is maintained at an acceptable level and the driver's cost for the trip is effectively reduced.

Περίληψη

Οι παραδοσιακές προσεγγίσεις στο ridesharing (ή αλλιώς διαμερισμό διαδρομών) περιλαμβάνουν την ομαδοποίηση οδηγών και επιβατών βάσει των δρομολογίων και των χρονοδιαγραμμάτων τους. Ο στόχος είναι να μοιραστεί το κόστος της διαδρομής, ελαχιστοποιώντας παράλληλα τις καθυστερήσεις και την επιπλέον αύξηση της απόστασης που καλύπτει ο οδηγός κατά την εξυπηρέτηση των επιβατών. Αποτελεί μία ευέλικτη και οικονομική εναλλακτική λύση στα παραδοσιακά μέσα μεταφοράς, προσφέροντας οφέλη φιλικά για το περιβάλλον και την αντιμετώπιση της κυκλοφοριακής συμφόρησης, καθώς με τη χρήση του ridesharing από περισσότερα άτομα, μειώνεται ο αριθμός των οχημάτων στους δρόμους.

Σε αυτή τη διπλωματική εργασία, προσεγγίζουμε το πρόβλημα του ridesharing λαμβάνοντας υπόψη κριτήρια που έχουν να κάνουν τόσο με την τοποθεσία, όσο και με τις προτιμήσεις των πρακτόρων σχετικά με τα χαρακτηριστικά των συνεπιβατών τους. Αυτό συμπεριλαμβάνει και την ομαδοποίηση μεταξύ επιβατών, εφαρμόζοντας συνάθροιση προτιμήσεων μεταξύ τους, προκειμένου να επιλεχθούν αυτοί που θα μοιραστούν το ίδιο όχημα. Η ικανοποίηση των προτιμήσεων είναι ένα σημαντικό κομμάτι της δουλειάς μας, καθώς συμβάλλει στο να έχουν όλοι μια καλύτερη εμπειρία ταξιδιού, αλλά και να αισθάνονται οι επιβάτες ασφαλείς, λαμβάνοντας υπόψη τις προτιμήσεις τους σχετικά με τους συνεπιβάτες τους.

Βασισμένη στην Θεωρία Παιγνίων, η προσέγγισή μας παράγει core-stable ζευγάρια οδηγών-επιβατών και kernel-stable κατανομή κόστους του ταξιδιού. Με τη χρήση υπεργράφων, γίνεται εντοπισμός επιβατών που μετακινούνται στην ίδια περιοχή με τον οδηγό, ενώ διεξάγοντας πειράματα που περιλαμβάνουν παραλλαγές στο σχήμα της περιοχής από την οποία ο οδηγός μπορεί να εξυπηρετήσει επιβάτες, αξιολογούμε την επίδρασή της στην επιπλέον απόσταση που αυτός θα πρέπει να καλύψει. Με τη χρήση του αλγορίθμου Gale-Shapley, "ταιριάζουμε" οδηγούς με επιβάτες, εξασφαλίζοντας ότι τα αποτελέσματα βρίσκονται στον πυρήνα (core). Για τον σχεδιασμό της τελικής διαδρομής, εφαρμόζουμε έναν αλγόριθμο Branch and Bound για τον εντοπισμό μιας καλής ακολουθίας στάσεων που πρέπει να κάνει ο οδηγός για να ελαχιστοποιήσει το κόστος του ταξιδιού. Ο αλγόριθμος του Dijkstra συνδέει αυτές τις στάσεις για να δημιουργήσει την τελική διαδρομή. Στη συνέχεια, το κόστος του ταξιδιού μοιράζεται δίκαια μεταξύ των επιβατών κάθε οχήματος, χρησιμοποιώντας την παιγνιοθεω-

ρητική έννοια χαρακτηρισμού σταθερότητας συνασπισμών "kernel".

Η απόδοση της δουλειάς μας ελέγχεται με τη διεξαγωγή προσομοιώσεων στην πόλη των Χανίων, για ένα εύρος από 800 έως 2500 πράκτορες, όπου το 35% αυτών είναι οδηγοί. Τα αποτελέσματα υποδηλώνουν ότι η υλοποίησή μας ξεπερνά σε επιδόσεις προηγούμενη δουλειά πάνω στο αντικείμενο, όσον αφορά την ικανοποίηση των προτιμήσεων. Επιπλέον, συνδυάζει αποτελεσματικά τους πράκτορες, τοποθετώντας περίπου τρεις ανά όχημα, υποθέτοντας μέγιστη χωρητικότητα τεσσάρων επιβατών, συμβάλλοντας με αυτό τον τρόπο στη μείωση του όγκου των οχημάτων στους δρόμους. Τέλος, η αύξηση της επιπλέον απόστασης που καλύπτει ο οδηγός προκειμένου να εξυπηρετήσει τους επιβάτες διατηρείται σε αποδεχτά επίπεδα, ενώ το κόστος διαδρομής του οδηγού μειώνεται.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

Ridesharing [2, 3] is a transportation service where people with similar itineraries share a vehicle. Typically, it entails a driver who offers rides to one or more individuals. We categorize the ridesharing domain into two major categories. In the first category, drivers are ordinary individuals seeking cost-sharing while going about their daily activities. In such scenarios, passengers simply share a portion of the travel expenses in return for the ride. In the second category, ridesharing takes the form of mobility-as-a-service[1]. In these instances, drivers are affiliated with professional car service companies and provide *ride-hailing* services, typically facilitated through smartphone applications. Passengers compensate with a fee to reach their destination.

Ridesharing provides a convenient alternative for commuting in urban areas. It offers greater flexibility than public transportation, with less crowding, and is a more affordable option compared to traditional taxi services. Increased participation in ridesharing has the potential to curtail the volume of vehicles in urban centers, thereby mitigating traffic congestion. Furthermore, this practice contributes to a reduced environmental footprint, given the resulting reduced number of automobiles on the road, leading to a corresponding reduction in pollution. Additionally, it can aid in alleviating the parking problem in large cities, as a substantial amount of time and fuel is wasted while searching for a parking spot, and money is also spent on renting parking spaces. All of this could be diminished

[1] See e.g. Uber, Lyft

with the widespread use of ridesharing.

Considering the aforementioned factors, along with the continuous growth of the population in the cities, ridesharing is increasingly relevant. The rise of technology, the extensive use of smartphones and the ease of access mobile apps provide have played a key role in the growth and innovation of ridesharing. Companies like Uber and Lyft, with their diverse and evolving services, have disrupted traditional transportation providers. With a burgeoning market valued at USD 69.3 billion in 2022 and projected to reach USD 205.83 billion by 2030, growing at CAGR 13.5%, the ridesharing industry continues to attract significant investment and competition among the primary companies looking to capitalize on its expanding potential.

In this thesis, we study ridesharing scenarios with regular drivers who are not associated with transportation service companies. We focus on matching and cost allocation among the commuters of a vehicle, as we consider these to be some of the main challenges in ridesharing. To this end, our goal is to maintain a balance between the extra distance covered by the driver to accommodate the passengers and the preference satisfaction of all individuals. In other words, our ridesharing approach offers automated matching of drivers and passengers based on both spatial and preferential criteria, and also preference-centered matching among passengers in the same vehicle. While one-passenger matches are easier to produce and result in less extra distance, multi-passenger matches offer other benefits. Besides splitting the travel expenses among more people, commuting together as a group can increase the feeling of safety.

1.1 Thesis Contributions

As in any ridesharing approach, our aim is to effectively match drivers with passengers, while avoiding long detours and delays. However, our work places equal emphasis on the quality of the ride, specifically its overall pleasantness for all participants. In this thesis, we expand the preference-aware ridesharing model introduced in [1], wherein agents' preferences regarding the characteristics of their co-riders are taken into consideration. More specifically, the matching of agents is conducted in two stages, with preferences considered in each. In the first stage, drivers are matched with passengers based on both spatial and preferential criteria, while in the second stage, passengers are matched with one another through preference aggregation. This ensures that everyone experiences an enjoyable ride. For instance, an older person may find it challenging to socialize with a group of teenagers, and thus, they might prefer individuals closer to their age. Similarly, university students may prefer to ride with and meet people from their school,

while women might seek to ride with other women to enhance their feeling of safety. All these attributes are taken into consideration in the coalition formation, and the results demonstrate that our approach outperforms previous work in terms of preference satisfaction. Furthermore, rooted in game theoretic concepts, our approach ensures stability. Specifically, the matches produced are core-stable, indicating that agents have no incentive to deviate from their coalitions. Additionally, the trip cost allocation is kernel-stable, ensuring fair payments. The assurance of fairness in payments holds crucial importance in ridesharing settings, as its absence can deter people from participating. The efficiency of our work is assessed through a series of simulations of ridesharing scenarios. The experimental findings reveal that our approach to the ridesharing problem offers successful matching of agents in terms of preference satisfaction, cost reduction and average number of passengers per car—an essential factor in reducing vehicles on the streets.

1.2 Thesis structure

The rest of this thesis is structured as follows: The second chapter offers a comprehensive theoretical background crucial for a complete understanding of our work. It includes a literature review summarizing previous research in the ridesharing domain, an examination of real-world ridesharing related platforms, and an overview of the work integrated into our study. The third chapter analyzes our approach to the ridesharing problem. This contains the definition of feasible coalitions through the establishment of constraints and the use of hypergraphs, the coalition formation using the Gale-Shapley algorithm for stable matching, the route extraction for each coalition, and finally, the payment allocation of the trip cost, using the kernel solution concept. In the fourth chapter, the performance of our work is evaluated through a series of experiments in the city of Chania. This entails the examination of different agent group sizes for both versions of the Gale-Shapley algorithm. The results of each version are compared to each other, in addition to those of previous work. Lastly, the fifth chapter presents concluding remarks and outlines potential future research directions.

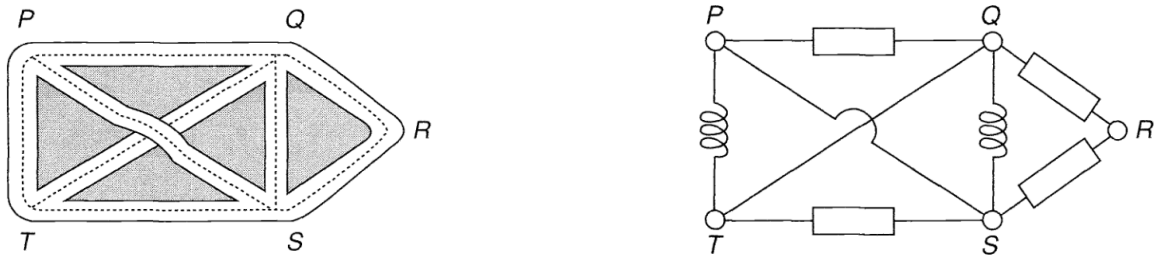
2

Theoretical Background & Related Work

In this chapter, we present the necessary theoretical background for our work. We begin with a basic introduction to graph theory and trees and we continue with Dijkstra's algorithm for pathfinding. Subsequently, we define hypergraphs, coalition formation, characteristic function games and related solution concepts. We present the Gale-Shapley algorithm for stable matching, and finally, we discuss related work in the ridesharing domain.

2.1 Graph Theory

Graph Theory can be defined as the theoretical study of graphs. Graphs are mathematical structures representing discrete objects and the connections between them. They have applications in many scientific fields, including discrete mathematics, chemistry and computer science, as they can help represent and solve complex problems. Roads and electrical circuits are some examples in which graphs can be used to represent them, as we can see in Figure 2.1 from [4].

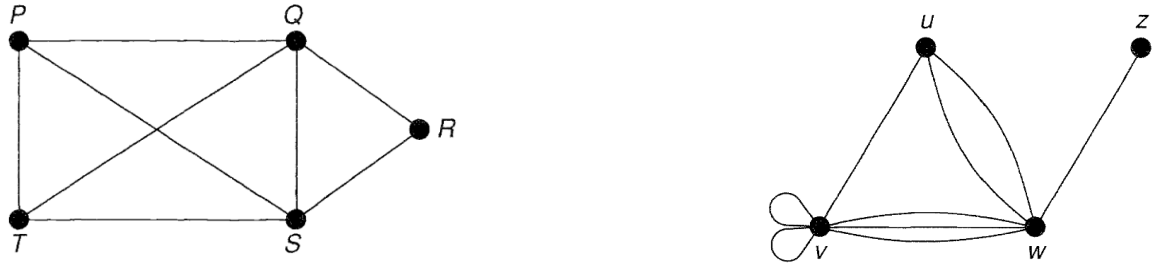


(a) Road

(b) Electrical Circuit

Figure 2.1: Applications of Graphs

*A **graph** $G = (V, E)$ consists of a set of objects $V = \{v_1 v_2, ...\}$ called vertices, and another set $E = \{e_1, e_2, ...\}$, whose elements are called edges, such that each edge e_k is identified with an unordered pair (v_i, v_j) of vertices. The vertices $v_i v_j$ associated with edge e_k are called the end vertices of e_k.* [5]



(a) Simple Graph

(b) General Graph

Figure 2.2: Simple and general Graphs

Edges connect two *adjacent* vertices. Likewise, a vertex connects adjacent edges. Also the number of edges connecting a pair of vertices, can help categorise the graph. More formally defined:

*We say that two vertices v and w of a graph G are **adjacent** if there is an edge vw joining them, and the vertices v and w are then incident with such an edge. Similarly, two distinct edges e and f are adjacent if they have a vertex in common.* [4]

*If there is at most one edge connecting a particular pair of vertices and there are no loops connecting a vertex to itself, then we say that the graph is **simple**. Otherwise, the graph is general.* [4]

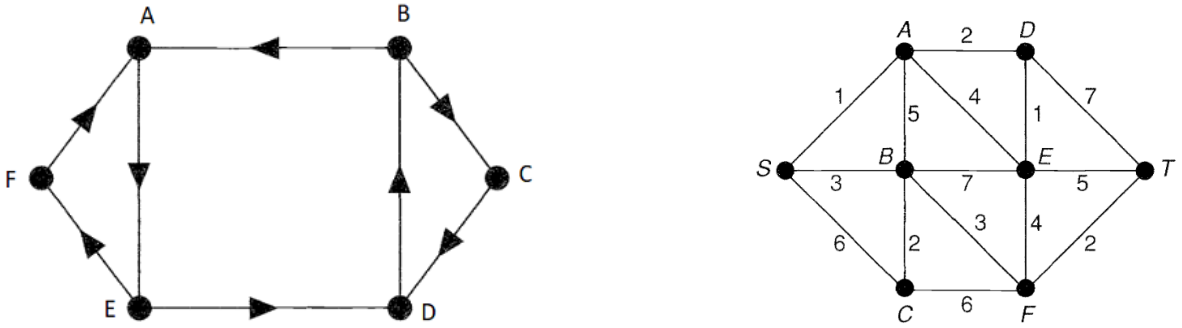
In Figure 2.2 from [4], for the simple graph, the set of vertices comprises P, Q, R, S and T, while for the general graph, U, V, W and Z. The set of edges entails the 8 lines

that connect the vertices and the 9 lines respectively. The vertices P and Q are adjacent and so are the edges connecting P to Q and Q to R. In these examples, the edges of the graph simply state the connection between the vertices. But more information between them might be needed. For example, in Figure 2.1a of a road, we might wish to know if there is a one-way street. If we assume that the course from P to S is indeed one-way, that would mean that you can move from P to S, but not from S to P. To depict this, we would need to show direction. Here, the *directed graphs* come in handy.

*A **directed graph** (or a digraph for short) G consists of a set of vertices* $V = \{v_1, v_2, ...\}$, *a set of edges* $E = \{e_1, e_2, ...\}$ *, and a mapping* Ψ *that maps every edge onto some ordered pair of vertices* (v_i, v_j). *As in the case of undirected graphs, a vertex is represented by a point and an edge by a line segment between* v_i *and* v_j *with an arrow directed from* v_i *to* v_j. *The vertex* v_i, *which edge* e_k *is incident out of, is called the initial vertex of* e_k. *The vertex* v_j, *which* e_k *is incident into, is called the terminal vertex of* e_k. [5]

A useful category of graphs whose edges also give us some kind of information is *weighted graphs.* Each edge is associated with a weight that can represent many things. For example, in the road graph of Figure 2.1a, the weights could be the distance between the vertices, or the cost of fuel to travel the particular distance.

*A **weighted graph** is a graph in which every edge* e_i *is associated with a number, a weight* $w(e_i)$. [5]



(a) Directed Graph

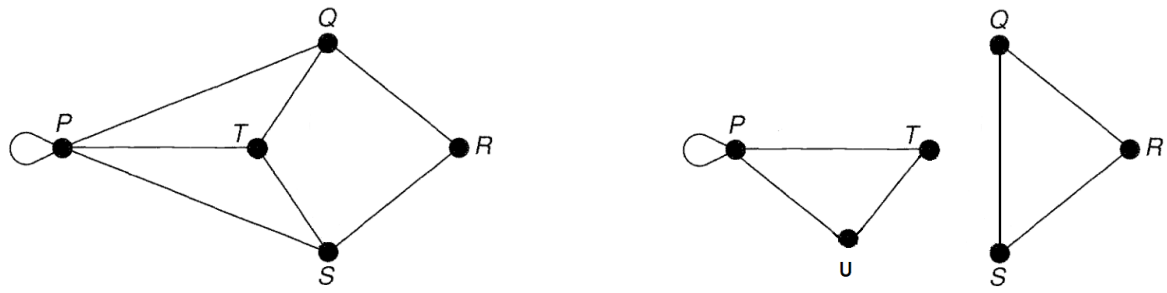
(b) Weighted Graph

Figure 2.3: Graphs with more informative edges

In Figure 2.3a from [4], we see an example of a directed graph with 6 vertices and 8 edges, where the direction of the edge that connects A and B is from vertex B to A. Similarly, in Figure 2.3b from [4] of a weighted graph, a weight has been assigned to each edge with the weight of the edge that connects D and T being 7. In these examples, we saw how the edges of a graph can be used to represent different kinds of

information. However, graphs have numerous other properties one can discover while studying them. A fundamental way of studying graphs is studying substructures of them and their properties. An essential case is the *path*. Before we define the path, we must first define the *walk*.

Given a graph G, a ***walk*** *in G is a finite sequence of edges of the form* $v_0v_1, v_1v_2, ..., v_{m-1}v_m$ *also denoted by* $v_0 \to v_1 \to v_2 \to ... \to vm$*, in which any two consecutive edges are adjacent or identical. Such a walk determines a sequence of vertices* $v_0, v_1..., v_m$*. We call* v_0 *the initial vertex and* v_m *the final vertex of the walk, and speak of a walk from* v_0 *to* v_m*. The number of edges in a walk is called its length.* [4]



(a) Connected Graph (b) Disconnected Graph

Figure 2.4: Connected and Disconnected Graphs

In Figure 2.4a from [4], $Q \to T \to P \to P \to S \to T \to Q \to R$ is an example of a walk of length 7 from Q to R. Because the concept of the walk is too general, by placing some constrains we have:

A walk in which all the edges are distinct is a ***trail****. If, in addition, the vertices* $v_0v_1, v_1v_2, ...,$ $v_{m-1}v_m$ *are distinct (except, possibly,* $v_0 = v_m$ *), then the trail is a* ***path****. A path or trail is* ***closed*** *if* $v_0 = v_m$*, and a closed path containing at least one edge is a* ***cycle****. Note that any loop or pair of multiple edges is a cycle* [4]

To make things clear, in Figure 2.4a $Q \to P \to P \to T \to S$ is a trail, $S \to P \to T \to Q \to R$ is a path and $Q \to T \to S \to R \to Q$ is a cycle. Expanding on the topic of paths, we can use them to define new types of graphs.

A graph G is said to be connected if there is at least one path between every pair of vertices in G. Otherwise, G is disconnected. [5]

A ***tree*** *is a simple, connected graph without any cycles.*[5]

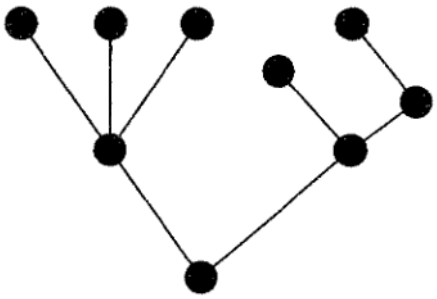


Figure 2.5: Tree

Figure 2.5 from [4] depicts a tree. The concept of trees is one of the most essential in graph theory. Trees have many useful applications. They can be used in decision-making, search and even for predicting outcomes in Game Theory. [6]

2.2 Dijkstra's Algorithm

In the previous section, we defined paths and briefly mentioned how graph structures such as trees can be used for searching purposes. In this section, we will expand on the topic of *pathfinding*. Searching for a path between a starting and a finishing point is an important topic in computer science. Pathfinding algorithms explore the graph by starting from a vertex and exploring adjacent nodes until a destination node is reached. The goal is to reach the destination through the optimal path. This path can be the shortest, the fastest, the cheapest, etc, depending on the objective and the kind of graph. For example, if the graph in question is weighted and the weights represent the distance between two nodes, the optimal path would be the one with the least accumulated weight when its edges are connected. Otherwise, without weights, the optimal path would be the shortest node-wise. Because of this, these algorithms are frequently applied in problems containing maps and mazes, where they can be used to explore all the possible options before finding the best route. There are many such algorithms. BFS, DFS [7], A* [8] and Dijkstra's algorithm [9] are among the most popular ones. In this thesis, we will be using Dijkstra's, an algorithm widely used in computer science due to its reliability.

Dijkstra's algorithm, created by Edsger W. Dijkstra, is used for finding the shortest path in a weighted graph with nonnegative edges. In the original algorithm, the shortest path is calculated between two nodes, however, one can find the shortest path between a starting node and all others. The way the algorithm works is by exploring all the possible alternative routes that can be followed, while also maintaining a set of visited vertices whose final shortest-path from the source has already been determined. By source, we

refer to the starting vertex. The algorithm repeatedly selects the vertex, from the unvisited ones, with the minimum shortest-path estimate, adds it to the visited set, and relaxes all edges leaving it [7], i.e., the distance to all adjacent nodes is calculated and if it is found to be less than the current minimum distance, it is updated along with the "parent" node. This way, after the algorithm finishes, by backward induction we can reach the source node from every node, while also getting the shortest path. Furthermore, keeping track of the visited nodes avoids evaluating the same node twice, making the overall execution more efficient.

In our approach, we apply Dijkstra's algorithm in order to find the shortest path between two points and thus form the path the driver must follow. The path is calculated on city graphs, where roads are represented by edges and intersections by vertices. The weight of every edge depicts the distance in kilometers between two vertices, which can be calculated by using the longitude and latitude of each vertex. The path is calculated initially for the driver alone. However, if at the end of the coalition formation of Section 3.2 there are passengers who matched with the driver, the path will be calculated again to include their stops. In every case, the source of the path is the driver's start and the end is their destination. The pseudocode for Dijkstra's algorithm is presented in Algorithm 1.

Algorithm 1: Dijkstra(source, destination)

```
1  set unvisited;
2  for v in vertices do
3      v.distance = ∞;
4      v.parent = NULL;
5      unvisited.insert(v);
6  source.distance=0;
7  while !unvisited.empty do
8      v = getVertexMinDistance(unvisited);
9      unvisited.remove(v);
10     if v==destination then
11         return v.path;
12     for u in neighbors of v do
13         d = v.distance + weight(v, u);
14         if d < v.distance then
15             v.distance = d;
16             v.parent = u;
```

2.3 Hypergraphs

So far we've seen how we can use graphs to represent cities and Dijkstra's algorithm to find the shortest path. However, graphs have their limitations, especially when the relationships we want to represent are not pairwise. In such cases, we can utilize *hypergraphs*. They are essentially a generalization of graphs where edges can connect any number of vertices. In other words, a simple graph is a simple hypergraph each of whose edges has cardinality 2 [10], i.e., every edge connects only 2 vertices. A formal definition is given below.

*A **hypergraph** $G = (V, H)$ | H is a set of h hyperedges, i.e., edges that can connect any number of vertices. Formally, $h \in \mathscr{P}(V)$ where $\mathscr{P}(V)$ is the set of all sets of V (a.k.a power-set of V).* [11]

A hypergraph may be drawn as a set of points representing the vertices. The edge E_j is represented by a continuous curve joining the two elements if $|E_j| = 2$, where $|E_j|$ the cardinality of the edge E_j. Alternatively, the edge is represented by a loop if $|E_j| = 1$, and by a simple closed curve enclosing the elements if $|E_j| \geq 3$ [10]. Figure 2.6 depicts an example of a hypergraph G from [10], which contains eight vertices $X = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8\}$ and six hyperedges $E = \{E_1, E_2, E_3, E_4, E_5, E_6\}$. As we can see, edges E_2 and E_5 which connect 2 vertices each, look like the edges of a simple graph. Likewise, edge E_6 which connects vertex x_7 to itself, is represented by a loop and finally, edges E_1, E_3 and E_4 connect more than 2 vertices each and thus they are represented with a closed curve.

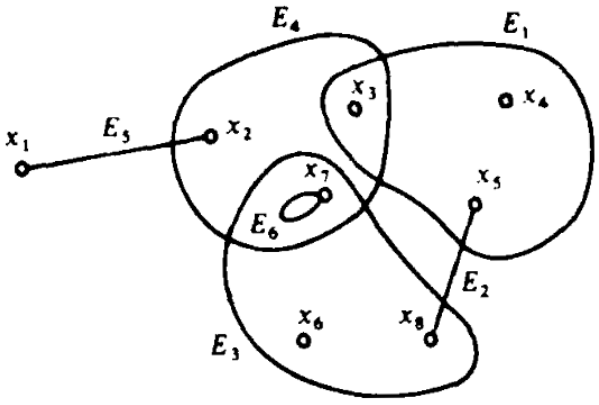
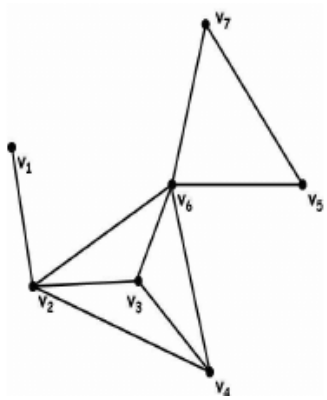


Figure 2.6: Representation of a hypergraph G

Hypergraphs are typically more informative than graphs, making representation of complex relational objects possible. While graphs can show a connection between two

objects (vertices) through edges, hypergraphs can differentiate and cluster objects based on common features. To elucidate the above, we refer to an example from Zhou et al. in [12]. In figure 2.7 we see a hypergraph, along with a simple graph. The set of vertices $V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$ represent articles, while the set of edges $E = \{e_1, e_2, e_3\}$ represent authors. From the incidence matrix we can see the entries $(v_2, e_1) = 1, (v_2, e_2) = 0, (v_2, e_3) = 1$ which means that article v_2 was written by authors e_1 and e_3. Looking at the hypergraph, this relationship between the vertices is clear, as v_2 is enclosed by both hyperedges e_1 and e_3. Furthermore, we can clearly see which articles each author has written. On the other hand, by observing the graph, we cannot tell if an author has written more than two articles, as each edge can connect only two vertices. We utilize the above properties of hyperedges in our approach. If the vertices represent passengers in a city and the hyperedges cars, we can easily show their placement in them and thus take the first step into the coalition formation. Like in the aforementioned example of articles-authors, hypergraphs give us the option to place a passenger in multiple cars. This will be proven very useful while trying to form coalitions. The constraints by which we examine which agents can be placed in each car, as well as the finalization of the coalitions, are described thoroughly in sections 3.1 and 3.2 respectively.

	e_1	e_2	e_3
v_1	1	0	0
v_2	1	0	1
v_3	0	0	1
v_4	0	0	1
v_5	0	1	0
v_6	0	1	1
v_7	0	1	0
v_2	1	0	1



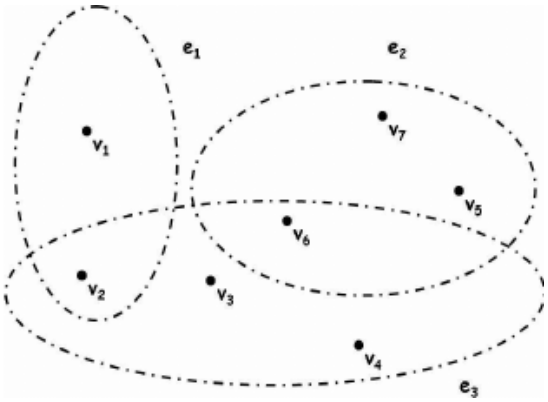


Figure 2.7: Hypergraph vs. simple graph

2.4 Coalition Formation

In Multiagent Systems [13], *coalition formation* is an essential part of cooperative game theory [14, 15]. Typically agents form coalitions(groups) and take joint actions in order to achieve their goals. They do so when joining a coalition is more fruitful than acting separately, either for them or for the overall welfare. Based on this, we distinguish two broad classes of coalition formation. According to Chalkiadakis et al. [14] we have "*coali-*

tion structure generation activities, undertaken when agents are not selfish but willingly agree to implement the agenda of a single system designer; and *coalition formation activities by selfish rational agents*, where agents choose to participate in coalitions in order to maximize their own utility". As *coalition structure* (CS), we refer to a partition of agents into disjoint coalitions. CS includes the set of all coalitions. More formally:

A ***coalition structure*** *over N is a collection of non-empty subsets* $CS = \{C^1, ..., C^k\}$ *such that*

- $U_{j=1}^{k} C^j = N$, and
- $C^i \cap C^j = \emptyset$ for any $i, j \in \{1, ..., k\}$ such that $i \neq j$

where N = {1, ..., k}, the non-empty set of all agents [14]. The first condition states that all agents must belong to a coalition. A coalition can include only one agent, in which case, it is called a *singleton*. Additionally, a coalition that includes all the agents is called *grand coalition*. The second condition states that no agent can belong to more than one coalition. The generation of CS is the first of the three parts that coalition formation is comprised of [16]. The second part is solving each coalition's *optimization problem*. This means using each agent's resources to jointly solve the problem at hand. By doing so, we aim to maximize the coalition's profit and achieve a cost-profit balance. This will prevent agents from leaving the coalition. Lastly, the third part is the *partition of the coalition value* among its members. The partition must be fair and proportional to each agent's contribution. All parts are mutually dependent as the participation of an agent in a coalition will rely on the value they will be apportioned.

The importance of coalition formation can be seen in everyday life. Complex problems or even common challenges can be solved easily through collaboration. Take as an example the Ridesharing problem that we study in our work. It is a typical case where forming a coalition benefits all parties involved. We distinguish two types of agents, drivers and passengers. Passengers wish to join a coalition, or get in a car, to get to their destination. On the other hand, drivers' motive in joining lies in sharing the travel expenses with the passengers. Of course, not all coalitions are profitable for the participants. As we mentioned before, the key factor in joining a coalition is finding the balance between cost and profit, so that agents have motive to participate. In the next chapter, we present our approach in trying to achieve this balance.

2.5 Characteristic Function Games

The model of *characteristic function games* (CFG) is the most basic and widely studied model of cooperative games. They consist of a non-empty set of $N = \{1, ..., n\}$ players, a coalition C which is a subset of the set of players N, and a function v which assigns a value to every coalition. This model does not specify how the coalitional value $v(C)$ should be distributed among the members, thus it concerns the coalition as a whole. As a matter of fact, coalitional value distribution is a research topic of its own in cooperative game theory. In Section 2.6 we present some solution concepts that handle it in a fair and stable way.

*A **characteristic function game** G is given by a pair (N, v), where $N = \{1, ..., n\}$ is a finite, non-empty set of agents and $v : 2^N \to \mathbb{R}$ is a **characteristic function**, which maps each coalition $C \subseteq N$ to a real number $v(C)$. The number $v(C)$ is usually referred to as the **value** of the coalition C.* [14]

Once the characteristic function has been specified, we can try to predict the *outcome* of the game. In cooperative games, the outcome of a characteristic function game consists of two parts; a coalition structure CS and a *payoff vector* which divides the coalitional value among the members of a coalition. More formally:

*An **outcome** of G is a pair (CS, x), where CS is a coalition structure over G and x is a payoff vector for CS. A vector $x = (x_1, ..., x_n) \in \mathbb{R}^n$ is a **payoff vector** for a coalition structure $CS = \{C^1, ..., C^k\}$ over $N = \{1, \ldots, n\}$ if*

- $x_i \geq 0$ for all $i \in N$, and
- $\sum_{i \in C^j} x_i \leq v(C^j)$ for any $j \in \{1, ..., k\}$ [14]

The above definition states that the payoff received by players is non-negative and the total payoff distributed among players, cannot surpass the coalitional value. In our case, this feasibility requirement will be stricter. The total payoff should be equal to the coalitional value, i.e., $\sum_{i \in C^j} x_i = v(C^j)$ for every $j \in \{1, ..., k\}$. In other words, all of the payoff will be shared among the members of the coalition. In this case, we say that the payoff vector is *efficient*. If additionally, it satisfies *individual rationality*, i.e., $x_i \geq v(\{i\})$ for all $i \in N$, then the payoff vector x is said to be an *imputation*. Given a payoff vector x, we write $x(C)$ to denote the total payoff of coalition $C \subseteq N$ under x.

2.6 Solution Concepts

In the previous section, we defined the outcome of a game. In practice, some outcomes are more preferable than others. The partition into coalitions and payoff distribution should be such that no player has any incentive to deviate. Furthermore, agents' payoff should be proportional to their contribution. In other words, *stability* and *fairness* are key factors in coalition formation. We can use these two notions to evaluate the outcomes of a game. In light of this, *solution concepts* come into the picture. According to Osborne et al. [17], "a solution concept assigns to each game a set of outcomes. It captures the consequences of a natural line of reasoning for the participants in a game; it defines a set of arrangements that are stable in some sense". Solution concepts vary depending on the kind of the game (cooperative - non-cooperative), or the focus of the solution (stability - fairness). Some of the most famously used include the Nash equilibrium [17] and the Shapley value [18]. In our work, we will be focusing on solution concepts for cooperative games, and specifically on the following two; the *core* and the *kernel*.

The core is a solution concept that requires that no subset of players in a coalition have any incentive to deviate from it. If they can take a joint action that will lead them to a better result, then we say that the outcome is *unstable*. Subsequently, the set of *stable* outcomes is called the core.

The ***core*** *C(G) of a characteristic function game G = (N, v) is the set of all outcomes (CS, x) such that x(C) ≥ v(C) for every C ⊆ N.* [14]

From the definition, we can see that if $x(C) < v(C)$ for some $C \subseteq N$, then it would be more beneficial for the players to depart from this coalition structure and form a new coalition of their own. Furthermore, because the outcomes in the core are stable, they are more likely to occur in a game. However, in some games the core is *empty* which means that there is no way of distributing the value $v(N)$ to which there are no objections. To make things clear, we present an example from Osborne et al. [17]. Suppose a three-player game that can be modeled as such: $N = \{1, 2, 3\}, v(N) = 1, v(S) = a$ whenever $|S| = 2$, where $a \in [0, 1]$ and $v(\{i\}) = 0$ for all $i \in N$. The core of this game is the set of all non-negative payoff profiles $(x1, x2, x3)$ for which $x(N) = 1$ and $x(S) \geq a$ for every two-player coalition S. Hence the core is nonempty if and only if $a \leq \frac{2}{3}$. Now if we slightly change the game as:$v(N) = 1 = v(S)$ whenever $|S| = 2$, and $v(\{i\}) = 0$ for all $i \in N$, we observe that if player i gets a positive payoff then the other two players automatically get less than 1, so they can object. Thus the core is empty.

The second solution concept we present is the kernel [19]. We say that an outcome is in the kernel if no player can credibly demand part of some other player's payoff. To

understand this, we must first define some other notions. For a CFG game and a coalition C, we define the *excess* of the coalition as:

$$e(C, x) = v(C) - x(C) \tag{2.1}$$

where $x(C) = \sum_{i \in C} x_i$, the total payoff of the coalition. If the excess is positive then it represents the amount that the coalition must renounce for the imputation x to be implemented. If it is negative then its absolute value measures the extra amount over the worth of C that the coalition will gain from the implementation of this imputation of x; it is C's *surplus* in the social order [17]. For any two players i and j, and any imputation x, we define the surplus $s_{ij}(x)$ to be the maximum excess of any coalition that contains i but not j. Formally:

$$s_{ij}(x) = \max\{e(C, x) | C \subseteq N, i \in C, j \in N \backslash C\} \tag{2.2}$$

The surplus of a player i over a player j corresponds to the amount that player i can obtain without player's j cooperation. If $s_{ij}(x) > s_{ji}(x)$, then player i should be able to demand a share of player j 's payoff, that is unless player j already receives the smallest payment that satisfies the individual rationality condition, i.e., $v(\{j\})$. Using this, we can formally define the kernel [14] as follows:

*The **kernel** K(G) of a superadditive game G [14] is the set of all imputations x such that for any pair of players (i, j) we have either:*

1. $s_{ij}(x) = s_{ji}(x)$; or
2. $s_{ij}(x) > s_{ji}(x)$ and $x_j = v(\{j\})$; or
3. $s_{ij}(x) < s_{ji}(x)$ and $x_i = v(\{i\})$

Contrary to the core, the kernel is guaranteed to be non-empty since it always contains the *nucleolus* [20], a solution concept that is closely related to the kernel. The nucleolus is based on the notion of *deficit*. Formally, given a superadditive game $G = (N, v)$, a coalition $C \subseteq N$, and a payoff vector x for this game, the deficit of C with respect to x is defined as $d(x, C) = v(C) - x(C)$: this quantity measures C's incentive to deviate under x [14]. To understand this notion, compared to the kernel where an objection is made by a single player, in the nucleolus an objection is made by a coalition. Therefore, according to the nucleolus, an imputation is deemed unstable if the excess of coalition C can be reduced without increasing the excess of a different coalition to a level as large as that of the original excess of C [17].

2.7 Gale - Shapley Algorithm

The deferred acceptance algorithm proposed by Gale and Shapley in 1962 [21] is an algorithm that produces a solution to the matching problem in polynomial time. They developed a straightforward model of two-sided matching that nonetheless influenced significantly the market design and sparked a substantial growth of literature [22]. The process involves two disjoint groups, where each side has preferences for the other in the form of rank order lists. Agents on one side make proposals to the other side which they can accept or decline.

The Gale-Shapley (GS) algorithm works in stages. First, one side makes offers to the their most preferable option. The other side accepts if the individual is in their preference list, otherwise, they decline. The process does not end there. The ones whose offer was declined, make a second offer to their next option in the list, if it exists. Now the other side will accept the new offer if it is more preferable to the one they have already accepted. Then the agents on the other side that are still not paired or were replaced in the previous step, will make an offer to their next option and so on until either everyone is matched or there are no more options for someone. GS's pseudocode can is in algorithm 2.

Algorithm 2: GS($men, women$)

```
1  Initialize men and women as free;
2  while (∃ free man with woman to propose to) do
3      m = free man;
4      w = his most preferred woman;
5      if (∃ (m', w) pair) then
6          if w prefers m to m' then
7              Erase (m', w);
8              Make pair (m, w);
9      else
10         Make pair (m, w);
```

What is interesting about the GS algorithm is that despite its simplicity, the matches it produces are *stable*. As shown in [21], once the matches have been finalized, there are no agents on either side that would prefer to leave their pair and form a new one together. If that was the case, they would have matched when the proposal was made as at every stage the agents who receive the proposals can replace their pair if a more preferable

appears. That means that their offer was denied at some point. To make things clear, we present an example. Let us consider a 1-1 pair problem, like the stable marriage problem presented in the original paper. Our two disjoint groups are men and women who want to form a pair in order to dance. The preferences of each agent can be seen in the table below.

	w1	w2	w3
m1	3,3	1,3	2,1
m2	3,2	1,2	2,2
m3	3,1	2,1	1,3

Table 2.1: Dancers' preferences.

The first number of each pair depicts men's ranking over women, while the second is the opposite. Consequently, m1's first choice is w2, then w3 and last w1. Similarly, w1's ranking order is m3, m2, m1. GS starts with the men making proposals to the women. Each one will ask their first choice to dance. This gives us the requests:
$m1 \to w2, \quad m2 \to w2, \quad m3 \to w3$
We notice that m1 and m2 have both proposed to w2. She will now accept the one that ranks higher in her preferences, i.e., m2. This will produce the *temporary* pairs:
$(m1, -),\ (m2, w2),\ (m3, w3)$
At this stage, m1 has not managed to match with his first option so he will propose to his next one, w3. By looking at Table 2.1, we see that m1 is the first choice for w3, thus she will break her current pair and form a new one with m1.
$(m1, w3),\ (m2, w2),\ (m3, -)$
Now that m3 has no pair, he makes a proposal to his second option, w2. We notice that m2 who is currently matched with w2 is her second choice, while m3 is her first. Therefore, she chooses m3 and the pairs become:
$(m1, w3),\ (m2, -),\ (m3, w2)$
Since m2 now is alone will try again to match by proposing to his second option, w3, who rejects him because she is already matched with her first option, m1. This leads to m2 now proposing to his last choice, w1. Because w1 has not accepted any proposals so far, she accepts and the final matches are:
$(m1, w3),\ (m2, w1),\ (m3, w2)$

We can see how at each step, each agent chooses the best option available to them. Therefore, at the end of the algorithm no one can do better than the match they already have. By the definition we gave in Section 2.6, the matches produced by GS belong in the *core*. Furthermore, apart from stable, the solution is also *optimal* for the party making

the proposals. This means that if the algorithm starts with the women proposing to their first choice, we would get a different stable matching which would be optimal for them. Of course, it is possible for the optimal solution to be the same for both parties, like in the above example. If we begin the algorithm with the women making the first step, we would still get the same solution. To understand how the result can differ, we present a simple example. We have the preferences of Table 2.2. If we follow the same process as before where the men make the proposals, the stable matches are $(m1, w1)$, $(m2, w2)$. Now if the women make the proposals, the stable matches are $(m1, w2)$, $(m2, w1)$. Both of these solutions belong in the core, however, the optimal solution is different for every party.

	w1	w2
m1	1,2	2,1
m2	2,1	1,2

Table 2.2: Simple table of preferences.

GS guarantees stability but in order to achieve that, agents sometimes will need to make sacrifices and settle for a match that is not their first choice (or even the second). More formally, in *two-sided* games where we have two disjoint groups with *strict* preferences over each other, i.e., no agent is indifferent between any two possible partners, the game has a non-empty core which contains the matching produced by each deferred acceptance procedure. The matching is the best for the side making proposals, and the worst for the one accepting them [23]. This statement can be extended to many-to-one games, like the "college admissions" problem presented in the original paper. It is a two-sided game where each college accepts a quota of applicants. The process is similar to the one we have already described. Each side has strict preferences. Students make applications to their first college of choice. If the number of applicants is less or equal to the quota, the college accepts them temporarily if they are included in its preference list. If the applicants are more than the positions offered, the ones with the lowest ranking will be rejected. In the next step, they can apply to their next choice. The process terminates either when all students get matched with a college, or when they run out of options. The result is in the core.

So far we have seen how GS produces stable solutions for two-sided games. We will now evaluate stability in a different class of games. We consider the "roommates problem" presented in Gale and Shapley's paper. An even number of people wish to be split into pairs of roommates. In this type of *one-sided* problem, any person can be matched with

any other person, contrary to the two-sided problems we have discussed so far. We have the people *a, b, c, d* where *a* ranks *b* first, *b* ranks *c* first and *c* ranks *a* first, while they all rank *d* last. It can be seen that regardless of *d*'s ranking, there can be no stable matching since, no matter who ends up with *d*, they will want to move out and there will always be someone who has an incentive to take them in. Therefore, the core of this game is empty[1].

The GS algorithm influenced greatly the research on deferred acceptance algorithms and their application in practical matching mechanisms. They are the foundation for several labor market clearinghouses. Similarly to the "college admissions" problem in the paper, school choice systems in Boston and New York adopted this model. The deferred acceptance model was also used to match hospitals with doctors in the 1950's. As it was later proved, 10 years before GS, a similar algorithm was invented to solve the "hospitals-residents problem" [25] of matching hospitals and doctors with ranked preferences over each other. Even though the algorithm is different than GS, the outcome it produces is in the core and it is precisely the same as GS's, i.e., optimal for the hospitals. That being said, Gale and Shapley, and the designers of the hospitals-residents algorithm, were not aware of each other's implementation until the mid-1970's.

In our work, we employ the GS algorithm to efficiently allocate passengers to vehicles in a stable manner. Consequently, our two disjoint groups comprise drivers and passengers. Unlike the marriage problem that yields one-on-one matches, our scenario involves many-to-one matches, as we match a driver with multiple passengers (since passengers are placed in the driver's vehicle). The process followed will be thoroughly analyzed in Sction 3.2.2.

2.8 Preference Aggregation

In the previous section, we saw the functionality of the GS algorithm and how it is employed to match drivers with passengers. In this section, we will delve into *preference aggregation*, a crucial aspect of our thesis, as it is used to match passengers with one another based on their preferences. Preference aggregation refers to the process of combining the preference rankings of multiple individuals for two or more social alternatives into a unified, collective preference ranking (choice) over these alternatives [26]. It lies at the center of *social choice theory* [27], which studies collective decision mechanisms where individual preferences are aggregated to make joint choices. Before we introduce some decision mechanisms, we must first define the basic framework.

[1]The set of matches for the roommates problem can be stable under certain conditions. A substantial literature has grown around this issue and the performance of the algorithms that can produce such matches. See Irving, 1985 [24]

For a set of individuals $N = \{1, 2, ...n\}$ where $n \geq 2$, and a set of social alternatives $X = \{x, y, z, ..\}$, each individual $i \in N$ has a *preference ordering* R_i over these alternatives, i.e., a *preference relation* on X. For any $x, y \in X$, we have:

- xR_iy (or $x \succcurlyeq y$): individual i weakly prefers x to y
- xP_iy (or $x \succ y$) if xR_iy and not yR_ix: i strictly prefers x to y
- xI_iy (or $x \sim y$) if xR_iy and yR_ix: i is indifferent between x and y [26]

*A **preference relation** on the set X is a complete and transitive binary relation on X. Specifically, a binary relation R on the set X is **complete** if for all members x and y of X, either xRy or yRx (or both). A complete binary relation is, in particular, reflexive: for every $x \in X$ we have xRx. Additionally, a binary relation R on the set X is **transitive** if for any members x, y and z of X for which xRy and yRz, we have xRz.* [28]

In order to aggregate preferences to reach a decision, we require a mechanism that will address how these individual preferences should be weighted and combined to form a representative collective preference. This leads us to the concept of *preference aggregation rules*, which map the preferences of the individuals into single "social" preference relations.

*A **preference aggregation rule**, f, is a function that assigns to each profile $\langle R_1, R_2, \ldots, R_n \rangle$ (in some domain of admissible profiles) a social preference relation $R = f(R_1, R_2, \ldots, R_n)$ on X. Here, a **profile**, denoted as $\langle R_1, R_2, \ldots, R_n \rangle$, represents a combination of preference orderings across individuals.* [26]

Voting systems are such rules where voters express preferences over candidates, which are then aggregated to elect a winner. The representation of preferences varies depending on the voting system. For instance, many systems require preference lists that rank the candidates in descending order according to the degree of preference. To further clarify, we provide examples of such mechanisms and explain how they work.

An important class of voting systems, are *scoring protocols*. In these protocols, a scoring vector determines how many points a candidate receives by being placed at a certain position in a vote. For m candidates, a scoring vector has the form:

$$\alpha = (\alpha_1, \alpha_2, ..., \alpha_m)$$

where the α_i are non negative integers satisfying $\alpha_1 \geq \alpha_2 \geq ... \geq \alpha_m$. Thus, a candidate taking the ith position in the preference list of a voter receives α_i points from this voter. Whoever scores the most points wins; it is possible that there are several winners [15]. We will be focusing on two examples of scoring protocols; *plurality voting* and *Borda count*.

In plurality voting [15], one of the simplest scoring rules, only the first choice of alternatives gets a point, all others get zero points. Therefore, in this case, the scoring vector is:

$$\alpha = (1, 0, ..., 0)$$

To make things clear, let us consider the following example. Alice, Bob and Mary want to decide what to eat. The available alternatives are: Pizza, Burger and Sushi. Their preference profiles can be seen in Table 2.3 where the order suggests the degree of preference, e.g. Alice's first choice is Burger, her second is Pizza and her last is Sushi.

Alice	Burger > Pizza > Sushi
Bob	Pizza > Sushi > Burger
Mary	Burger > Pizza > Sushi

Table 2.3: Preference profiles

The results of plurality voting can be seen in Table 2.4. As explained, only the first choice of an individual is awarded a point. The points for each alternative are summed and the winner in the one with the highest score; in our case, the three friends will eat burgers.

	Pizza	Burger	Sushi
Alice	0	1	0
Bob	1	0	0
Mary	0	1	0
Sum of points	1	2	0

Table 2.4: Plurality voting

The second widely known scoring rule we present, is Borda Count [15]. According to this rule, given m alternatives, the first choice is awarded m-1 points, the second m-2, and so on with the last alternative getting zero points. In this case, the scoring vector is:

$$\alpha = (m-1, m-2, ..., 0)$$

Borda Count and its variations find applications in real-world elections, as seen in the Eurovision Song Contest for instance, where the winner is determined by a Borda-like scheme. Applying it to the preference profiles of Table 2.3, we get the results of Table 2.5. As we see, pizza and burger score the same points and thus they are both Borda winners.

	Pizza	Burger	Sushi
Alice	1	2	0
Bob	2	0	1
Mary	1	2	0
Sum of points	4	4	0

Table 2.5: Borda Count

Another significant group of voting systems comprises those rooted in *pairwise comparisons* [15]. In other words, alternatives are compared with each other in individual match-ups. The more preferable one is determined by a majority of voters during these pairwise comparisons. In settings with an even number of voters, ties are possible. The most famous of these systems is Condorcet voting, proposed by the Marquis de Condorcet. According to it, a Condorcet winner is a candidate who is preferred to each other candidate in a pairwise comparison by the majority of the voters [15]. To illustrate this, we consider once again the preference profiles of Table 2.3. Applying the Condorcet rule, we have the results of Table 2.6a, where P=Pizza, B=Burger, S=Sushi. As we can see, burger wins all pairwise comparisons and therefore is the Condorcet winner.

	P-B	B-S	P-S
Alice	B	B	P
Bob	P	S	P
Mary	B	B	P
Winner of comp	B	B	P

(a) Condorcet winner

	P-B	B-S	P-S
Alice	B	B	P
Bob	P	S	P
Mary	B	S	S
Winner of comp	B	S	P

(b) Condorcet cycle

Table 2.6: Pairwise comparisons and Condorcet voting

However, there are settings where there the Condorcet winner does not exist. In these cases, we have a *Condorcet top cycle* expressing the *Condorcet paradox*, i.e., the alternatives in this cycle defeat all alternatives outside of the cycle in their pairwise comparison, yet none of them defeats all other alternatives in the cycle [15]. For example, in the preference profiles of Table 2.3, if Mary's preferences were: Sushi > Burger > Pizza, then from the pairwise comparisons, we get the results of Table 2.6b where B defeats P, P defeats S and S defeats B.

The next voting system we present respects the Condorcet winner as it elects them whenever one exists, but—unlike Condorcet's system—always has one winner or more,

even if there is no Condorcet winner. The type of system that respects the Condorcet winner is also called *Condorcet-consistent* [15]. The one we will be focusing on is *Copeland* voting. In this system, to determine each candidate's points, they participate in pairwise comparisons like before. The candidate that is preferred by the majority of voters, receives one point, while the other receives zero points. In cases of a tie, they both receive half a point. Each candidate's Copeland score occurs after their points have been added and the candidate with the most points is the Copeland winner. To understand this, consider the following example. Alice, Bob and Mary, the three friends of the previous examples, are joined by Jessie, Rose and John and they now need to decide their vacation destination. The candidates are Iceland, Japan, Peru and Morocco and their preference profiles can be seen in Table 2.7.

Alice	Iceland > Japan > Peru > Morocco
Bob	Peru > Japan > Morocco > Iceland
Mary	Peru > Japan > Morocco > Iceland
Jessie	Morocco > Japan > Iceland > Peru
Rose	Iceland > Peru > Japan > Morocco
John	Iceland > Peru > Morocco > Japan

Table 2.7: Vacation destination preference profiles

	I-J	I-P	I-M	J-P	J-M	P-M
Alice	I	I	I	J	J	P
Bob	J	P	M	P	J	P
Mary	J	P	M	P	J	P
Jessie	J	I	M	J	M	M
Rose	I	I	I	P	J	P
John	I	I	I	P	M	P
Winner of comp	?	I	?	P	J	P

Table 2.8: Pairwise comparisons with ties

Pairwise comparisons on these profiles yield the results presented in Table 2.8, where I=Iceland, J=Japan, P=Peru, M=Morocco. As observed, in two head-to-head contests, there are ties, and thus, no Condorcet winner can be elected. However, a Copeland winner can be determined. Each candidate's score is calculated as:

- $C\_score(I) = 1 + 0.5 + 0.5 = 2$
- $C\_score(J) = 1 + 0.5 = 1.5$
- $C\_score(P) = 1 + 1 = 2$
- $C\_score(M) = 0.5$

Hence, there are two Copeland winners in this case, both having the highest scores among the candidates.

2.9 Related Work

In this section, we present ridesharing-related work. We begin with a literature review of research conducted in the ridesharing field. Subsequently, we list platforms used either by developers for simulating and testing ridesharing algorithmic approaches, or by users as shared mobility transportation providers. Finally, we discuss in detail previous work that we utilized in this thesis.

2.9.1 Ridesharing problem: Literature Review

Ridesharing is a widely studied topic that researchers have approached from different angles and with different objectives. Alonso-Mora et al. [3] proposed a real-time, high-capacity ridesharing model that scales to thousands of users and millions of trips and can dynamically produce optimal routes. Their algorithm assigns a batch of requests to vehicles by solving an integer linear program (ILP) while also including "rebalancing" of the fleet, i.e., idle vehicles can be distributed to areas of high demand. It can be used with vehicles of varying capacities and can effectively reduce the number of vehicles needed to serve riders' requirements. The algorithm was tested in the New York City taxicab dataset and it showed that 13000 single-rider vehicles can be replaced with 3000 vehicles with a capacity of 4, or 2000 vehicles with a capacity of 10 and can handle 98% of the demand with mean waiting time of 2.7 and 2.8 min and mean delay of 2.3 and 3.5 min, respectively.

Simonetto et al. [29] modified the above model by assuming that the requests processed at the same time cannot be merged. i.e., at most one new request can be assigned to each vehicle at a time. This reduces the computational load and the algorithm is based on a LP instead of an ILP. This restriction does not affect the performance as it can be covered with more frequent sampling of the requests. Their approach is constructed using a federated architecture designed to optimize computational efficiency and can be distributed among

multiple companies without compromising privacy. The results showed that it is 4 times faster than Alonso-Mora et al.'s and is also accessible from simpler hardware. Finally, it can be adopted by smaller companies as a small percentage of the requests (5% - 20%), with a respective percentage of the fleet, can achieve comparable results in terms of service rate.

Stiglic et al. [30] investigated the advantages of implementing meeting points in ridesharing, with regard to cost savings and congestion mitigation. In their work, they evaluate a system of private drivers who wish to share their rides instead of "taxi hailing" type of companies, like Uber. Meeting points give more flexibility which as they show, can result in additional matches in cases where the riders are too far for the driver to pick them up. Furthermore, meeting points can reduce the overall extra distance that the drivers have to cover in order to serve the riders. Their approach supports more than one rider per car, but only if they can be picked up and dropped off from the same meeting points. This allows fewer detours and stops in order to maintain the ride pleasant for the passengers. However, this restriction can be quite limiting as this is applicable only when the riders are close to each other.

Fagnant et al. [31] propose a dynamic ridesharing model focused on optimizing a shared autonomous vehicle (SAV) system in the city of Austin, Texas. SAVs are assigned to the nearest rider in FIFO order and multiple travelers can be picked up as long as each rider's travel time is not excessively compromised. They address the challenge of determining the optimal fleet size for the SAV system, with respect to day-to-day demand variations, in a time frame of a year. Their results show that dynamic ridesharing can decrease the size of the fleet, total service time and also reduce the overall cost. Finally, they investigate the profitability of SAV systems for future investors with their simulations showing that for a 70,000$ per new SAV purchase, an annual return of 19% is expected if the cost charged per mile is 1$.

A cooperative game theoretic approach is examined by Bistaffa et al. [2], where riders are connected via a social network. A Graph Constraint Coalition Formation is used, where possible coalitions are dictated by the graph/network. This method prioritizes reducing the overall system's travel expenses while ensuring the participants' spatial and temporal preferences are met. Additionally, the authors developed a cost-sharing scheme that produces kernel-stable payments for thousands of agents. In particular, when tested on 2,000 agents, the payments were calculated in less than an hour. The results showed that their ridesharing model improved social welfare by 36.22%.

Nourinejad and Roorda [32] introduced a decentralized agent-based model for dynamic ridesharing in which the matching between drivers and passengers is determined through a single-shot first-price Vickery auction. Their model supports multi-passenger rides as

well as multi-driver ones, in which passengers can "hop" from one car to the other to get closer to their destination in cases where they cannot find a single ride. It is tested on the Sioux Falls network of 24 nodes, 76 transportation links and 1.96 million users. The results show that ridesharing induces cost reduction for individual users along with overall kilometers traveled savings. Finally, while the decentralized approach proves to be less computationally demanding, it does not compromise the quality of service.

In this thesis, we take into consideration agents' preferences while matching them. More specifically, drivers are matched with passengers based on both spatial and preferential criteria. Furthermore, we apply a second level of preference aggregation, which is used to match passengers in the same car. With preference satisfaction rates high, we can provide a more enjoyable experience for all individuals involved in the ride. Additionally, the matches are in the core, i.e., are stable; and the payments are in the kernel, making them both stable and fair.

2.9.2 Ridesharing-Related Real-World Platforms

As the population in the cities keeps rising and car congestion is getting worse, ridesharing is becoming a hot topic in research. Many companies offer platforms for the development of Mobility as a Service (MaaS) algorithms and fleet management, which can be used to simulate ridesharing scenarios. Aimsum, in its Ride Research program, offers a platform to researchers where they can test their code. The simulation includes trip requests, the status of congestion in the network and the status of the vehicles. The developers through their algorithms can regulate the fleet and take feedback on the performance such as average waiting time for riders and average idle time for vehicles. Likewise, SUMO is an open-source traffic simulation platform that can be used to test algorithms in urban environments. MATSim provides a framework for large-scale agent-based transport simulations. It comprises multiple modules that can be combined, utilized independently or even replaced with custom implementation in order to evaluate specific aspects of one's work. TransCAD is a Geographic Information System (GIS) that can be used for transpiration planning. It includes simulation capabilities for modeling ridesharing transportation systems of any scale and level of detail. Lastly, PTV Vissim is a multimodal traffic simulation software that integrates all modes of transportation. It is capable of emulating the traffic patterns of all road users and also tackling issues such as road congestion and vehicle emissions, making it a dependable tool in evaluating the performance of ridesharing algorithms.

Long and expensive urban commutes, as well as the wide use of smartphones, have given rise to another category of platforms associated with ridesharing, which involves

shared mobility service providers. Companies known for their "taxi-like" services, are now extending them to include ridesharing scenarios in which as we explained, passengers can pay a smaller fare if they share their ride. Currently, there are ridesharing companies operating all over the world available to users through their smartphones. Uber, the most famous in its category, offers UberXShare, an extension to the classic private rides that Uber is known for. You can match with a rider alongside your route and save money on the trip with at most an 8-minutes delay. It operates in over 70 countries around the globe. Lyft Shared is Lyft's ridesharing option for the U.S. and Canada. It allows passengers to share rides, considering that they are going to similar destinations. GrabShare is Grab's shared ride service available in Southeast Asia and Ola Share is Ola's in India.

Depending on the needs of the user, ridesharing services can focus on different goals. Unlike the services listed so far, Via sees itself more like "a public transit service than as a ride-hail equivalent" as it tries to fill as many seats as possible in vehicles headed to popular destinations. Riders will likely have to walk to get to the pick-up point and their destination. Via is available in over 35 countries worldwide. Contrary to the common urban services, BlaBlaCar is a long-distance ridesharing platform that connects drivers with available seats in their cars with passengers traveling in the same direction. It operates in most countries of Europe as well as in Mexico, Brazil, India and Turkey. Finally, GoKid is not an app for the daily commuter. Instead, it helps organize carpooling among parents to arrange rides for their kids to and from school, as well as other activities. To aid scheduling, it includes features such as in-app messaging and calendar synchronizing.

2.9.3 The basis for this work

Before we begin analyzing our approach to the Ridesharing problem in the next chapter, we first have to mention the work that served as the basis for ours. In his thesis [1], Emmanouil Pagkalos developed a novel model for preferences-aware ridesharing, which was tested through simulations of ridesharing scenarios for the four capital cities of Crete. By taking data from OpenStreetMap[2], he created corresponding city graphs for each capital, and assigned agents to random locations within the graph. Using information like their position and destination, the thesis defined the *service area* of a driver, from which they can pick up and drop off passengers. This information along with their preferences, was used in order to measure the value of a possible collaboration between a driver and a passenger and form the hypergraph of each city. In the hypergraph, each hyperedge represents a car and the nodes it encloses, the passengers. Coalitions are formed given the hypergraph information and the values calculated for each driver-passenger relationship.

[2]https://www.openstreetmap.org/

To do so, the novel PASR-CF (Preferences Aware Social Ridesharing - Coalition Formation) algorithm is employed, which was inspired by Dang et al.'s [33] greedy algorithm. The process starts with every passenger trying to form a coalition with the driver with whom they have the highest value (Equation 3.4) in the hyperedges. If the number of passengers who try to get in a car is more than the car's capacity, i.e. 4, then the algorithm finds the "optimal" coalition out of all the possible combinations of the passengers. This coalition is the one with the highest coalition value which is calculated via Equation 2.3.

$$v(C) = \sum_{A \in C} v_D(A) + \sum_{\substack{i,j \in C \\ i \notin j}} w(i,j) \tag{2.3}$$

where C, a possible coalition combination, A the passengers in it, $v_D(A)$ the value representing the compatibility between the driver and a passenger from Equation 3.4, and $w(i,j)$ the preference satisfaction rate between two passengers. To form the feasible combinations and find the optimal coalition out of all the feasible ones, the author adapted Binshtok et al.'s [34] *Branch and Bound* algorithm.

After PASR-CF terminates, the coalition structure CS has been formed. Then, for every coalition in CS, the route that the driver must follow to accommodate all passengers is extracted. This route must be such that starts from the driver's starting point, picks up and drops off every passenger before reaching the end of the route which is the driver's destination. However, it is not apparent which sequence of stops should be followed in the process. The optimal sequence of stops must be calculated in order to keep the extra distance that the driver will cover minimal. With this in mind, the author created the CFRE (Coalition's Final Route Extraction) algorithm which also uses the Branch and Bound method. The implementation is based on Search Trees, where potential routes are explored using Depth First Search. Each node contains a point and information about the cost up to it. The cost between two nodes is calculated using the Euclidean distance. Thus, while traversing the tree, in order to calculate the cost up to a specific point, you calculate the distance between the current node and the next and then you add it to the current cost (cost = distance x cost/km). Every time a leaf is reached, i.e. the destination of the driver, if the cost is lower than the current minCost, this path is shorter and minCost gets updated. Likewise, if at any node of the path the cost exceeds that minCost, the path gets discarded and the tree gets pruned. CFRE returns the optimal sequence of stops which is then used to form the path. The stops, i.e. the starting and finishing nodes of passengers, are connected through Dijkstra's algorithm and the final route is complete. The implementation of route extraction was influenced by the route

computation of Bistafa et al. in [35]. In their approach, they used the A* algorithm to create the path. As Pagkalos pointed out, in their work [1] they experimented with A*, but it didn't yield satisfactory results, leading them to ultimately opt for Dijkstra's algorithm.

In the final step of his work, he proposes how the cost of the route should be distributed among the agents of the coalition. His approach is straightforward and depends on the extra distance the driver has to cover. Formally, the driver's cost is calculated by Equation 2.4.

$$driverCost = \frac{initCost - (1 - \frac{init_{KM}}{final_{KM}}) \cdot initCost}{|C|} \tag{2.4}$$

If the extra distance $(final_{KM} - init_{KM})$ is equal to 0, the cost is split evenly among the agents, including the driver. Otherwise, the higher the extra distance, the less the driver's cost. The remaining cost is paid by the passengers. This is done by calculating how much each one must contribute, according to the distance of their trip (start to destination). For every passenger, a coefficient is computed as in Equation 2.5.

$$c_i = \frac{pass_i\_dist}{\sum_{j \in P} pass_j\_dist} \tag{2.5}$$

where P, is the set of passengers in the coalition. Each coefficient is multiplied by the remaining cost to find the amount each passenger must pay.

We will now clearly indicate the parts of the above work that we "inherited" in ours.[3] In the diagram of Figure 2.8, we present the two approaches in parallel. On the left side is the work of [1], while on the right is ours. In the middle are the parts that we share. It is evident that both of us start from a common basis. We used the same city graphs to simulate the Ridesharing scenarios and create the hypergraphs; however, we experimented with the spatial criteria in which we placed the passengers in a driver's hyperedge. As depicted in the service area of the diagram, we explored two additional forms it can have. Then, as far as the coalition formation is concerned, it occurred at two distinct levels. First, we employed the Gale-Shapley algorithm to match drivers with passengers, and subsequently, we applied preference aggregation to match passengers with each other. For the route extraction and path formation, we also used the CFRE and

[3]For these parts we utilized code from [1], which we adapted to fit our implementation.

Dijkstra's algorithm respectively, as we perceived their use to be an appropriate choice and, additionally, we did not want to place too much emphasis on this particular part. Finally, to determine how the cost of the ride should be partitioned among the members of the coalition, we applied a transfer scheme that converges to the kernel, ensuring fairness. Every stage of the process will be analyzed thoroughly in the following chapter.

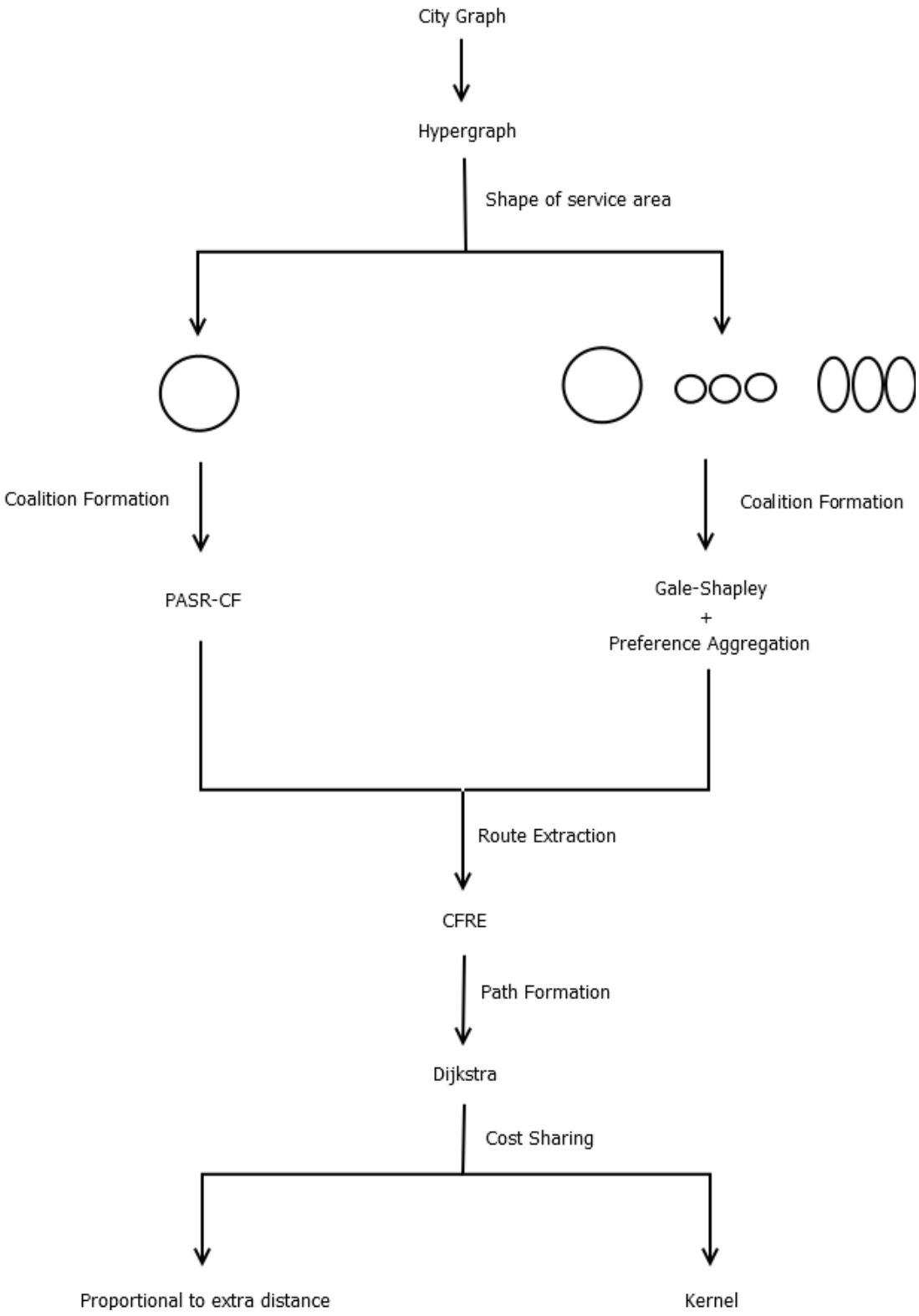


Figure 2.8: Comparing the steps taken in [1] to ours. Left side indicates steps unique to [1], right side indicates steps unique in our work.

3

Our Approach

In this chapter, we present our approach to the ridesharing problem. We start by defining the constraints that characterize a feasible coalition as well as those that contribute to a pleasant one. We elucidate the use of the hypergraph to restrict the potential candidates for each coalition by introducing the concept of a drivers' service area and examining different cases for it. Subsequently, we describe the application of the Gale-Shapley algorithm in forming the coalition structure, and analyze the route extraction process for each coalition. Finally, we delve into the payment allocation problem and into how the kernel solution concept was applied to generate stable and equitable payments.

3.1 Feasible Coalitions

In general, for the coalition formation process, one needs to define the characteristics these coalitions will have. In this work, we study the ridesharing scenario, where coalitions represent vehicles with one driver and some number of passengers which depends on the capacity of the vehicle and the route each agent wants to follow. In order to model this setting we apply several constraints which will help guarantee the homogeneity of the coalitions.

3.1.1 Constraints

We will now provide a definition for the aforementioned constraints. Since every coalition must satisfy them, they comprise the list of *hard constraints* and they are listed below.

- If $|C| \geq 1 \Rightarrow \exists \quad r_i \in C : \quad r_i \in D \; and \; |C| \leq capacity$,
  where C represents the coalition, D the set of drivers and r_i a driver. Each coalition can have exactly 1 driver.
- $|C \cap D| = 1$
  The number of drivers per coalition is exactly one.
- If $r_i \in D : \; P^1 = r_i^s \wedge P^n = r_i^d$,
  where P^1, P^n, the starting and finishing point respectively of the route. Coalition's route begins at driver's starting point and finishes at driver's destination.
- $\forall \; r_i \in C : P^x = r_i^s \wedge P^y = r_i^d \Rightarrow x < y$
  For every agent in coalition, their starting point precedes their destination.
- $\nexists \; C_i, C_j \in CS : \quad C_i \cap C_j \neq \emptyset \quad \Rightarrow \quad \sum_{i \in CS} |C_i| \leq N$
  No agent can belong to more than one coalition. Coalitions are disjoint.

Our work focuses on a preference-associated ridesharing problem, i.e., the formation of coalitions based on the preferences of both drivers and passengers. As such, we consider that each individual has a set of attributes, as well as a set of preferences regarding the attributes that their fellow participants have. In our model, such preferences represent the *soft constraints* of the problem and they are listed as follows.

- **Gender**: Agents can have a specific preference for matches with males or females or, no preference at all.
- **Employment**: Agents can specify their preference to be matched with Students or Employed people.
- **Age range**: Agents can specify a broad preference for the age of their matches, e.g. 20 - 30.

Given that these are soft constraints, an agent can have a preference for one or more of them, but also for none. The way they affect the matching is that they make for a more

pleasant ride if satisfied. With this in mind, the goal is to match agents with as many soft constraints satisfied as possible, without also causing too many detours for the drivers. For this reason, we need to calibrate their influence in the coalition formation process. This trade-off is analyzed in depth in Section 3.1.2.2.

3.1.2 Hypergraph

The first step towards matching passengers and vehicles is the *hypergraph* formation. The hypergraph will be used to represent the viable options agents have while trying to form coalitions. Specifically, each *hyperedge* represents a vehicle while the passengers are represented as *vertices*. Hence, the number of hyperedges is equal to the number of drivers, and each hyperedge contains the vertices, i.e., the passengers who can be considered as options for the driver. These options are based on the constraints mentioned above, as well as on the driver's Service Area.

Generally, hypergraphs provide an efficient representation of large-scale problems, as they can break them down into simpler sub-problems. Specifically in our case, clustering agents who match spatially is more efficient than examining all the potential collaborations among them. For instance, agents who move in different areas would cause long detours and thus, an overall inconvenient ride. Additionally, a large number of agents who do not satisfy the constraints for a feasible coalition, i.e., the requirements to belong in a driver's hyperedge, will be excluded from the process, simplifying the problem further.

3.1.2.1 Service Area

As can be easily understood, a driver cannot pick up passengers from any place in the city. This would be costly but also time-consuming, leaving the driver with no motivation to participate in the ridesharing process. As such, it is natural to assume a border of some kind to define the area that is convenient for both the passengers and the driver to share a ride. Therefore, a need for an extra constraint arises. The final constraint that must be satisfied for a passenger to be included in the driver's hyperedge, is that the passenger must be inside the driver's *Service Area*. This is practically an area around the driver's path, in standard distance from its center. The path is created using Dijkstra's algorithm. An example can be seen in Figure 3.1, in which the length of the path is approximately 3 km.

We evaluate three cases of the service area. In the first one, the area has the form of a circle whose diameter is defined by the distance between the starting and finishing point of the driver's path. Thus, candidates for every coalition, are passengers in a radius of half this distance from the center of the circle. An example of this can be seen in Figure 3.2.

Figure 3.1: Driver's initial path

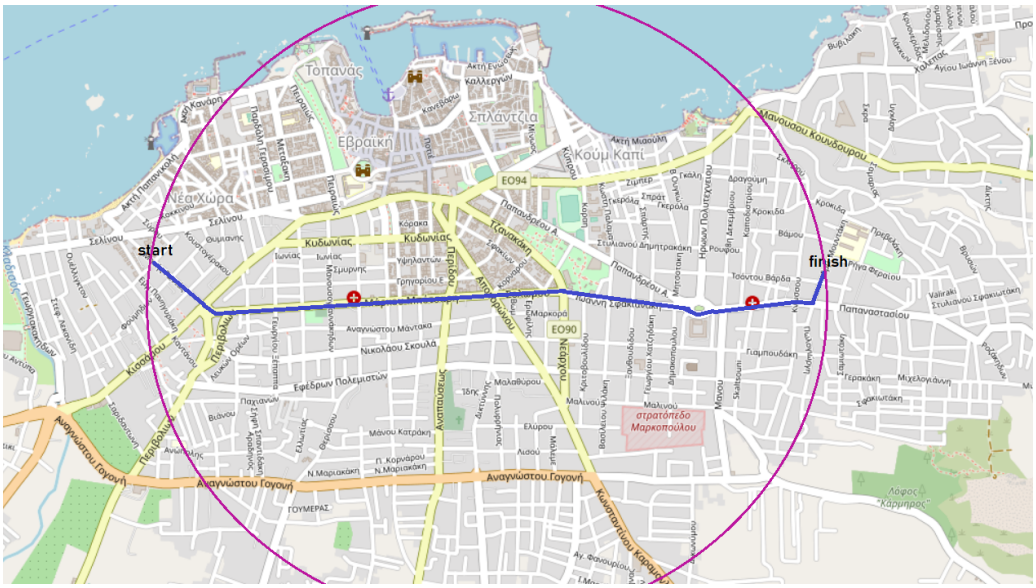


Figure 3.2: First case of service area.

It is important to note that the coordinates of both the start and finish of a passenger's path, must be inside the service area, or in other words, inside the circle that defines it. More formally, if (x_s, y_s) the longitude and latitude of the starting point of a passenger's path and (x_f, y_f) the coordinates of their finishing point respectively then, a passenger is included in a driver's hyperedge iff:

$$(x_s - h)^2 + (y_s - k)^2 \leq r^2 \quad \&\& \quad (x_f - h)^2 + (y_f - k)^2 \leq r^2 \tag{3.1}$$

where (h, k), the coordinates of the center of the circle and r, its radius. It is clear that for larger paths the number of passengers examined is augmented and thus, the size of the hyperedges as well. Furthermore, this implementation is not very flexible with respect to following the course of the path.

While trying to resolve these issues, the second case was implemented. In this, the driver's path is partitioned into sub-paths of standard length. To do so, we define the "cuts", which are the specific nodes in which the path is cut as the name implies. The process is as follows: while traversing the nodes of the driver's path, the distance in km is added to each node and after x km, the path is partitioned. Naturally, the first cut is the start of the driver's path and the last one is the finish. After running some tests, we noticed that the average length of a path is 2 km. Thus, in the experiments, we tested cases for cuts that happen after every x = 0.5 - 2 km (see appendix). There are in total $\lceil \frac{path\_length}{cut\_length} \rceil$ cuts. Each part of the service area is defined using 2 sub-paths, i.e., 3 cuts. This is to avoid discontinuity. For every part there is an equation that defines the area around it. In this case, the equations used are those of a *circle*. Using the start of one sub-path and the finish of the next, the diameter of each circle is defined. To make this clear we present an example. If we take the driver's path from Figure 3.1, the path is approximately 3 km. If we use cuts of 0.5 km, the path is partitioned as in Figure 3.3. The length of every sub-path in this case is 0.5 km. Thus, the service area is formed as in Figure 3.4.

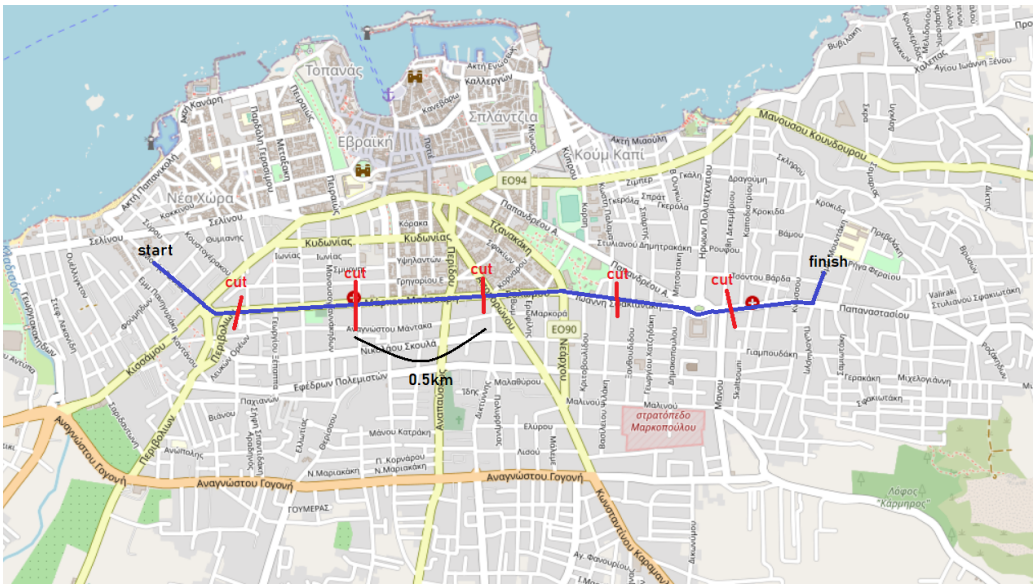


Figure 3.3: Path partitioned from 0.5 km cuts.

In order to include a passenger in the hyperedge in this case, we can simply check whether their starting and finishing points belong in any of the areas these circle equations

define. As we can see in Figure 3.4, the service area is comprised of smaller circles whose diameter is defined by the distance between the cuts that form the 2 sub-paths. For example in the first circle, the start of the path is the first cut, so the diameter is defined by the first and the third cut. In the second circle is defined by the second and the fourth cut and so on. If we assume that the path in this example is similar to a straight line, then the radius of each circle is approximately 0.5 km. Of course, if the path is not linear, the radius of the circle is reduced.

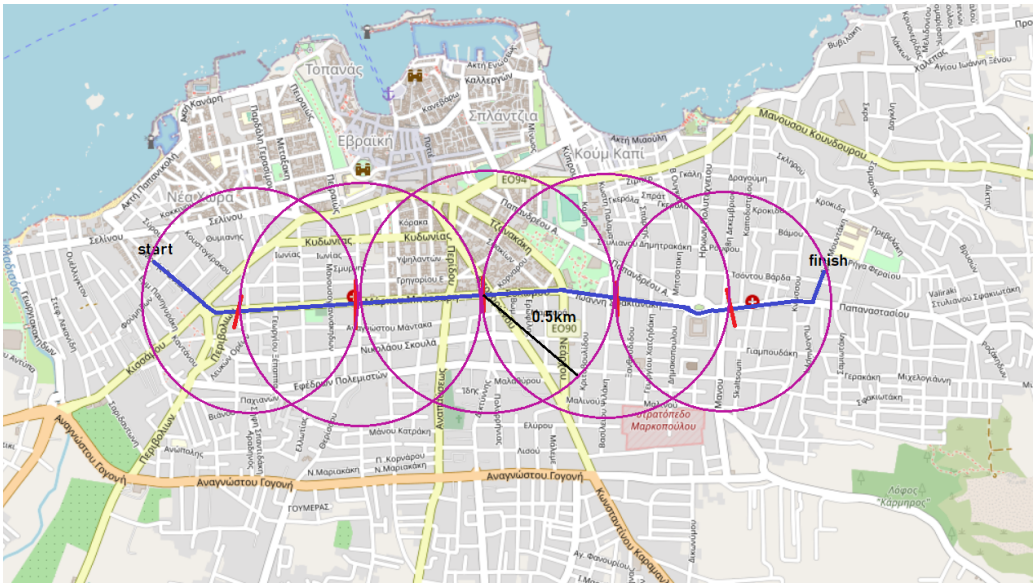


Figure 3.4: Second case of service area for 0.5 km cuts.

The issue with this implementation is that when the length of the cuts is smaller, e.g., 0.5 km, the size of the service area is decreased significantly. For this reason, its shape in the final case changes into an *ellipse*. This is because we want the minor radius to be defined by the cuts alongside the path, but also the major radius should be perpendicular to the path to have better coverage of the area. The equation of the ellipse that the passengers' start and finish must satisfy, is:

$$\frac{((x-h)\cdot cos(\theta)+(y-k)\cdot sin(\theta))^2}{r_{minor}^2}+\frac{((x-h)\cdot sin(\theta)-(y-k)\cdot cos(\theta))^2}{r_{major}^2}\leq 1 \quad (3.2)$$

where, (x,y) the longitude and latitude of the start or finish of the passenger, (h,k) the coordinates of the center of the ellipse, r_{minor} and r_{major} the minor and major radii in the xx' and yy' axis respectively and θ the rotation angle measured from x axis. In the above equation, the minor radius will depict the sub-path of cuts, hence it will be rotated in

this direction. The rotation angle θ will be calculated as such:

$$\theta = arctan(\frac{lat_f - lat_s}{lon_f - lon_s}) \tag{3.3}$$

where (lon_s, lat_s) and (lon_f, lat_f), the coordinates of the start and finish of the sub-paths, respectively. The minor radius is half the distance between the start and finish, while the major is $r_{major} = r_{minor} + \frac{r_{minor}}{2}$, a formula that occurred after experimentation. As above, for an agent to be included in a hyperedge, their starting and finishing points, must both belong in the area these equations define. For every set of equations that they satisfy, the agent is added in the hyperedge of that driver. An example of the third case of the service area can be seen in Figure 3.5. Like before, the radii of the ellipses depicted are approximations regarding a linear path. It is important to note that at this stage, an agent can belong to more than one hyperedges without having the fifth constraint violated since the hyperedges are just options and the coalitions have not yet been finalized.

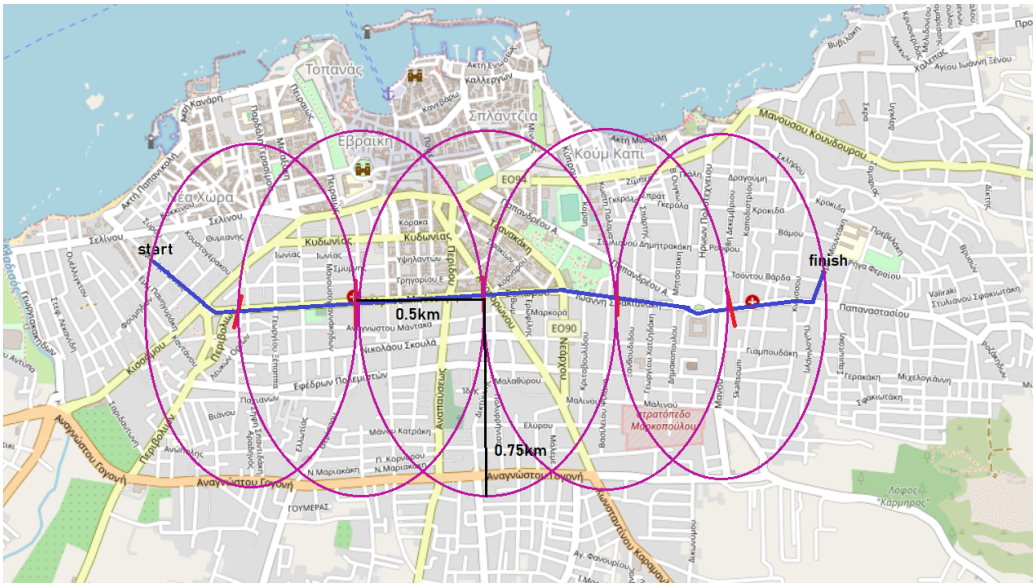


Figure 3.5: Third case of service area for 0.5 km cuts.

3.1.2.2 Values for Hyperedge

At this point, the hyperedges contain all the passengers who based on their route can be accommodated by the driver, but a way to know which agent is best suited for each coalition is also needed. Therefore, for each hyperedge a passenger belongs, a corresponding *value* is calculated. Each value depicts a score for a passenger's participation in the particular coalition with respect to the driver. The score is calculated from three factors: the direction of the passenger's route in relation to the driver's, the distance between the two

paths, and the percentage of soft constraints satisfied (SCS) between the passenger and the driver. Each factor is then multiplied by a coefficient, with all three of the coefficients adding up to one. The first two factors focus on finding the best matches route-wise, while the third one, on the social aspect of this work. By calibrating the coefficients, one can choose the factor they want to place emphasis on. The equation for the value calculation is presented below.

$$v_D = dir \cdot cf_1 + dist \cdot cf_2 + SCS \cdot cf_3 \tag{3.4}$$

and,

$$cf_1 + cf_2 + cf_3 = 1 \tag{3.5}$$

where v_D the hyperedge value representing the driver's relationship with a particular passenger, *dir* the direction, *dist* the distance and SCS the soft constraints satisfied, as we explained above. For example, if the Equation 3.4 is: $v_D = dir \cdot 0.8 + dist \cdot 0.1 + SCS \cdot 0.1$, means that finding a passenger with a path that runs in the same direction as the driver's, is more important than the distance or the preference satisfaction between them. Generally, we want to place passengers in a vehicle that will cause the least deviation from the driver's path possible. To do so, in this work, the direction coefficient will have the greatest value out of the three, since it will allow the driver to continue moving in the same direction as their destination and thus make shorter detours and get a faster and less expensive final route. By calculating the values for each hyperedge this way, routes that have the opposite direction of the driver's, or have a greater distance or no soft constraints are satisfied, get a lower score and these agents are less likely to be chosen in the coalition formation later on.

3.2 Coalition Formation

After the agents have been added to the hyperedges and their values with respect to the drivers have been calculated, the coalition formation process can begin. A *Coalition Structure* (CS) is created, which will include all the formed coalitions. As mentioned before, each coalition represents a vehicle. In other words, for every driver, there is a corresponding coalition and a corresponding hyperedge. For the selection of the passengers, the values calculated above will be used as ordered preference lists, as well as a version of the *Gale-Shapley* (GS) algorithm for stable matching. Since preference satisfaction is a focal point of this work, the matching will occur in two stages. In the first stage, the GS algorithm will assume that the capacity of each car is seven passengers instead of four.

In the second stage, the actual four passengers will be selected after preference aggregation is performed among the seven chosen. This course of action was taken because GS matches two disjoint groups, in our case drivers and passengers, based on the rankings each group assigns to the other. Consequently, it can solely consider the preferences that drivers have for passengers and vice versa. By incorporating the second stage atop the initial one, we can perform preference aggregation over passengers' preferences for each other and therefore guarantee a pleasant ride for everyone.

3.2.1 Ordering of Agents' Preferences

The GS algorithm takes the preferences of two disjoint groups ranked and matches them. Hence, the ranking will be implemented in the form of ordered lists of preferences for every agent. In light of this, two structures are created. Each one will contain the preference lists of drivers and passengers separately. To create those, the hyperedges will be utilized. Since every hyperedge represents the driver's feasible choices for a coalition, and the values show how beneficial the particular matching is for both the driver and the passenger, it follows that:

- For the drivers, the preference lists can be obtained by simply finding each driver's corresponding hyperedge and ordering the passengers it contains by their value for that hyperedge.
- For the passengers, in order to create the preference list for each one, the passenger must be first located in every driver's hyperedge it exists. The list is then composed by those drivers after they have been ordered by the hyperedge value.

The preference lists will only contain agents, and the order they are placed in the list declares that an agent is more preferable to the one after him and less than the one before him.

3.2.2 Gale - Shapley

The next step in constructing the coalitions is applying the Gale-Shapley algorithm. This is the main algorithm used in this thesis since it will match drivers with passengers based on their rankings for each other. The problem presented here, placing passengers in cars, is similar to the College Admissions problem presented in the original paper [21], where we have many to-one matches. The cars correspond to colleges and the passengers to applicants, respectively. The matches produced will be stable, meaning that at the end of the algorithm, the agents will have no incentive to break the pair they are already

in to form a better one. Since GS uses the ordered preference lists mentioned above, each agent's preferences are examined in order. In other words, the algorithm tries to match each agent with their first option and only if the preferences are not bidirectional, it will move to the next one in the list. GS terminates, either after all agents have been matched, or when there are no more options for matching in their lists. After each run, it will return all the Driver - Passenger pairs. The pseudocode for the Gale-Shapley many-to-one algorithm, can be seen in Algorithm 3.

Algorithm 3: GS(drivers_prefLists, passengers_prefLists, max_capacity)

```
1  while ∃ unmatched passenger j who has non empty preference list do
2      for every passenger j do
3          Find passenger's first preference, i.e., driver i;
4          Check if max capacity for driver i's car has been reached;
5          if car(i).capacity < max_capacity then
6              Form match;
7              (i, j);
8              car(i).capacity++;
9          else
10             Find the "worst" passenger k in the car;
11             Compare them with the current passenger;
12             if j better than k then
13                 Replace match;
14                 (i, j);
15                 Free the replaced passenger k;
```

While trying to solve the ridesharing problem, two versions of the GS algorithm were implemented. The first one is based on the classic College Admissions problem mentioned above. The capacity of each car is standard and was set at seven passengers per vehicle. It's important to note that this capacity does not correspond to the actual car capacity. Since GS can only match two disjoint groups using their preferences, the pairs it produces is the aforementioned first stage. They constitute temporary coalitions, based solely on the Driver - Passenger relationship. Selecting a capacity greater than the actual, will make it possible to take into account preference satisfaction among passengers too. This is done by keeping the four passengers out of the seven, whose preferences match best as is analyzed in Section 3.2.3. The capacity was set at seven after preliminary tests

we performed indicated that a lower capacity would not allow for adequate passengers' preference satisfaction, while a higher one might place more emphasis on it and therefore cause detours and a non-preferable route for the driver in general. It is important to note that after the termination of GS, we have temporary coalitions with sizes in the range of 1 - 7. The preference aggregation for the passengers is applied in cases where the size is greater than four. For simplicity, we will be explaining the seven-passenger case, but it is the same for five or six passengers.

The second version of GS utilizes dynamic capacity. In other words, with this implementation, vehicles do not have to be empty for the algorithm to work. On the contrary, it can be used to fill empty seats wherever they exist. Because of this, the second version was used in addition to the first one, in order to fill empty seats in vehicles that have not yet reached their maximum capacity. It replaces the max capacity in line five of Algorithm 3, and is calculated for each vehicle. The dynamic capacity is calculated as

$$dynamic\_cap = max\_cap - passengers\_num \tag{3.6}$$

where, $max\_cap = 4$ is the actual capacity of the vehicles, and $passengers\_num$ is the number of passengers already in the vehicle after the execution of the first version of GS. The dynamic capacity is calculated for every coalition. By filling empty seats, this approach improves the average number of passengers per coalition and thus can help reduce the cost of each participant.

3.2.3 Preference Aggregation among Passengers

GS handles driver-passenger preferences. This section will focus on preferences among passengers in the coalition. An election mechanism is needed in order to figure out which agents are the most preferable out of the ones in the temporary seven-passenger coalition. The first one we considered, was *Borda count*. As we explained in Section 2.8, under this mechanism, each agent assigns a score to each alternative, which depends on its rank in their preference ranking. The preference ranking will occur after we calculate the value of their relationship in relation to every other passenger in the coalition. This value represents the percentage of soft constraints satisfied between them. Then, these values are sorted and the agent with the highest one will get the highest score $k = |C| - 1 = 6$, the second, $k - 1 = 5$, and so on, with 0 being assigned to the relationship with oneself. The same process is followed for every agent, with the final score of each one being the sum of the preference ranking that the others gave them. Ultimately, the passengers with the 4 highest scores, participate in the final coalition. The issue with this implementation was

that it differentiates ties. As it was observed in the experiments, it was pretty common that agents would have the same soft constraints satisfaction percentage. However, while the algorithm traverses the vector of the values from the highest to the lowest, it assigns the ever-reducing score to the corresponding agents. Thus, agents with the same value will get different scores and the outcome would then not reflect the actual preferences of the agents. For a clearer illustration, let us examine the following scenario. Suppose we have three passengers, denoted as p_1, p_2, p_3, among the seven candidates competing for four seats in a car. We assume that between p_1 and p_2, two out of the three preferences are satisfied, and the same holds true for p_1 and p_3. However, even if p_1 has an identical preference for both p_2 and p_3, they will inevitably assign different scores to each, say, giving a score of 5 to p_2 and 4 to p_3. Given that there are only three preferences, this disparity is likely to occur for most passengers. Therefore, the result may not accurately represent passengers' preferences.

In light of this, a second mechanism was developed which tackles the issues of Borda, while also utilizing its strengths. In order to decide which passengers are more preferable in the seven-passenger coalition, like before, the percentage of soft constraints satisfied between them is calculated. The key difference is that it is added to the score of the agent. Put differently, there is no preference ranking. For every agent, we sum the scores that all the others gave them. Finally, we sort the final scores and the 4 ones with the highest, will join the coalition. This mechanism improved the preference satisfaction metrics, but still we could not be sure that it produces the best combinations. To clarify this, let us consider again seven passengers/candidates for the four seats. We assume that p_4, p_5, p_6 have the three highest scores and therefore claim three seats in the vehicle. Now, let us assume that p_1 and p_7 are the candidates for the fourth seat. Regarding p_7's score, we have: p_4 has $\frac{2}{3}$ compatibility with p_7's preferences, p_5 $\frac{1}{3}$ and p_6 $\frac{2}{3}$. Therefore, p_7's score is: $\frac{2+1+2}{3} = \frac{5}{3}$. For p_1's score on the other hand, we have: p_1 has no compatible preferences with the three passengers who have already claimed the three seats in the vehicle. p_2 has $\frac{3}{3}$ compatibility with p_1 and p_3 also $\frac{3}{3}$. Thus, p_1's score is: $\frac{3+3}{3} = \frac{6}{3}$. This means that p_1 claims the fourth seat as they have a higher score, even though they are incompatible with the rest of the passengers in the vehicle and p_7 was left out despite being the one matching best. We see now how the highest score does not necessarily guarantee the best coalitional preference satisfaction.

Therefore, in our third attempt, we explore all possible quartet combinations from the pool of 7 passengers within the temporary coalition. For each quartet, we calculate the overall preference satisfaction. To do so, initially, for each agent in the quartet, we calculate their preference rate (PR) in relation to their soft constraints satisfied and the other three passengers. Then, the four individual PRs are summed to form the

coalitional PR (CPR). Once CPR is computed for every quartet, we select the one with the highest CPR to establish the final coalition. In cases where multiple quartets yield the same highest CPR, we choose the one that best aligns with the driver's preferences by evaluating passengers' values in the hyperedge. This decision also aids in optimizing other metrics, such as extra distance. Our conducted experiments confirm that this approach effectively enhances coalitional preference satisfaction and yields better results than the previous ones. Finally, all the agents participating in the final coalition are removed from the preference lists since they are no longer considered a feasible choice.

3.2.4 Putting the algorithmic components together into a complete coalition formation process

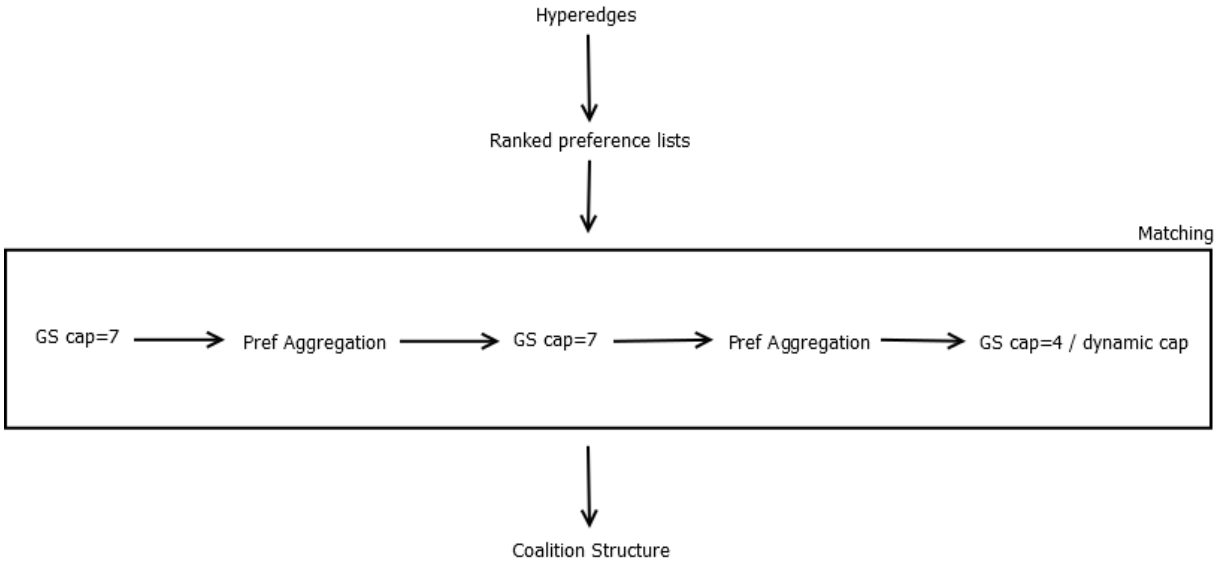


Figure 3.6: Complete coalition formation process.

After calculating passengers' preferences, it is time to explain how all the elements mentioned above are combined for the formation of the coalitions. First, the hyperedge values are used to create ranked preference lists. Then, the GS algorithm is run three times. In the first two iterations, the capacity of each coalition is set to seven, as mentioned earlier. After each of these iterations, cars that reach capacity four are added in the coalition structure, while preference aggregation for the passengers is applied to form the final coalitions in those with capacity higher. The reason for repeating this process twice is that, by initially setting the capacity of each car to seven, some passengers will be placed in a coalition, only to be removed later. Considering that the rejected agents have other options in their preference lists, in the next iteration, they can be placed in a

coalition. It is important to note that during these two iterations of GS, only car-filling coalitions are formed. All other agents placed in cars that do not reach capacity four or higher are not included in the coalition structure and therefore the drivers of those cars in the next iteration will begin with empty seats again. In the third and final iteration, the GS algorithm will run with a capacity of four, which is the actual capacity of the car from a passenger perspective, and all coalitions formed are added in the coalition structure. This way, the agents rejected in the second iteration, along with all the remaining ones, will have a final opportunity to form coalitions. The process is illustrated in Figure 3.6.

The decision to run GS with a capacity of seven twice was primarily based on empirical observations. After conducting several experiments, it became evident that the repetition did not result in the formation of coalitions that filled the entire car. Additionally, since passengers' preference aggregation is only applied when the car exceeds full capacity (i.e., more than four passengers), it was seldom utilized in most instances. Hence, it follows logically that the capacity should be adjusted to the actual car capacity for the final iteration, and therefore focus on improving the average size of the coalitions.

3.3 Route Extraction

Once the coalition structure (CS) has been established, the next step is to determine the route that each driver must take to accommodate the passengers in their vehicle. It turns out that as the number of passengers increases, this task becomes more complicated. As per the hard constraints defined in Section 3.1.1, for a route to be considered feasible, it must begin at the driver's starting point and finish at their destination. Furthermore, every agent's starting point precedes their destination. Keeping these in mind, we consider three cases of the final route.

- If the driver is alone, then their route is their original path.
- If there is one passenger, then from the constraints defined, the sequence of the stops followed is :
  $driver's\_start -> passenger's\_start -> passenger's\_finish -> driver's\_finish.$
- If there are two or more passengers, we need to identify the optimal sequence of stops the driver has to make, in order to minimize the extra distance and thus the overall cost.

Expanding on the third case, there are multiple permutations of stops that can be followed, even within the restrictions we established for a route to be declared feasible.

To illustrate this, consider the following example. Let C be a coalition with a driver and three passengers. We denote as s_i, d_i the start and destination of each passenger i. The route commences and concludes at the driver's starting point and destination respectively. From the constraints defined, the permutation $start \rightarrow s_3 \rightarrow s_1 \rightarrow d_1 \rightarrow s_4 \rightarrow s_2 \rightarrow d_4 \rightarrow d_2 \rightarrow d_3 \rightarrow destination$ is a feasible one, but so is $start \rightarrow s_2 \rightarrow s_4 \rightarrow s1 \rightarrow d_4 \rightarrow d_2 \rightarrow s_3 \rightarrow d_1 \rightarrow d_3 \rightarrow destination$. On the contrary, the permutation $start \rightarrow s_2 \rightarrow s_3 \rightarrow d_4 \rightarrow s_1 \rightarrow d_2 \rightarrow d_3 \rightarrow s_4 \rightarrow d_1 \rightarrow destination$ is not feasible because #4 passenger's destination stop proceeds their start. To grasp the scale of possible permutations, we present the tree of Figure 3.7. In each level of the tree, we display only the feasible options for the next stop.

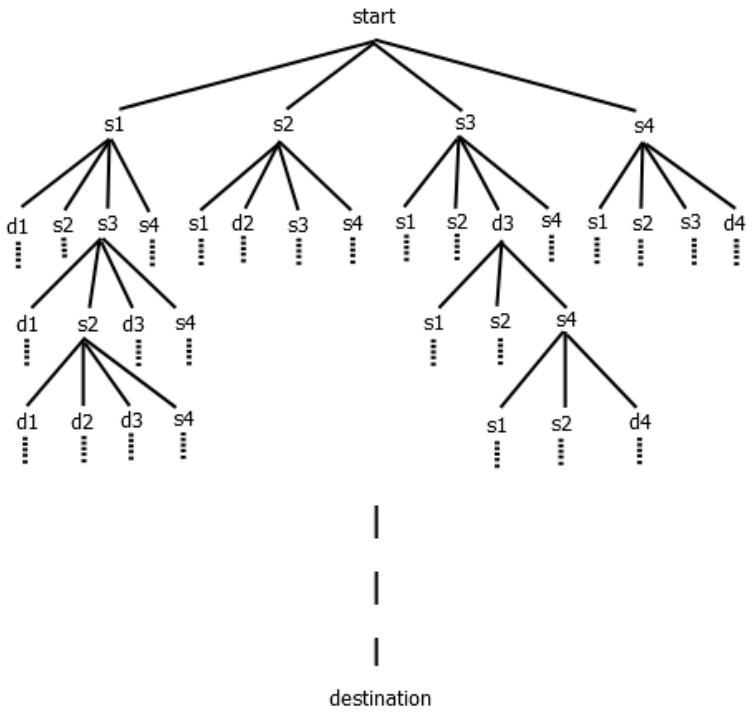


Figure 3.7: Feasible permutations of stops.

To determine the optimal permutation of stops, we apply the CFRE[1] algorithm [1] discussed in Section 2.9.3. As previously explained, CFRE employs search trees, where potential routes are investigated through the application of Depth First Search. At each stop, the cost of traveling from the previous one is accumulated until the driver's destination is reached and the final cost of the route can be calculated. For instance, for the first permutation of the above example, the cost is calculated as follows:

[1]In our approach, CFRE has undergone slight modifications to align with our work. While the functionality remains unchanged, we have modified the structure by breaking down the algorithm into functions, enhancing its versatility. This modification enables us to utilize specific components in other sections of our work.

$total\ cost = c(start \rightarrow s_3) + c(s_3 \rightarrow s_1) + c(s_1 \rightarrow d_1) + c(d_1 \rightarrow s_4) + c(s_4 \rightarrow s_2) + c(s_2 \rightarrow d_4) + c(d_4 \rightarrow d_2) + c(d_2 \rightarrow d_3) + c(d_3 \rightarrow destination)$.
Formally, the total cost is represented by Equation 3.7.

$$total\ cost = \sum_{\substack{i \neq j \\ i \rightarrow j\ \ feasible}} c(stop_i \rightarrow stop_j) \tag{3.7}$$

Each cost is calculated using Euclidean distance measured in kilometres. This approach is faster than computing the exact distance between two stops, which would require the application of a pathfinding algorithm to measure it. Given the numerous transitions that need to be calculated for each permutation and the amount of feasible permutations for each route, this method proves to be significantly more efficient. CFRE's pseudocode can be seen in Algorithm 4.

Algorithm 4: CFRE(Coalition Structure)

```
1  for C in Coalition Structure do
2      if |C| == 1 then
3          stops{driver's origin, driver's destination};
4      else if |C| == 2 then
5          stops{driver's origin, s_1, d_1, driver's destination};
6      else
7          State root = NEW_STATE(driver's origin, null);
8          costTable = costs of feasible transitions;
9          Q.insert(root);
10         minCost = ∞;
11         while !Q.empty() do
12             N = Q.pop();
13             if !N.blocked && N.cost ≤ minCost then
14                 for p in N.options do
15                     State n = NEW_STATE(p, N);
16                     Q.insert(n);
17         stops = optimalSolution.stops;
18     return stops;
```

Once we have the final sequence of stops, Dijkstra's algorithm is employed to determine the actual route. The algorithm is applied in each transition, connecting the nodes

between the stops to form "sub-paths", which are then linked together to create the final path. Since it is a shortest-path algorithm, the result is guaranteed to be the fastest one.

Algorithm 5: NEW_STATE(p, parentState)

```
1  New State s;
2  s.parent = parentState;
3  s.options = options of p according to costTable;
4  s.cost = parentState.cost + transition cost to p (euclidean distance);
5  if destination is reached then
6      s.blocked = true;
7      if s.cost ≤ minCost then
8          minCost = s.cost;
9          optimalSolution = N;
10 return s;
```

3.4 Payment Allocation

Payment allocation is arguably a crucial aspect of any real-world ridesharing approach. It entails the computation of a payoff vector x where $x[i]$ represents agent a_i's payoff. In this work, the payment vectors are *efficient*, meaning that the coalitional value is distributed entirely among its members. It is essential to ensure that participants have confidence in the fairness of the payments they make, as this significantly influences their willingness to participate in the process. To achieve this, we employ the kernel stability concept. As elucidated in Section 2.6, kernel-stable payoffs are deemed fair, as they guarantee that no agent can plausibly claim a share of their partners' payoff. Thus, payoffs should be divided among agents according to their bargaining power. Moreover, the kernel of a coalition is ensured to be non-empty, therefore this solution concept is consistently able to produce a kernel-stable payment allocation.

Before we explain how the kernel is computed, we must first define the final cost of the coalition. From this cost, the driver's cut will be extracted and compared to the cost they would pay, had they traveled alone. The driver's cost decrease is then used as a metric in the evaluation of our work. To calculate the final cost, the distance in km of the route found in the previous section is multiplied with the cost per km (fuel cost) as shown in Equation 3.8.

$$final\_cost = distance(km) \cdot cost/(km) \tag{3.8}$$

3.4.1 The kernel

For an outcome to be in the kernel, the payoffs should be distributed in such a way that no agent can outweigh the other members of the coalition, i.e., be able to claim a share of their payoff. To achieve this, Sterns [36] presented a payoff transfer scheme that converges to a kernel-stable payoff vector. In this scheme, payoffs are transferred from agents with less bargaining power to those with more, until the latter cannot make further claims. This implementation, however, may need infinite iterations to complete. To address this limitation, Klusch and Shehory implemented the ϵ-kernel [37], an approximation to the kernel where its payoffs do not differ more than ϵ from those in the kernel. This concept was further developed by Shehory and Kraus [38], representing the state of the art at the time. Subsequently, Bistaffa et al. [2] introduced the PK (Payments in the Kernel) algorithm, which improved upon Shehory and Kraus' algorithm with respect to calculating the surplus matrix and avoiding redundancy by not considering the same coalitions more than once while forming it. Furthermore, PK utilizes the graph's structure to take into account only feasible coalitions, confining the computation time further.

In this thesis, we use the PK algorithm to split the travel cost. Our application of the algorithm is slightly different than Bistaffa et al.'s. In their work, they use a social network from which feasible coalitions are considered. They developed a parallel version where they sample over the feasible coalitions from which the surplus matrix (SM) is updated. In our case, the feasible coalitions considered in the SM formation, are created "randomly"[2] from the pool of agents in the CS. More specifically, given a coalition in CS, in order to calculate the SM, for every agent in it we create 10 random coalitions using the hyperedges that each agent belongs to. The constraint is that the agents selected in the random coalitions must belong in the CS, i.e., in one of the final coalitions formed by the GS algorithm. The change in the calculation of SM can be seen in line 2 of Algorithm 7 of the modified COMPUTE_MATRIX. The random coalitions are then used to update the SM by calculating the excess of each one. This can be seen in the UpdateMax routine of Algorithm 8, where the excess is used to update the appropriate elements of the surplus matrix. The excess was defined in Equation 2.1, with the cost $v(C)$ of the coalition being the final cost of Equation 3.8. The sum of x_i is computed using the x allocation that each agent in the random coalition got in the CS. This is the reason we specified that the agents who participate in the random coalition formation, must also belong in the CS. The original PK algorithm is detailed in Algorithm 6.

To make the process more clear, we present an example. We suppose that the coalition

[2]The agents participating in these coalitions are not completely random. They must belong in the same hyperedges to be able to form a coalition in the first place, and additionally, they must belong in the coalition structure to be able to calculate the SM from their payoff allocations.

formation has finished and CS has been formed with 140 final coalitions. We consider the coalitions

- $C_1 = \{a_1, a_2, a_3, a_4, a_5\}$, with cost $u(C_1) = 2.5$ and thus after splitting the cost as $x[i] = \frac{u(C_1)}{|C_1|} : x = \{0.5, 0.5, 0.5, 0.5, 0.5\}$
- $C_2 = \{a_6, a_7, a_8, a_9, a_{10}\}$, $u(C_2) = 2$ and $x = \{0.4, 0.4, 0.4, 0.4, 0.4\}$
- $C3 = \{a_{11}, a_{12}, a_{13}, a_{14}, a_{15}\}$, $u(C_3) = 1.5$ and $x = \{0.3, 0.3, 0.3, 0.3, 0.3\}$

We assume that a_1 is the driver of C_1, and our objective is to create 10 random coalitions involving a_1 to compute the surplus matrix of C_1. We search for passengers belonging to a_1's hyperedge, indicating they can form a coalition with a_1, while also being part of the CS, signifying their participation in some other coalition among the 140 available. Suppose we discover that a_7 and a_{13} meet these criteria. Then we can form the random coalition S, where $S = \{a_1, a_7, a_{13}\}$, $u(S) = 1.7$. We calculate the excess using the payoff allocation that the agents received for the *final* coalitions (C_1, C_2, C_3) as: $e(S, X) = u(S) - \sum_{i \in S} xi = 1.7 - (0.5 + 0.4 + 0.3) = 0.5$. Subsequently, this excess is used to update $s_{12}, s_{13}, s_{14}, s_{15}$ of C_1's surplus matrix as the random coalition S contains a_1, but does not contain a_2, a_3, a_4, a_5.

Algorithm 6: PK(CS, ϵ) [2]

for ***all*** $S \in CS$ **do**
  for ***all*** $a_i \in S$ **do**
    $x_i \leftarrow \frac{v(S)}{|S|}$; {Equally split coalitional value}
repeat
  {Compute surplus matrix}
  s $\leftarrow$ COMPUTE_MATRIX(CS, x);
  {a_{i*} and a_{j*} have the maximum surplus difference δ }
  $\delta \leftarrow max_{(a_i,a_j) \in A^2}(s_{ij} - s_{ji})$;
  $(a_{i*}, a_{j*}) \leftarrow argmax_{(a_i,a_j) \in A^2}(s_{ij} - s_{ji})$;
  if $x[j*] - v(\{a_{j*}\}) \leq \frac{\delta}{2}$ **then**
    $d \leftarrow x[j*] - v(\{a_{j*}\})$;
  else
    $d \leftarrow \frac{\delta}{2}$;
  {Transfer payment from a_{j*} to a_{i*} }
  $x[j*] \leftarrow x[j*] - d$;
  $x[i*] \leftarrow x[i*] + d$;
until $\frac{\delta}{v(CS)} \leq \epsilon$;

Algorithm 7: COMPUTE_MATRIX(CS, x)

```
1 s ← −∞; {Initialize the entire matrix with −∞}
2 for all random coalitions (RCS) produced for S do        ▷ modified here
3 |  s ← UPDATE_MAX(S, RCS, s, x);
4 return s
```

Algorithm 8: UPDATE_MAX(S, RCS, s, x)

```
1 e_S ← e(S, x); {Compute the excess of coalition S}
2 for all a_i ∈ S do
3 |  S' ← the coalition in RCS that contains a_i;
4 |  for all a_j ∈ S' − S do
5 |  |  {s_ij is updated with the maximum between its old value and the excess of
  |  |   coalition S}
6 |  |  s_ij ← max(s_ij, e_S);
7 return s
```

4

Experiments & Results

In this section, a series of experiments is conducted, in order to evaluate the efficiency of our work. Through simulations for the city of Chania, we test both versions of the Gale-Shapley algorithm we presented in the previous chapter, the kernel payment allocation, as well as the three cases of the service area. Our results are compared with those of [1].

4.1 Simulations

For the evaluation process, we utilised the simulation method described in Section 2.9.3 of previous work [1]. More specifically, through OpenStreetMap, a connected graph is created for the city of Chania, where vertices are possible locations in the city and edges represent roads connecting them. The city graph contains 3278 vertices and for every one, we know their coordinates of longitude and latitude. Depending on the number of agents participating in each simulation (here 800, 1600, 2500), random vertices are selected to represent their starting and finishing point locations. Furthermore, for each agent, their soft constraints are produced randomly. The percentage of drivers is specified by the user of the program and some of the agents are also randomly assigned as drivers. For instance, if the simulation is executed for 800 agents with 35% of them being drivers, then 280 agents are randomly selected to be the drivers.

As mentioned in Section 3.1.2.2, the results can vary depending on the allocation of

coefficients, emphasizing different aspects. In this thesis, we used the allocation $v_D = dir \cdot 0.8 + dist \cdot 0.1 + SCS \cdot 0.1$ to evaluate our work. Since our focus is on preference satisfaction when forming coalitions, we didn't want to highlight this aspect further by multiplying SCS with a higher coefficient. Furthermore, the distance factor is already confined by the service area of the driver as we explained in Section 3.1.2.1. This makes direction the factor that should be multiplied by the highest value.

The experiments are conducted in the city of Chania with 35% of the total number of agents being drivers. We examine the performance of the two versions of the Gale-Shapley algorithm and the kernel payment allocation for three agent group sizes (800, 1600, 2500). Additionally, every version of the GS algorithm is applied to the three cases of the service area. The first case (one-circle service area) is compared with previous work [1], while the other two cases (circle and ellipse cuts) are compared to GS implementations since they are unique to our approach.[1] To measure the effectiveness of our work, we used the following metrics:

1. **AH**: The size of the hyperedges lists in average.

2. **Coal #**: Average number of final coalitions formed. Singletons are not included.

3. **CPR**: The preference satisfaction rate among the agents of the coalition. To calculate this, first, for each agent's preference, we find how many members of the coalition satisfy it, e.g., $pref_i = \frac{k}{n}$, which means that k out of the n members of the coalition satisfy this preference. Then, for each agent j we calculate their preference rate as

$$pr_j = \frac{\sum_{i=1}^{pref\_num} pref_i}{pref\_num} \tag{4.1}$$

   where $pref\_num$, the number of preferences expressed for each agent. For instance, if they have a preference for age and employment, then $pref\_num = 2$. Lastly, the coalitional preference rate is calculated as the average of the atomic pr

$$CPR = \frac{\sum_{j=1}^{|C|} pr_j}{|C|} \tag{4.2}$$

4. **AED**: The average extra distance. As the name suggests, it is the percentage of increase in the distance that drivers have to cover in order to accommodate the

[1] The main difference between the one-circle service area and the "cuts" cases is that, in the latter, the service area is formed alongside the driver's path to minimize additional distance when accommodating the passengers.

passengers. It is calculated as

$$AED = \frac{final - initial}{initial} \cdot 100(\%) \tag{4.3}$$

5. **CR**: The percentage of the driver's cost reduction after the final cost is shared among the members of the coalition. Formally

$$CR = \frac{init\_cost - final\_cost}{init\_cost} \cdot 100(\%) \tag{4.4}$$

6. **PPC**: The average number of passengers per car.

4.2 Gale-Shapley & Preference Aggregation

As previously mentioned, for the coalition formation, three iterations of the GS algorithm will be used. The first two with maximum capacity of seven passengers, on which preference aggregation among passengers is applied for the selection of the final four passengers in each car. In the third iteration, GS will be used with capacity four and thus no preference aggregation among passengers (since this only happens to keep the final four when there are more candidates as we explained in Section 3.2.3). We tested this version of GS with both the kernel payment allocation and the one from [1] (see Equation 2.4) to compare the results. We evaluate the performance of this approach for the three cases of the service area.

4.2.1 One-circle service area

# of agents → metrics ↓	800	1600	2500
AH	68	136	214
Coal #	148	300	471
CPR	64%	68.25%	71.16%
AED	109.25%	103.14%	99.75%
PPC	2.91	2.95	2.96

Table 4.1: Results for the first case of GS.

Kernel CR	68.37%	70.53%	74.84%
Previous work [1] CR	81.9%	81.68%	81.63%

Table 4.2: Payment allocation methods comparison for the first case of GS.

In Table 4.1 we notice that as the number of agents is increased, the percentage of extra distance (AED) is decreased. This happens because, with more agents, drivers have more alternatives to match with, thus agents who are more preferable route-wise, are selected. The same can be said for the Coalition Preference Rate (CPR), which also increased. The Passengers Per Coalition (PPC) are close to three in all cases, meaning that the matching is successful. Observing Table 4.2, we see that for the kernel case, as the agent group grows, so does the cost reduction in the driver's share. This is expected since with more options, agents have a higher chance to match with their more preferable options, resulting in an overall less incentive to deviate from their coalitions. As a result, in the absence of their original driver match, they end up paying a higher price. Therefore, if they wish to maintain a coalition with that driver, they must share part of the driver's cost. Contrary to the findings of the kernel, we observe that the cost reduction of the previous work gets smaller as the size of the agent group grows. The reason this happens is because AED is reduced. From Equation 2.4, we have the factor

$$1 - \frac{initKM}{final_{KM}} = \frac{final_{KM} - init_{KM}}{final_{KM}} = \frac{extra_{KM}}{final_{KM}}$$

Thus, when the extra distance is less, we also have less cost reduction. Of course, this method offers greater overall cost decrease for the driver, but we argue that our method is more fair for all the participants.

4.2.2 Circle-cuts

We now change the shape of the service area to smaller circle-cuts alongside the driver's path. Since the driver's average path length is approximately 2km, we examine cuts in the range of 0.5km - 2km. If the path has a length smaller than the cut's length, the service area is calculated as one big circle like in the first case. For the cost reduction calculation, we used the previous work method, as it is faster and we can still compare the results.

Cuts		800 agents	1600 agents	2500 agents
	AH	24	46	75
	Coal#	145	299	476

Every 0.5 km	CPR	59.95%	64.19%	66.96%
	AED	79.89%	76.36%	71.33%
	CR [1]	77.57%	77.76%	77.68%
	PPC	2.57	2.65	2.69
Every 1 km	AH	53	106	160
	Coal#	148	302	476
	CPR	62.6%	67.22%	69.81%
	AED	95.48%	93.71%	89.87%
	CR [1]	80.71%	80.69%	80.47%
	PPC	2.81	2.85	2.87
Every 1.5 km	AH	66	125	204
	Coal#	149	296	477
	CPR	64.05%	68.12%	70.84%
	AED	104.39%	103.19%	98.33%
	CR [1]	81.32%	81.77%	81.28%
	PPC	2.86	2.95	2.93
Every 2 km	AH	63	132	211
	Coal#	147	299	473
	CPR	63.94%	68.66%	71.3%
	AED	110.93%	104.31%	101.12%
	CR [1]	81.72%	81.53%	81.52%
	PPC	2.88	2.93	2.95

Table 4.3: Results for different circle service area radii - GS

Comparing the results with those of Table 4.1, it is clear that for smaller cut lengths, the size of the service area has indeed been confined. This is evident in the size of the hyperedges which is significantly smaller. Consequently, a large number of passengers was left out of the coalition formation process, resulting in a reduction of the number of passengers per car. This has also affected AED which in this case has been reduced significantly, as expected. While the length of the cuts rises, so do the metrics, to the point that in the last two cases, the results are similar to those of the first case of the service area.

4.2.3 Ellipse-cuts

In this case, the service area changes to smaller ellipses alongside the path, to cover more area when the cuts have smaller length. The length of the minor radius depends

on the length of the cuts, while the major is calculated as $r_{major} = r_{minor} + \frac{r_{minor}}{2}$. This formula was decided upon, after experimentation. We examine cut lengths in the range of 0.5 km - 1.5 km. After that the length of the major radius is augmented greatly, affecting the extra distance factor excessively.

Cuts		800 agents	1600 agents	2500 agents
Every 0.5 km	AH	43	88	136
	Coal#	152	311	484
	CPR	62.14%	66.6%	70.41%
	AED	95.19%	90.56%	87.18%
	CR [1]	80.13%	80.23%	80.34%
	PPC	2.72	2.77	2.82
Every 0.75 km	AH	66	137	213
	Coal#	155	312	493
	CPR	62.94%	68.32%	71.03%
	AED	109.56%	100.51%	101.65%
	CR [1]	81.8%	81.47%	81.71%
	PPC	2.81	2.85	2.88
Every 1 km	AH	82	159	255
	Coal#	159	310	487
	CPR	64.28%	68.83%	71.84%
	AED	114.36%	112.32%	110.26%
	CR [1]	82.36%	82.58%	82.84%
	PPC	2.83	2.92	2.98
Every 1.5 km	AH	86	178	282
	Coal#	154	313	489
	CPR	65.67%	69.71%	72.2%
	AED	128.35%	118.84%	117.25%
	CR [1]	83.1%	83%	83.1%
	PPC	2.9	2.94	2.97

Table 4.4: Results for different ellipse service area radii - GS

Comparing with Table 4.3, we notice that the average size of the hyperedges has improved, as well as the number of coalitions. Even for small length cuts, the number of passengers is larger than the circle cuts case, and CPR is comparable to that of Table 4.1. While the cuts' length is increased, the metrics get significantly better, however, this increase also affects the AED factor. This is especially clear in the 1.5km case. Overall, this

method achieves high preference satisfaction even among a small group of agents. Specifically for the 0.5km radius, if a smaller (but still satisfying) PPC number is acceptable, drivers can accommodate passengers while covering less additional distance, reducing the time and cost of the trip.

4.3 Gale-Shapley & Preference Aggregation: "dynamic capacity" version

We now test the efficiency of the dynamic capacity version. As explained above, this version of the GS algorithm utilises dynamic capacity to "fill" empty seats in the cars with passengers. For the three cases of the service area the results are the following:

4.3.1 One circle service area

# of agents → metrics ↓	800	1600	2500
AH	67	135	212
Coal #	179	362	566
CPR	59.77%	63%	66.14%
AED	120.37%	113.42%	109.11%
PPC	3.1	3.17	3.2

Table 4.5: Results for dynamic GS.

Kernel CR	69.27%	70.78%	72.27%
Previous work [1] CR	83.83%	84.03%	83.92%

Table 4.6: Payment allocation methods comparison for the dynamic version GS.

It is clear that the dynamic version has indeed increased the PPC metric with it being above three for all sizes of the agent groups. We also observe that the CRP has decreased and AED has risen almost 10% compared to that of Table 4.1. Since this version fills empty seats in vehicles, it is expected that agents' preferences might not fit as well as before. The same applies to AED since now the driver has to cover a greater extra distance to pick up the passengers, as they might not be the perfect match spatially. The number of coalitions is also higher, meaning that there are less singletons. As far as the kernel CR

of Table 4.6 is concerned, we notice that in the 800 and 1600 agents cases it has slightly increased compared to Table 4.2, since more passengers share the cost, however, in the last case it has decreased. We think that is because in this case, the matching is not ideal, meaning that some agents could be better off in some other coalitions and therefore, they might have a motive to deviate from the coalition. In this case, their bargaining power is greater than the driver's and thus they can credibly claim part of the driver's payoff. As a result, the driver's share of the cost is augmented, but not significantly. Contrary, in the case of the previous work CR, we observe that the driver's cost has been reduced further. As we explain above, that is due to the extra distance increase since this method is heavily dependent on it.

4.3.2 Circle - cuts

Applying DGS in the service area that is now comprised of smaller circles alongside the driver's path, we have the following results:

Cuts		800 agents	1600 agents	2500 agents
Every 0.5 km	AH	24	49	74
	Coal#	182	374	523
	CPR	55.22%	59.5%	62.62%
	AED	93.37%	76.36%	84.46%
	CR [1]	81.23%	86.27%	81.79%
	PPC	2.88	2.98	3.06
Every 1 km	AH	51	105	165
	Coal#	178	366	540
	CPR	58.66%	63.03%	65.7%
	AED	115.9%	108.66%	103.07%
	CR [1]	84.29%	83.9%	83.69%
	PPC	3.17	3.18	3.19
Every 1.5 km	AH	63	132	206
	Coal#	179	357	564
	CPR	59.32%	63.26%	66.8%
	AED	117.4%	113.3%	110.42%
	CR [1]	83.66%	84.39%	83.95%
	PPC	3.11	3.21	3.19
	AH	67	139	212
	Coal#	179	322	561

Every 2 km	CPR	58.93%	64.29%	66.5%
	AED	121.03%	112.88%	111.77%
	CR [1]	84.2%	84.14%	84.24%
	PPC	3.13	3.19	3.22

Table 4.7: Results for different circle service area radii - DGS

As in the one-circle case of Table 4.5, the number of coalitions is greater than the first GS version of Table 4.1. The CPR is still lower, but the PPC metric is high even for the 0.5km case. As the length of the radius is increased, so does CPR but also AED.

4.3.3 Ellipse - cuts

After changing the service area into smaller ellipses and applying DGS, the results are the following:

Cuts		800 agents	1600 agents	2500 agents
Every 0.5 km	AH	46	89	147
	Coal#	187	384	593
	CPR	57.64%	66.44%	65.19%
	AED	109.84%	103.17%	98.85%
	CR [1]	84.01%	83.46%	83.57%
	PPC	3.07	3.08	3.13
Every 0.75 km	AH	67	137	215
	Coal#	185	376	586
	CPR	58.98%	63.62%	66.3%
	AED	126.13%	120.98%	116.42%
	CR [1]	85.14%	85.37%	84.93%
	PPC	3.14	3.22	3.21
Every 1 km	AH	76	158	255
	Coal#	183	371	586
	CPR	59.29%	64.46%	67.18%
	AED	140.87%	130.77%	127.07%
	CR [1]	86.15%	85.47%	85.57%
	PPC	3.22	3.21	3.25
Every 1.5 km	AH	87	179	281
	Coal#	187	339	581
	CPR	59.91%	65.34%	67.55%

	AED	139.99%	132.94%	132.67%
	CR [1]	85.73%	85.59%	85.37%
	PPC	3.18	3.21	3.26

Table 4.8: Results for different ellipse service area radii - DGS

We can see that in this case, the PPC has increased further along with the driver's CR and CPR, but unfortunately so has AED. For the first case of 0.5km radius we observe satisfactory results in terms of AED being at an acceptable level and PPC being over three for all sizes of the agent-groups. After that, AED is affected greatly.

4.4 Comparison with Previous Work

We now compare our approach with the previous work [1]. In that implementation, the PASR-CF algorithm is used for the matching of the agents, and the Equation 2.4 for the driver's cost. Since the circle and ellipse cuts cases of the service area are unique to our work, we compare the results only with the first case of the service area; that is the one-circle case of Section 4.2.1. The results are:

# of agents → metrics ↓	800	1600	2500
AH	69	136	212
Coal #	146	294	464
CPR	60.8%	65.33%	68.66%
AED	102.14%	98.57%	97.13%
PPC	2.89	2.93	2.97

Table 4.9: Results of the previous work [1]

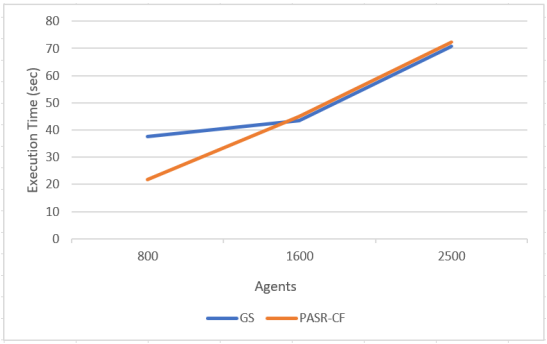
Kernel CR	72.43%	70.16%	72.54%
Previous work [1] CR	81.63%	81.58%	81.65%

Table 4.10: Payment allocation methods comparison for previous work [1]

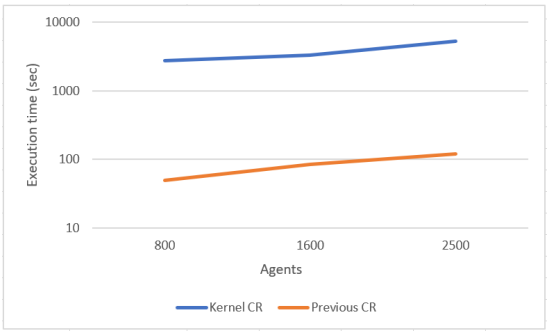
It is evident when comparing with Table 4.1 that GS provides higher preference satisfaction. The PPC metrics and the number of coalitions are comparable, while the AED in Table 4.9 is the lowest for every agent group size compared to our approach. In the kernel CR of Table 4.10 where we applied our kernel implementation to the PASR-CF

algorithm, we observe that in the case of 800 agents, it has a higher value compared to the 800 case of Table 4.2. This is likely because PASR-CF spatially matches the agents well, which is also evident in the AED results. For larger group sizes, our approach appears to perform slightly better, as our kernel CR is larger than the corresponding one of [1]. More specifically, for 2500 agents our approach achieves a 74.84% decrease compared to the 72.54% of [1]. Finally, for the case of the previous CR method, the results are similar for both of our implementations.

We will now compare the execution time of each algorithm, both for the coalition formation and the route extraction and payment allocation (they are computed in parallel). Because the same algorithm is used for the route extraction, we can still compare the cost-sharing schemes. The charts of the execution time for both, can be seen in Figure 4.1.



(a) Execution time comparison for coalition formation algorithms



(b) Execution time comparison for payment allocation approaches

Figure 4.1: Coalition formation and payment allocation execution times

As we can observe, concerning the coalition formation algorithms, the results are quite similar. For smaller agent groups, PASR-CF exhibits better execution time, but as the group sizes increase, our approach appears to perform slightly better. In contrast, the kernel payment scheme significantly lags behind the previous cost-sharing method in speed. While the former fluctuates around 100 seconds, the kernel demands approximately 5000, marking a difference between minutes and an hour. As much as the kernel solution concept is fair, it lacks in computation speed. Consequently, we realize that our approach of the kernel transfer scheme is not the ideal choice for dynamic ridesharing implementations where response time must be in seconds.

4.5 Application of each approach

After evaluating the performance of each implementation and comparing the results we make some final remarks. Regarding the main version of the GS algorithm (Section

4.2.1), we can say that it is a well balanced approach. It achieves successful matching of the agents, as it keeps both CPR and PPC high across the different agent group sizes it was tested on. Thus, it can be applied to every setting and produce satisfactory results.

The dynamic version of the GS algorithm (Section 4.3.1) aims to increase the number of coalitions and passengers per car (PPC), maintaining it above three in all cases, as indicated by the experiments. To achieve this, preference satisfaction is compromised, evidenced by a decrease compared to the aforementioned version of GS, while AED has shown an increase. For these reasons, this approach is suitable for scenarios where a high number of passengers per car is the top priority.

The PASR-CF algorithm by [1] is another stable approach, yielding satisfactory results across all metrics in agent matching. Unlike our primary implementation of the Gale-Shapley algorithm, PASR-CF prioritizes spatial matching of agents. As a consequence, it exhibits a lower percentage of preference satisfaction but also a reduced percentage of extra distance.

Regarding the circle and ellipse-cuts implementations of the service area, we could say that they are more versatile, adaptable to the needs of the problem when combined with each version of the GS algorithm. Originally created to confine extra distance, the circle-cuts case achieves this for smaller radii. However, as the service area is confined and agents are left out of the coalition process, CPR is compromised. When combined with the first case of the GS algorithm, PPC is also decreased. As the radius is increased, the results get similar to that of the one-circle case. On the contrary, when combined with the dynamic version of GS, the PPC are maintained high across all group sizes and all radii. Moreover, if the extra distance resulting from the increased radius is not a concern, a higher preference rate can also be achieved.

The ellipse-cuts, an extension of the circle-cuts, confine AED but not to the extent of the circle-cuts. Therefore, the overall metrics are better, but AED is also higher. Combined with the first case of GS, as the radius is increased, the results can get better than those of the one-circle service area. On the other hand, when combined with the dynamic version of GS, the PPC is also maintained high in this case as well. For small radii, the results of this implementation are satisfactory, since CPR and AED are at an acceptable rate. However, as the radius is increased, even though the CPR is higher in this case, the AED is increased excessively. This makes larger radii suitable for settings where additional distance is tolerable when achieving high PPC and CPR.

Furthermore, we believe that the “cuts” approach to the service area would be more appropriate for longer path lengths. In scenarios where the average path length is, for instance, 10km, even in our cases with larger radii presented here, the service area would still be significantly restricted. Conversely, the one-circle service area would not be the

optimal choice, as the driver would need to cover an area of 20km diameter.

5

Conclusion & Future Work

In this thesis, we present our approach to the ridesharing problem, rooted in game-theoretic concepts. Our primary goal is to establish stable coalitions where agents have no incentive to deviate. To this end, we produce core-stable matches and kernel-stable payment allocations. To match drivers with passengers, we utilize hypergraphs to find candidates who move in the same area as the driver. This will confine the extra distance, a desirable feature considering that our work caters to regular individuals seeking to share trip costs, as opposed to professional taxi drivers. We employ the Gale-Shapley algorithm to form the driver-passenger matches from the pool of candidates, ensuring that the outcome resides in the core.

Beyond spatial matching, preference satisfaction and preference aggregation is a crucial aspect of our work. Agents have the opportunity to express preferences regarding their partners' characteristics, and they are taken into account when matching them. These preferences also concern the relationships between passengers in the same vehicle. A higher preference satisfaction rate among the members of a coalition, will naturally result to an overall more pleasant trip. Our experimental results confirm that our approach meets this objective.

Then, to find the optimal sequence of stops the driver has to make to accommodate the passengers, a Branch and Bound algorithm is used, along with Dijkstra's algorithm to find the shortest path. Finally, the kernel solution concept is applied for the payment

allocation.

To evaluate the performance of our work, we conduct a series of simulation experiments in the city of Chania and compare the results with those of [1]. Our results demonstrate that our implementation produces coalitions with high preference satisfaction rate and efficient passenger distribution. Furthermore, ridesharing is decreasing the cost per agent, with drivers seeing a significant reduction. However, even though the kernel solution concept achieves fairness, at least our implementation of the kernel transfer scheme was shown to be slow and computationally demanding.

Moving ahead, future work directions could explore ways to enhance the performance of the kernel solution concept, making it more feasible for real-time applications. A parallel version of the PK algorithm, suggested by Bistaffa et al. [2], could be utilized, along with the D-slyce algorithm for sampling over the feasible coalitions. Additionally, learning could be utilized for the identification of the optimal coefficient allocation for the Equation 3.4, in order to get the best combination of spatial and preferential matching between drivers and passengers. Meeting points are another interesting topic in ridesharing, worth exploring. Meeting points could allow passengers who are theoretically outside of the driver's service area to participate, provided they are willing to walk a short distance to reach the meeting point. This way, more agents could participate in the process. Furthermore, meeting points could be utilized to get agents to gather in one place if they are close, minimizing the stops, or they could be used to get agents to meet the driver on main roads. As a result, delays caused by traffic on small roads and long detours to reach the passengers could be avoided. Furthermore, GIS (Geographic Information Systems) originating information in real-time traffic congestion could be incorporated to form the service area and optimize the final route. With this data, crowded areas could be avoided, reducing delays. Finally, the implementation of a real-life ridesharing platform for TUC, would address the transportation needs of students and faculty to and from the university.

Bibliography

[1] E. Pagkalos, "Preferences aware social ridesharing," 2021, Senior Undergraduate Diploma Thesis (MEng), School of Electrical and Computer Engineering, Technical University of Crete. [Online]. Available: https://doi.org/10.26233/heallink.tuc.90586

[2] F. Bistaffa, A. Farinelli, G. Chalkiadakis, and S. D. Ramchurn, "A cooperative game-theoretic approach to the social ridesharing problem," *Artificial Intelligence*, vol. 246, pp. 86–117, 2017.

[3] J. Alonso-Mora, S. Samaranayake, A. Wallar, E. Frazzoli, and D. Rus, "On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment," *Proceedings of the National Academy of Sciences*, vol. 114, no. 3, pp. 462–467, 2017. [Online]. Available: https://www.pnas.org/doi/abs/10.1073/pnas.1611675114

[4] R. J. Wilson, *Introduction to graph theory*. Pearson Education India, 1979.

[5] N. Deo, *Graph theory with applications to engineering and computer science*. Courier Dover Publications, 2017.

[6] S. J. Russell and P. Norvig, *Artificial intelligence a modern approach*. London, 2010.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2022.

[8] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968. [Online]. Available: https://doi.org/10.1109/tssc.1968.300136

[9] E. W. Dijkstra *et al.*, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[10] C. Berge, *Hypergraphs: combinatorics of finite sets*. Elsevier, 1984, vol. 45.

[11] P. Valdivia, P. Buono, C. Plaisant, N. Dufournaud, and J.-D. Fekete, "Analyzing dynamic hypergraphs with parallel aggregated ordered hypergraph visualization," *IEEE transactions on visualization and computer graphics*, vol. 27, no. 1, pp. 1–13, 2019.

[12] D. Zhou, J. Huang, and B. Schölkopf, "Learning with hypergraphs: Clustering, classification, and embedding," *Advances in neural information processing systems*, vol. 19, 2006.

[13] M. Wooldridge, *An introduction to multiagent systems*. John wiley & sons, 2009.

[14] G. Chalkiadakis, E. Elkind, and M. Wooldridge, *Computational aspects of cooperative game theory*. Springer Nature, 2022.

[15] J. Rothe, Ed., *Economics and Computation, An Introduction to Algorithmic Game Theory, Computational Social Choice, and Fair Division*, ser. Springer texts in business and economics. Springer, 2016. [Online]. Available: https://doi.org/10.1007/978-3-662-47904-9

[16] T. Sandholm, K. Larson, M. Andersson, O. Shehory, and F. Tohmé, "Coalition structure generation with worst case guarantees," *Artificial intelligence*, vol. 111, no. 1-2, pp. 209–238, 1999.

[17] M. J. Osborne and A. Rubinstein, *A course in game theory*. MIT press, 1994.

[18] L. S. Shapley *et al.*, *A value for n-person games*. Princeton University Press Princeton, 1953.

[19] M. Davis and M. Maschler, "The kernel of a cooperative game," *Naval Research Logistics Quarterly*, vol. 12, no. 3, pp. 223–259, 1965.

[20] D. Schmeidler, "The nucleolus of a characteristic function game," *SIAM Journal on applied mathematics*, vol. 17, no. 6, pp. 1163–1170, 1969.

[21] D. Gale and L. S. Shapley, "College admissions and the stability of marriage," *The American Mathematical Monthly*, vol. 69, no. 1, pp. 9–15, 1962.

[22] A. E. Roth, "Deferred acceptance algorithms: History, theory, practice, and open questions," *International Journal of Game Theory*, vol. 36, pp. 537–569, 2008.

[23] M. J. Osborne *et al.*, *An introduction to game theory*. Oxford university press New York, 2004, vol. 3, no. 3.

[24] R. W. Irving, "An efficient algorithm for the "stable roommates" problem," *Journal of Algorithms*, vol. 6, no. 4, pp. 577–595, 1985.

[25] D. F. Manlove, *Hospitals/residents problem*. Springer, 2008.

[26] C. List, "Social Choice Theory," in *The Stanford Encyclopedia of Philosophy*, Winter 2022 ed., E. N. Zalta and U. Nodelman, Eds. Metaphysics Research Lab, Stanford University, 2022.

[27] K. J. Arrow, *Social choice and individual values*. Yale university press, 2012, vol. 12.

[28] M. J. Osborne and A. Rubinstein, *Models in Microeconomic Theory*. Open Book Publishers, 2020.

[29] A. Simonetto, J. Monteil, and C. Gambella, "Real-time city-scale ridesharing via linear assignment problems," *Transportation Research Part C: Emerging Technologies*, vol. 101, pp. 208–232, 2019.

[30] M. Stiglic, N. Agatz, M. Savelsbergh, and M. Gradisar, "The benefits of meeting points in ride-sharing systems," *Transportation Research Part B: Methodological*, vol. 82, pp. 36–53, 2015.

[31] D. J. Fagnant and K. M. Kockelman, "Dynamic ride-sharing and fleet sizing for a system of shared autonomous vehicles in austin, texas," *Transportation*, vol. 45, pp. 143–158, 2018.

[32] M. Nourinejad and M. J. Roorda, "Agent based model for dynamic ridesharing," *Transportation Research Part C: Emerging Technologies*, vol. 64, pp. 117–132, 2016.

[33] V. D. Dang, R. K. Dash, A. Rogers, and N. R. Jennings, "Overlapping coalition formation for efficient data fusion in multi-sensor networks," in *Association for the Advancement of Artificial Intelligence*, vol. 6, 2006, pp. 635–640.

[34] M. Binshtok, R. I. Brafman, S. E. Shimony, A. Martin, and C. Boutilier, "Computing optimal subsets," in *AAAI*, 2007, pp. 1231–1236.

[35] F. Bistaffa, A. Farinelli, and S. Ramchurn, "Sharing rides with friends: A coalition formation algorithm for ridesharing," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, no. 1, 2015.

[36] R. E. Stearns, "Convergent transfer schemes for n-person games," *Transactions of the American Mathematical Society*, vol. 134, no. 3, pp. 449–459, 1968.

[37] M. Klusch and O. Shehory, "A polynomial kernel-oriented coalition algorithm for rational information agents," *Tokoro, ed*, pp. 157–164, 1996.

[38] O. Shehory and S. Kraus, "Feasible formation of coalitions among autonomous agents in nonsuperadditive environments," *Computational Intelligence*, vol. 15, no. 3, pp. 218–251, 1999.